SIEMENS EDA

Tessent™ Shell Reference Manual

Software Version 2022.4
Document Revision 29

SIEMENS

Unpublished work. © 2022 Siemens

This Documentation contains trade secrets or otherwise confidential information owned by Siemens Industry Software Inc. or its affiliates (collectively, “Siemens”), or its licensors. Access to and use of this Documentation is strictly limited as set forth in Customer’s applicable agreement(s) with Siemens. This Documentation may not be copied, distributed, or otherwise disclosed by Customer without the express written permission of Siemens, and may not be used in any way not expressly authorized by Siemens.

This Documentation is for information and instruction purposes. Siemens reserves the right to make changes in specifications and other information contained in this Documentation without prior notice, and the reader should, in all cases, consult Siemens to determine whether any changes have been made.

No representation or other affirmation of fact contained in this Documentation shall be deemed to be a warranty or give rise to any liability of Siemens whatsoever.

If you have a signed license agreement with Siemens for the product with which this Documentation will be used, your use of this Documentation is subject to the scope of license and the software protection and security provisions of that agreement. If you do not have such a signed license agreement, your use is subject to the Siemens Universal Customer Agreement, which may be viewed at https://www.sw.siemens.com/en-US/sw-terms/base/uca/, as supplemented by the product specific terms which may be viewed at https://www.sw.siemens.com/en-US/sw-terms/supplements/.

SIEMENS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS DOCUMENTATION INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY. SIEMENS SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL OR PUNITIVE DAMAGES, LOST DATA OR PROFITS, EVEN IF SUCH DAMAGES WERE FORESEEABLE, ARISING OUT OF OR RELATED TO THIS DOCUMENTATION OR THE INFORMATION CONTAINED IN IT, EVEN IF SIEMENS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

TRADEMARKS: The trademarks, logos, and service marks (collectively, "Marks") used herein are the property of Siemens or other parties. No one is permitted to use these Marks without the prior written consent of Siemens or the owner of the Marks, as applicable. The use herein of third party Marks is not an attempt to indicate Siemens as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A list of Siemens' Marks may be viewed at: www.plm.automation.siemens.com/global/en/legal/trademarks.html. The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

About Siemens Digital Industries Software

Siemens Digital Industries Software is a leading global provider of product life cycle management (PLM) software and services with 7 million licensed seats and 71,000 customers worldwide. Headquartered in Plano, Texas, Siemens Digital Industries Software works collaboratively with companies to deliver open solutions that help them turn more ideas into successful products. For more information on Siemens Digital Industries Software products and services, visit www.siemens.com/plm.

Support Center: support.sw.siemens.com
Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Revision History ISO-26262

Revision	Changes	Status/ Date
29	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Tessent Release Notes* for this product are reflected in this document. Approved by Ron Press.	Released Dec 2022
28	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Tessent Release Notes* for this product are reflected in this document. Approved by Ron Press.	Released Sep 2022
27	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Tessent Release Notes* for this product are reflected in this document. Approved by Ron Press.	Released Jun 2022
26	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Tessent Release Notes* for this product are reflected in this document. Approved by Ron Press.	Released Mar 2022

Author: In-house procedures and working practices require multiple authors for documents. All associated authors for each topic within this document are tracked within the Siemens documentation source. For specific topic authors, contact Siemens Digital Industries Software documentation department.

Revision History: Released documents include a revision history of up to four revisions. For earlier revision history, refer to earlier releases of documentation on Support Center.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table of Contents

Revision History ISO-26262

Chapter 1
Introduction 59

Overview 59
Product Introduction 62
Tessent BoundaryScan 62
Tessent Diagnosis 63
Tessent FastScan and Tessent TestKompress 64
Tessent IJTAG 66
Tessent LogicBIST 67
Tessent MemoryBIST 67
Tessent MissionMode 68
Tessent Scan 69
Tessent ScanPro 70
Tessent SiliconInsight 70
Tessent DesignEditor 71

Chapter 2
Tool Invocations 73

Tool Invocation Command Descriptions 74
create_skeleton_design 75
stil2tessent 77
tessent 84
tessent -diagserver 85
tessent -shell 88
tessent -siliconinsight 92
tessent -visualizer 93
tessent -yieldinsight 95

Chapter 3
Command Dictionary (A-C) 97

Command Descriptions 98
add_atpg_constraints 106
add_atpg_functions 110
add_bist_capture_range 113
add_black_boxes 115
add_bsdl_extraction_user_instruction 119
add_cdp_test 121
add_cell_constraints 122
add_cell_models 129
add_chain_masks 131

add_clocks 136
add_config_element 147
add_config_message 152
add_config_tab 154
add_control_points 155
add_core_instances 157
add_design_partition 166
add_dft_clock_enables 168
add_dft_clock_mux 173
add_dft_control_points 178
add_dft_modal_connections 185
add_dft_signals 198
add_drc_violation 240
add_edt_blocks 247
add_edt_configurations 250
add_false_paths 252
add_fault_sites 259
add_faults 261
add_gate_to_rtl_mapping 275
add_icl_ports 277
add_icl_scan_interfaces 287
add_icl_ssn_datapaths 290
add_iddq_exceptions 292
add_ijtag_logical_connection 294
add_input_constraints 299
add_layout_core_information 306
add_lfsr_connections 308
add_lfsr_taps 310
add_lfsrs 312
add_lists 315
add_loadboard_loopback_pairs 318
add_misrs 320
add_nofaults 322
add_nonscan_instances 325
add_notest_points 328
add_observe_points 330
add_output_masks 332
add_pade_logical_connection 334
add_pattern_trigger 337
add_primary_inputs 338
add_primary_outputs 341
add_processors 343
add_read_controls 347
add_register_value 349
add_rtl_to_gates_mapping 351
add_scan_chains 358
add_scan_groups 361
add_scan_instances 364
add_scan_mode 366

add_scan_segments . . . 374
add_schematic_callout . . . 377
add_schematic_connections . . . 379
add_schematic_objects . . . 381
add_schematic_path . . . 383
add_simdut_fault . . . 385
add_simulation_context . . . 387
add_simulation_forces . . . 389
add_synchronous_clock_group . . . 391
add_tcl_variables_to_flat_model . . . 395
add_tied_signals . . . 397
add_to_collection . . . 399
add_wave_generator_pins . . . 401
add_write_controls . . . 402
analyze_atpg_constraints . . . 404
analyze_bus . . . 407
analyze_compression . . . 411
analyze_control_signals . . . 434
analyze_drc_violation . . . 437
analyze_fault . . . 439
analyze_graybox . . . 447
analyze_icl_suspect . . . 454
analyze_layout_hierarchy . . . 456
analyze_restrictions . . . 459
analyze_scan_chains . . . 462
analyze_simulation_mismatches . . . 464
analyze_test_points . . . 472
analyze_wrapper_cells . . . 479
analyze_xbounding . . . 483
annotate_diagnosis . . . 486
append_to_collection . . . 488
apply_specification_defaults . . . 490
assert_icl_sib . . . 493
catch_output . . . 495
check_design_rules . . . 497
check_synthesis . . . 499
check_testbench_simulations . . . 502
close_layout . . . 509
close_pattern_set . . . 510
close_tsdb . . . 521
close_visualizer . . . 523
collect_cdp_test_failure_logs . . . 524
compare_collections . . . 525
compress_layout . . . 527
compress_patterns . . . 528
convert_design_path_format . . . 531
copy_collection . . . 532
copy_module . . . 534
copy_simulation_context . . . 536

create_bisr_segment_order_file 538
create_capture_procedures 539
create_connections 543
create_custom_bisr_register 548
create_dft_specification 550
create_diagnosis_patterns 566
create_feature_statistics 568
create_flat_model 571
create_icl_flush_pattern 573
create_icl_setup_pattern 575
create_icl_verification_patterns 576
create_initialization_patterns 582
create_instance 584
create_layout 589
create_memory_repair_flow 598
create_module 600
create_net 603
create_open_socket_failure 605
create_patterns 606
create_patterns_specification 614
create_pin 622
create_port 624
create_reversible_patterns 626
create_scan_chain_family 628
create_scan_segment 634
create_silicon_insight_setup_specification 636
create_ssn_continuity_patterns 638

Chapter 4
Command Dictionary (D-F) 643

Command Descriptions 646
deassert_icl_sib 653
defer_schematic_updates 654
delete_atpg_constraints 655
delete_atpg_functions 657
delete_bist_capture_ranges 659
delete_black_boxes 660
delete_bsdl 662
delete_bsdl_extraction_option 663
delete_bsdl_extraction_user_instruction 665
delete_capture_procedures 666
delete_cdp_test 668
delete_cell_constraints 669
delete_cell_library 671
delete_cell_models 672
delete_chain_masks 674
delete_clocks 675
delete_config_element 676

delete_config_messages 678
delete_config_tabs 680
delete_connections 681
delete_core_descriptions 683
delete_core_instances 685
delete_design 688
delete_design_partitions 689
delete_dfm 690
delete_dft_clock_enables 692
delete_dft_clock_muxes 694
delete_dft_control_points 696
delete_dft_modal_connections 698
delete_dft_signals 701
delete_edt_blocks 703
delete_edt_configurations 706
delete_false_paths 708
delete_fault_sites 711
delete_faults 715
delete_flat_model 727
delete_gate_to_rtl_mapping 728
delete_icl_modules 729
delete_icl_ports 730
delete_icl_scan_interfaces 731
delete_icl_ssn_datapaths 732
delete_iddq_exceptions 733
delete_ijtag_logical_connection 734
delete_input_constraints 736
delete_instances 739
delete_iprocs 741
delete_layout_core_information 743
delete_layout_verification 744
delete_lfsr_connections 746
delete_lfsr_taps 747
delete_lfsrs 748
delete_lists 749
delete_loadboard_loopback_pairs 750
delete_misrs 752
delete_multicycle_paths 753
delete_nets 756
delete_nofaults 758
delete_nonscan_instances 761
delete_notest_points 763
delete_output_masks 766
delete_pade_logical_connection 768
delete_pattern_trigger 770
delete_patterns 771
delete_pins 773
delete_ports 775
delete_primary_inputs 777

delete_primary_outputs 779
delete_processors 781
delete_read_controls 784
delete_register_value 785
delete_rtl_to_gates_mapping 786
delete_scan_chain_families 788
delete_scan_chains 789
delete_scan_elements 790
delete_scan_groups 791
delete_scan_instances 792
delete_scan_modes 794
delete_schematic_callout 795
delete_schematic_objects 796
delete_schematic_path 798
delete_simdut_fault 799
delete_simulation_contexts 800
delete_simulation_forces 802
delete_synchronous_clock_group 804
delete_test_points 807
delete_tied_signals 809
delete_wave_generator_pins 811
delete_write_controls 812
diagnose_failures 813
display_diagnosis_report 818
display_message 819
display_specification 821
dofile 823
echo 825
elaborate_icl_modules 827
encrypt_layout 828
execute_cdp_test 830
execute_gpib_command 836
execute_tester_command 839
exit 841
expand_compressed_patterns 842
extract_fault_sites 843
extract_icl 850
extract_inter_cell_data 855
extract_sdc 862
filter_collection 864
find_design_names 866
foreach_in_collection 872
format_dictionary 875

Chapter 5
Command Dictionary (G-Q) 879

Command Descriptions 884
get_attribute_list 898

get_attribute_option 901
get_attribute_value_list 903
get_attributed_objects 907
get_auxiliary_pins 909
get_boundary_scan_port_option 912
get_bsdl_extraction_option 914
get_bundle_objects 916
get_capture_timing_option 919
get_burnin_option 923
get_cdp_test_data 924
get_cdp_test_list 926
get_clock_option 927
get_clocks 932
get_common_parent_instance 935
get_config_elements 936
get_config_messages 940
get_config_value 943
get_context 958
get_current_design 962
get_current_mode 964
get_current_physical_block 965
get_current_silicon_insight_setup 966
get_current_simulation_context 967
get_default_ssn_datapath_configuration 968
get_defaults_value 969
get_design_level 971
get_design_objects 972
get_design_sources 975
get_dfm_rules 977
get_dft_cell 979
get_dft_cell_selection_mapping 989
get_dft_info_dictionary 990
get_dft_signal 997
get_dft_signals_option 1004
get_dft_specification_requirement 1005
get_drc_handling 1007
get_equivalent_editable_node 1008
get_fanins 1010
get_fanouts 1014
get_fault_type 1018
get_flat_model_option 1019
get_gate_pins 1020
get_icl_extraction_options 1026
get_icl_fanins 1027
get_icl_fanins_in_module 1029
get_icl_fanouts 1031
get_icl_fanouts_in_module 1033
get_icl_instances 1035
get_icl_modules 1040

get_icl_module_parameter_list . . . 1043
get_icl_module_parameter_value . . . 1044
get_icl_network . . . 1046
get_icl_objects . . . 1048
get_icl_pins . . . 1055
get_icl_ports . . . 1059
get_icl_scan_interface_chain_list . . . 1062
get_icl_scan_interface_list . . . 1063
get_icl_scan_interface_port_list . . . 1065
get_icl_scan_mux . . . 1068
get_icl_scope . . . 1070
get_icl_ssn_datapath_list . . . 1072
get_icl_ssn_datapath_ports . . . 1073
get_iclock_list . . . 1075
get_iclock_option . . . 1077
get_ijtag_instances . . . 1081
get_ijtag_instance_option . . . 1083
get_ijtag_retargeting_options . . . 1085
get_input_constraints . . . 1088
get_insertion_option . . . 1092
get_insert_test_logic_option . . . 1095
get_instance_polygons . . . 1098
get_instances . . . 1099
get_instrument_dictionary . . . 1106
get_instrument_parent_icl_module_list . . . 1112
get_iproc_argument_default . . . 1113
get_iproc_argument_list . . . 1115
get_iproc_body . . . 1116
get_iproc_list . . . 1117
get_iproc_option . . . 1118
get_layout_core_instance . . . 1120
get_license_queue_timeout . . . 1121
get_load_unload_timing_option . . . 1122
get_loadboard_loopback_option . . . 1127
get_logfile . . . 1128
get_logical_library_list . . . 1129
get_lvx_option . . . 1130
get_master_module_name . . . 1131
get_memory_instances . . . 1132
get_memory_instance_option . . . 1135
get_module_matching_options . . . 1138
get_module_parameter_list . . . 1142
get_module_parameter_value . . . 1144
get_modules . . . 1146
get_multiprocessing_option . . . 1153
get_name_list . . . 1155
get_nets . . . 1159
get_net_name_list . . . 1164
get_open_pattern_set . . . 1165

get_pattern_cycle_count 1166
get_pattern_set_data 1167
get_pattern_set_list 1177
get_pattern_set_option 1178
get_pin_name_list 1180
get_pins 1181
get_port_name_list 1192
get_ports 1193
get_procfile_name 1203
get_read_verilog_option 1204
get_resource 1205
get_run_synthesis_options 1207
get_scan_chain_families 1209
get_scan_chain_option 1210
get_scan_elements 1211
get_scan_modes 1218
get_scratch_directory 1219
get_silicon_insight_job_status 1220
get_silicon_insight_option 1221
get_simdut_data 1222
get_simulation_context_list 1224
get_simulation_library_sources 1226
get_simulation_option 1228
get_simulation_value_list 1230
get_single_attribute_value 1232
get_single_name 1234
get_ssn_datapath_option 1237
get_ssn_option 1239
get_static_dft_signal_icall 1240
get_synchronous_clock_groups 1242
get_system_mode 1243
get_tcl_shell_option 1244
get_test_end_icall_list 1245
get_test_point_analysis_options 1246
get_test_point_insertion_options 1250
get_test_point_sharing_restrictions 1252
get_test_point_type 1254
get_test_points 1255
get_test_setup_icall_list 1260
get_testbench_simulation_options 1261
get_tester_option 1263
get_timeplate_list 1264
get_tool_info 1265
get_tool_option 1267
get_trace_flat_model_option 1268
get_transcript_style 1271
get_tsdb_info 1273
get_tsdb_list 1279
get_tsdb_option 1280

get_tsdb_output_directory 1281
get_validated_objects 1282
get_write_patterns_options 1285
get_xclock_handling 1288
get_xy_net 1289
get_xy_instance 1290
help 1291
history 1293
iApply 1295
iCall 1298
iClock 1302
iClockOverride 1304
iComparePort 1307
identify_redundant_faults 1310
iForcePort 1314
iMerge 1318
import_clocks 1320
import_dfm 1321
import_patterns_from_svf 1323
import_scan_mode 1326
index_collection 1329
iNote 1331
insert_test_logic 1337
intercept_connection 1340
iOverrideScanInterface 1348
iPDLLevel 1351
iPrefix 1352
iProc 1354
iProcsForModule 1356
iPulseClock 1358
iRead 1360
iRelease 1362
iReset 1363
iResumePdlRetargeting 1365
iRunLoop 1367
is_collection 1370
is_valid_object_name 1372
iScan 1373
isolate_icl_node 1377
iState 1378
iSuspendPdlRetargeting 1381
iTake 1385
iTopProc 1387
iUseProcNameSpace 1389
iWrite 1391
launch_sid_tester 1393
lock_current_registration 1395
macrotest 1396
map_tsdb_container_paths 1403

merge_cdp . . . 1405
migrate_layout . . . 1406
move_config_element . . . 1407
move_connections . . . 1412
no_transcript . . . 1420
open_layout . . . 1423
open_pattern_set . . . 1427
open_tsdb . . . 1438
open_visualizer . . . 1440
order_patterns . . . 1444
printenv . . . 1448
process_atpg_failures . . . 1449
process_dft_specification . . . 1451
process_patterns_specification . . . 1456
process_top_module_connections . . . 1461

Chapter 6
Command Dictionary (Ra-Report_i) . . . 1467

Command Descriptions . . . 1470
range_collection . . . 1477
read_bsdl . . . 1479
read_cell_library . . . 1480
read_config_data . . . 1481
read_core_descriptions . . . 1484
read_cpf . . . 1486
read_def . . . 1489
read_design . . . 1492
read_failures . . . 1499
read_fault_sites . . . 1507
read_faults . . . 1519
read_flat_model . . . 1530
read_icl . . . 1534
read_lef . . . 1536
read_liberty . . . 1537
read_modelfile . . . 1539
read_patterns . . . 1543
read_procfile . . . 1555
read_sdc . . . 1559
read_sdf . . . 1566
read_upf . . . 1569
read_verilog . . . 1571
read_vhdl . . . 1578
register_attribute . . . 1582
register_callback . . . 1586
register_drc . . . 1596
register_drc_class . . . 1631
register_static_dft_signal_names . . . 1632
register_tcl_command . . . 1636

remove_from_collection . . . 1650
rename_instance . . . 1652
rename_module . . . 1654
rename_net . . . 1655
rename_pin . . . 1657
rename_port . . . 1658
replace_instances . . . 1660
report_aborted_faults . . . 1662
report_atpg_constraints . . . 1665
report_atpg_functions . . . 1666
report_atpg_timing . . . 1667
report_attributes . . . 1672
report_bisr_repair_register_icl_instances . . . 1680
report_bist_capture_ranges . . . 1681
report_black_boxes . . . 1682
report_boundary_scan_port_options . . . 1684
report_bsdl . . . 1686
report_bsdl_extraction_options . . . 1687
report_bsdl_extraction_user_instructions . . . 1688
report_bus_data . . . 1689
report_bypass_chains . . . 1692
report_capture_procedures . . . 1695
report_capture_timing_options . . . 1701
report_cell_constraints . . . 1703
report_cell_models . . . 1706
report_chain_masks . . . 1708
report_clock_controller_pins . . . 1709
report_clock_controls . . . 1710
report_clock_domains . . . 1712
report_clock_gating . . . 1717
report_clocks . . . 1719
report_compactor_connections . . . 1724
report_config_data . . . 1727
report_config_messages . . . 1732
report_config_syntax . . . 1735
report_context . . . 1739
report_control_signals . . . 1741
report_core_descriptions . . . 1743
report_core_instance_parameters . . . 1746
report_core_instances . . . 1748
report_core_parameters . . . 1752
report_current_design . . . 1754
report_default_ssn_datapath_configuration . . . 1756
report_design_partitions . . . 1757
report_design_sources . . . 1758
report_dfm_rules . . . 1760
report_dft_cell_selection_mapping . . . 1762
report_dft_clock_enables . . . 1763
report_dft_clock_muxes . . . 1765

report_dft_control_points 1766
report_dft_modal_connections 1768
report_dft_signal_names 1772
report_dft_signals 1773
report_diagnosis 1775
report_drc_rules 1776
report_edt_abort_analysis 1784
report_edt_blocks 1799
report_edt_configurations 1800
report_edt_finder 1806
report_edt_instances 1811
report_edt_lockup_cells 1814
report_edt_pins 1819
report_environment 1822
report_external_simulator 1824
report_failures 1825
report_false_paths 1831
report_fault_sites 1840
report_faults 1848
report_feedback_paths 1865
report_flattener_rules 1867
report_gates 1869
report_gate_to_rtl_mapping 1894
report_graybox_statistics 1895
report_icl_extraction_options 1897
report_icl_modules 1898
report_icl_network 1902
report_iclock 1904
report_id_stamp 1906
report_iddq_exceptions 1908
report_ijtag_instances 1909
report_ijtag_logical_connections 1911
report_ijtag_retargeting_options 1913
report_input_constraints 1915
report_insert_test_logic_options 1918
report_instance_polygons 1920
report_iprocs 1921

Chapter 7
Command Dictionary (Report_j-R) 1923

Command Descriptions 1926
report_layout_core_information 1933
report_layout_files 1935
report_layout_hierarchy 1937
report_layout_rules 1939
report_lbist_configuration 1944
report_lbist_pins 1946
report_lfsr_connections 1948

report_lfsrs 1949
report_licenses 1950
report_lists 1951
report_load_unload_timing_options 1952
report_loadboard_loopback_pairs 1954
report_loops 1955
report_lpct_configuration 1957
report_lpct_pins 1958
report_measure_cycles 1960
report_memory_cluster_configuration 1965
report_memory_dft_summary 1971
report_memory_identification 1979
report_memory_instances 1982
report_memory_repair_groups 1985
report_mismatch_sources 1987
report_misrs 1991
report_misr_connections 1993
report_module_matching 1995
report_module_matching_options 1999
report_multicycle_paths 2001
report_multiprocessing_options 2006
report_net_polygons 2008
report_nofaults 2009
report_nonscan_cells 2011
report_nonscan_models 2014
report_notest_points 2015
report_output_masks 2017
report_pade_logical_connections 2018
report_pattern_filtering 2020
report_pattern_sets 2021
report_pattern_trigger 2023
report_patterns 2024
report_power_data 2035
report_power_metrics 2037
report_power_control 2044
report_primary_inputs 2048
report_primary_outputs 2050
report_procedures 2052
report_processors 2056
report_read_controls 2059
report_read_verilog_options 2060
report_register_value 2061
report_resources 2064
report_rtl_to_gates_mapping 2065
report_run_synthesis_options 2067
report_scan_cells 2070
report_scan_chains 2082
report_scan_elements 2088
report_scan_enable 2091

report_scan_groups . . . 2093
report_scan_models . . . 2094
report_scan_modes . . . 2095
report_scan_path . . . 2096
report_scan_polygons . . . 2109
report_scan_segments . . . 2113
report_scan_volume . . . 2116
report_seq_transparent_procedures . . . 2120
report_sequential_fault_depth . . . 2122
report_shift_registers . . . 2124
report_silicon_insight_result . . . 2127
report_simdut_data . . . 2129
report_simdut_faults . . . 2131
report_simulation_contexts . . . 2132
report_simulation_forces . . . 2134
report_simulation_library_sources . . . 2136
report_ssn_configuration . . . 2137
report_static_dft_signal_settings . . . 2139
report_statistics . . . 2141
report_synchronous_clock_groups . . . 2161
report_tcd_ldb_validation . . . 2163
report_tcl_shell_options . . . 2165
report_test_end_icall . . . 2166
report_test_logic . . . 2167
report_test_points . . . 2168
report_test_point_statistics . . . 2172
report_test_setup_icall . . . 2174
report_test_stimulus . . . 2175
report_testbench_simulation_options . . . 2184
report_tester_options . . . 2185
report_tester_statistics . . . 2186
report_tied_signals . . . 2188
report_timeplates . . . 2190
report_udfm_statistics . . . 2192
report_version_data . . . 2198
report_wrapper_cells . . . 2199
report_write_controls . . . 2203
report_write_patterns_options . . . 2204
report_xbounding . . . 2206
report_xy_instance . . . 2209
report_xy_net . . . 2210
reset_attribute_value . . . 2211
reset_au_faults . . . 2213
reset_bypass_chains . . . 2215
reset_capture_timing_options . . . 2216
reset_compactor_connections . . . 2217
reset_default_ssn_datapath_configuration . . . 2218
reset_design . . . 2219
reset_di_faults . . . 2222

reset_ijtag_retargeting_options . . . 2225
reset_load_unload_timing_options . . . 2228
reset_open_pattern_set . . . 2229
reset_physical_block_settings . . . 2230
reset_state . . . 2231
reset_static_dft_signal_values . . . 2232
restore_design . . . 2235
run_synthesis . . . 2236
run_testbench_simulations . . . 2245

Chapter 8
Command Dictionary (S) . . . 2267

Command Descriptions . . . 2272
set_abort_limit . . . 2284
set_atpg_fill . . . 2286
set_atpg_limits . . . 2287
set_atpg_timing . . . 2290
set_attribute_options . . . 2294
set_attribute_value . . . 2299
set_au_analysis . . . 2305
set_bidi_gating . . . 2306
set_bist_chain_test . . . 2309
set_bist_debug . . . 2310
set_bist_trace . . . 2314
set_boundary_scan_port_options . . . 2317
set_bsdl_extraction_options . . . 2319
set_burnin_options . . . 2321
set_bus_handling . . . 2323
set_bus_simulation . . . 2326
set_bypass_chains . . . 2328
set_capture_clock . . . 2331
set_capture_procedures . . . 2334
set_capture_timing_options . . . 2336
set_cdp_test_data . . . 2340
set_cdp_test_diagnosis_option . . . 2342
set_cell_library_options . . . 2344
set_chain_test . . . 2351
set_checkpointing_options . . . 2355
set_clock_controller_pins . . . 2357
set_clock_controls . . . 2361
set_clock_gating . . . 2364
set_clock_gating_enable . . . 2367
set_clock_gating_options . . . 2369
set_clock_off_simulation . . . 2371
set_clock_options . . . 2372
set_clock_restriction . . . 2375
set_compactor_connections . . . 2385
set_config_value . . . 2387

set_contention_check 2391
set_context 2396
set_core_instance_parameters 2402
set_critical_area_options 2410
set_current_design 2415
set_current_edt_block 2424
set_current_edt_configuration 2427
set_current_mode 2429
set_current_physical_block 2431
set_current_silicon_insight_setup 2432
set_current_simulation_context 2434
set_dedicated_wrapper_cell_options 2436
set_default_ssn_datapath_configuration 2441
set_defaults_value 2443
set_design_include_directories 2447
set_design_level 2449
set_design_macros 2455
set_design_sources 2457
set_dft_cell_selection_mapping 2461
set_dft_enable_options 2462
set_dft_signals_options 2464
set_dft_specification_requirements 2466
set_diagnosis_options 2473
set_display 2487
set_drc_handling 2489
set_driver_restriction 2498
set_edt_abort_analysis_options 2499
set_edt_finder 2500
set_edt_instances 2502
set_edt_mapping 2505
set_edt_options 2508
set_edt_pins 2521
set_edt_power_controller 2533
set_external_capture_options 2537
set_external_simulator 2540
set_fails_report 2542
set_failure_mapping_options 2543
set_fault_mode 2545
set_fault_sampling 2547
set_fault_selection_options 2549
set_fault_subclass_analysis 2550
set_fault_type 2553
set_flat_model_options 2562
set_flattener_rule_handling 2567
set_gate_level 2570
set_gate_report 2571
set_gzip_options 2606
set_icl_extraction_options 2608
set_icl_network 2610

set_icl_scan_interface_ports 2612
set_icl_scan_mux 2616
set_icl_ssn_datapath_ports 2617
set_iddq_checks 2621
set_ijtag_instance_options 2627
set_ijtag_retargeting_options 2629
set_insertion_options 2642
set_insert_test_logic_options 2651
set_internal_fault 2656
set_internal_name 2657
set_io_mask 2658
set_latch_handling 2659
set_layout_core_instance 2660
set_lbist_controller_options 2661
set_lbist_instances 2666
set_lbist_pins 2668
set_lbist_power_controller_options 2674
set_learn_report 2676
set_lfsr_seed 2678
set_lfsrs 2679
set_license_queue_timeout 2681
set_list_file 2683
set_load_unload_timing_options 2684
set_logfile_handling 2693
set_logical_design_libraries 2695
set_loop_handling 2697
set_lpct_condition_bits 2699
set_lpct_controller 2701
set_lpct_instances 2705
set_lpct_pins 2706
set_lvx_options 2714
set_macrotest_options 2716
set_marker_file_options 2718
set_memory_cluster_library_generation_options 2721
set_memory_cluster_validation_options 2724
set_memory_identification_options 2727
set_memory_instance_options 2729
set_misr_connections 2733
set_module_matching_options 2735
set_multiple_detection 2739
set_multiprocessing_options 2743
set_net_dominance 2752
set_net_resolution 2754
set_number_shifts 2755
set_observation_point 2757
set_output_masks 2758
set_parallel_load_subchains 2761
set_pathdelay_holdpi 2763
set_pattern_buffer 2764

set_pattern_classification 2765
set_pattern_filtering 2766
set_pattern_source 2778
set_pattern_type 2787
set_physical_translation 2791
set_possible_credit 2793
set_power_control. 2794
set_power_metrics 2803
set_procedure_cycle_checking 2807
set_procedure_retargeting_options 2808
set_procfile_name. 2811
set_quick_synthesis_options. 2813
set_ram_initialization 2817
set_random_clocks 2818
set_random_patterns 2820
set_read_verilog_options 2821
set_register_value 2823
set_relevant_coverage. 2825
set_run_synthesis_options 2829
set_scan_chain_options 2832
set_scan_enable 2834
set_scan_insertion_options 2838
set_scan_signals 2845
set_screen_display 2850
set_shadow_check 2851
set_shift_register_identification 2853
set_silicon_insight_option 2855
set_simulation_library_sources 2857
set_simulation_options 2860
set_skewed_load 2865
set_split_capture_cycle 2866
set_ssn_datapath_options 2867
set_ssn_options 2869
set_stability_check 2873
set_static_dft_signal_values 2877
set_static_learning 2881
set_system_mode 2883
set_tcl_shell_options. 2885
set_test_end_icall 2888
set_test_logic 2891
set_test_point_analysis_options 2893
set_test_point_insertion_options 2906
set_test_point_sharing_restrictions. 2909
set_test_point_types 2911
set_test_setup_icall 2913
set_testbench_simulation_options 2918
set_tester_options 2921
set_tied_signals 2923
set_timing_exceptions_handling 2925

set_tla_loop_handling . . . 2932
set_tool_options . . . 2934
set_trace_flat_model_options . . . 2936
set_trace_report . . . 2939
set_transcript_style . . . 2940
set_transient_detection . . . 2943
set_transition_holdpi . . . 2945
set_tristate_gating . . . 2946
set_tsdb_options . . . 2948
set_tsdb_output_directory . . . 2950
set_visualizer_preferences . . . 2952
set_wrapper_analysis_options . . . 2953
set_write_patterns_options . . . 2959
set_xbounding_options . . . 2964
set_xclock_handling . . . 2968
set_z_handling . . . 2970
set_zhold_behavior . . . 2972
setenv . . . 2973
shmoo_cdp_test . . . 2974
shutdown_sid_tester . . . 2978
simulate_clock_pulses . . . 2979
simulate_forces . . . 2981
simulate_patterns . . . 2983
sizeof_collection . . . 2986
sort_collection . . . 2988
start_silicon_insight . . . 2990
stop_silicon_insight . . . 2991
system . . . 2992

Chapter 9
Command Dictionary (T-Z) . . . 2995

Command Descriptions . . . 2997
trace_flat_model . . . 3000
uncompress_layout . . . 3009
uniquify_instances . . . 3010
unregister_attribute . . . 3012
unregister_static_dft_signal_names . . . 3014
unregister_tcl_command . . . 3015
unsetenv . . . 3016
update_icl_attributes_from_design . . . 3017
update_implication_detections . . . 3019
verify_patterns . . . 3021
verify_tcd_scan . . . 3025
write_atpg_setup . . . 3027
write_cell_library . . . 3030
write_config_data . . . 3034
write_core_description . . . 3037
write_core_timing_constraints . . . 3039

write_design . . . 3040
write_design_import_script . . . 3052
write_design_source_dictionary . . . 3054
write_diagnosis . . . 3056
write_edt_files . . . 3059
write_failing_paths . . . 3071
write_failures . . . 3074
write_fault_sites . . . 3079
write_faults . . . 3083
write_flat_model . . . 3096
write_gds . . . 3100
write_icl . . . 3101
write_loops . . . 3103
write_memory_repair_dictionary . . . 3104
write_modelfile . . . 3108
write_patterns . . . 3109
write_pdl . . . 3140
write_primary_inputs . . . 3142
write_primary_outputs . . . 3143
write_procedure_testbench . . . 3144
write_procfile . . . 3148
write_scan_order . . . 3150
write_scan_setup . . . 3157
write_schematic_dofile . . . 3159
write_test_point_dofile . . . 3160
write_tsdb_data . . . 3162

Chapter 10
Design Rule Checking . . . 3167

Design Rule Overview . . . 3169
Scan Insertion Checking . . . 3169
How to Troubleshoot Rules Violations . . . 3170
Rules Settings . . . 3170
How to Turn on ATPG Analysis . . . 3170
How to Debug DRC Violations in Tessent Visualizer . . . 3170
How to Set the Level of Gate Data . . . 3171
How to Set the Gate Information Type . . . 3172
How to Report Gate Data . . . 3173
Flattening Rule Violations . . . 3176
Design Rule Output . . . 3176
Design Rules . . . 3177
RAM Rules (A Rules) . . . 3191
A1 . . . 3191
A2 . . . 3192
A3 . . . 3193
A4 . . . 3193
A5 . . . 3194
A6 . . . 3195

A7 3195
A9 3196
A10 3197
A11 3197
A12 3198
A13 3199
A14 3200
A15 3200
A16 3201
A17 3201
A18 3202
Assertion Rules (ASSERT Rules) 3205
Assertion Values and the Enforcement of Assertions 3205
ASSERT1 3205
ASSERT2 3207
ASSERT3 3209
BIST Rules (B Rules) 3211
B2 3211
B4 3211
B5 3212
B6 3212
B7 3214
B17 3216
BSCAN Rules 3218
BSCAN1 3218
BSCAN2 3219
BSCAN3 3219
BSCAN4 3219
BSCAN5 3220
BSCAN6 3220
BSCAN7 3220
BSCAN8 3220
BSCAN9 3221
Clock Rules (C Rules) 3222
Clock Terminology 3222
The ATPG Analysis Option 3227
C1 3227
C2 3232
C3 3233
C4 3239
C5 3244
C6 3246
C7 3248
C8 3249
C9 3250
C10 3252
C16 3253
C17 3254
C19 3255

C21 3257
C22 3257
C23 3260
C24 3261
C25 3262
C26 3263
C27 3265
Scan Cell Data Rules (D Rules) 3267
D1 3267
D2 3271
D3 3272
D4 3273
D5 3274
D6 3275
D7 3276
D8 3277
D12 3278
Pre-DFT Clock Rules (DFT_C Rules) 3280
Simulation Contexts for Pre-DFT DRCs 3280
DFT_C1 3284
DFT_C2 3287
DFT_C3 3289
DFT_C4 3290
DFT_C5 3290
DFT_C6 3291
DFT_C7 3293
DFT_C8 3294
DFT_C9 3295
DFT_C10 3299
DFT_C11 3306
DFT_C12 3307
DFT_C13 3308
Pre-DFT Scan Rules (DFT_S Rules) 3310
DFT_S1 3310
Extra Rules (E Rules) 3312
E1 3312
E2 3313
E3 3313
E4 3314
E5 3317
E6 3319
E7 3320
E8 3321
E9 3322
E10 3322
E11 3327
E12 3328
E13 3328
E14 3329

E15 3333
E16 3333
EDT Finder Rules (F Rules) 3336
F3 3336
F4 3337
F5 3337
F6 3338
F7 3340
F8 3340
F9 3346
F10 3347
F11 3349
F12 3350
F13 3350
F14 3351
F15 3352
F16 3352
F17 3353
F18 3353
F19 3354
F20 3355
F21 3357
F22 3357
Flattening Rules (FN, FP, and FG Rules) 3361
FN1 3362
FN2 3362
FN3 3362
FN4 3362
FN5 3363
FN6 3363
FN7 3363
FN8 3363
FN9 3364
FP1 3364
FP2 3364
FP3 3364
FP4 3364
FP5 3365
FP6 3365
FP7 3365
FP8 3365
FP9 3366
FP10 3366
FP11 3366
FP12 3366
FP13 3366
FG3 3367
FG7 3367
FG8 3367

General Rules (G Rules) 3368
G1 3368
G2 3368
G3 3369
G4 3369
G5 3369
G6 3370
G7 3370
G8 3370
G9 3371
G10 3371
G11 3372
G12 3372
ICL Extraction Rules (I Rules) 3373
I1 3373
I2 3375
I3 3376
I4 3377
I5 3378
I6 3379
I7 3381
ICL Semantic Rules (ICL Rules) 3383
ICL1 3387
ICL2 3388
ICL3 3388
ICL4 3389
ICL5 3389
ICL6 3389
ICL7 3389
ICL8 3390
ICL9 3390
ICL10 3390
ICL11 3391
ICL12 3391
ICL13 3391
ICL14 3391
ICL15 3392
ICL16 3392
ICL17 3392
ICL18 3392
ICL19 3393
ICL20 3393
ICL21 3393
ICL22 3394
ICL23 3394
ICL24 3394
ICL25 3394
ICL26 3395
ICL27 3395

ICL28 3395
ICL29 3395
ICL30 3396
ICL31 3396
ICL32 3396
ICL33 3396
ICL34 3397
ICL35 3397
ICL36 3397
ICL37 3398
ICL38 3398
ICL39 3398
ICL40 3399
ICL41 3399
ICL42 3399
ICL44 3399
ICL45 3400
ICL48 3400
ICL49 3400
ICL50 3401
ICL51 3401
ICL52 3402
ICL53 3402
ICL54 3402
ICL55 3402
ICL56 3403
ICL57 3403
ICL58 3404
ICL59 3404
ICL60 3405
ICL61 3405
ICL62 3405
ICL63 3405
ICL64 3406
ICL65 3406
ICL66 3407
ICL67 3407
ICL68 3407
ICL70 3408
ICL71 3408
ICL75 3411
ICL76 3411
ICL77 3412
ICL78 3412
ICL79 3413
ICL80 3413
ICL81 3417
ICL82 3418
ICL83 3418

ICL84 3418
ICL85 3419
ICL86 3419
ICL87 3419
ICL88 3419
ICL89 3420
ICL90 3421
ICL91 3421
ICL92 3422
ICL93 3422
ICL94 3422
ICL95 3422
ICL96 3423
ICL97 3423
ICL98 3423
ICL99 3424
ICL100 3424
ICL101 3424
ICL102 3425
ICL103 3425
ICL104 3425
ICL105 3426
ICL106 3427
ICL107 3427
ICL108 3427
ICL109 3428
ICL110 3428
ICL111 3428
ICL112 3428
ICL113 3429
ICL114 3429
ICL115 3429
ICL116 3430
ICL117 3430
ICL118 3430
ICL119 3431
ICL120 3431
ICL121 3431
ICL122 3432
ICL123 3432
ICL124 3432
ICL125 3433
ICL126 3433
ICL127 3434
ICL128 3434
ICL129 3434
ICL130 3435
ICL132 3435
ICL133 3436

ICL134 3436
ICL135 3437
ICL136 3438
ICL137 3439
ICL138 3439
ICL139 3440
ICL140 3440
ICL141 3442
ICL142 3443
ICL143 3443
ICL144 3443
EDT Rules (K Rules) 3445
K1 3446
K2 3446
K3 3446
K4 3447
K5 3447
K6 3447
K7 3449
K8 3449
K9 3449
K10 3451
K11 3451
K12 3452
K13 3452
K14 3452
K15 3453
K16 3458
K17 3458
K18 3458
K19 3459
K20 3460
K21 3461
K22 3461
K23 3463
K24 3463
K25 3464
Procedure Rules (P Rules) 3466
P1 3468
P2 3469
P3 3469
P4 3470
P5 3470
P6 3470
P7 3471
P8 3471
P9 3472
P10 3472
P11 3472

P12 3473
P13 3474
P14 3474
P15 3475
P16 3475
P17 3475
P18 3476
P19 3476
P20 3476
P21 3477
P22 3477
P23 3478
P24 3478
P25 3478
P26 3479
P27 3479
P28 3480
P29 3480
P30 3481
P31 3481
P32 3482
P33 3482
P34 3483
P35 3483
P36 3483
P37 3484
P38 3484
P39 3484
P40 3485
P41 3485
P42 3485
P43 3486
P44 3486
P45 3487
P46 3487
P47 3488
P48 3488
P49 3488
P50 3489
P51 3489
P52 3489
P53 3490
P54 3491
P55 3491
P56 3492
P58 3492
P59 3492
P60 3493
P61 3494

P62 3494
P63 3495
P64 3495
P65 3495
P66 3496
P70 3496
P71 3497
P72 3497
P73 3497
P74 3498
P75 3498
P76 3499
P77 3500
P78 3500
P79 3500
P80 3501
P81 3502
P82 3502
P83 3503
P84 3504
P85 3504
P86 3504
P87 3505
P88 3505
P89 3506
P90 3506
P91 3507
P92 3507
P93 3508
P94 3508
P96 3511
P97 3513
Core Mapping for ATPG and Scan Pattern Retargeting Rules (R Rules) 3514
R1 3515
R2 3515
R3 3515
R4 3516
R5 3516
R6 3517
R7 3518
R8 3518
R9 3521
R10 3522
R11 3523
R12 3524
R13 3525
R14 3525
R15 3527
R16 3530

R17 3532
R18 3532
R19 3533
R20 3534
R21 3534
R22 3535
R23 3536
R27 3536
R28 3537
Scannability Rules (S Rules) 3539
S1 3540
S2 3542
S3 3542
S4 3543
S5 3544
S6 3545
S7 3546
S8 3547
Scan-Wrapper Rules (SW Rules) 3549
SW1 3549
SW2 3550
SW3 3550
SW4 3550
SW5 3551
SW6 3552
SW7 3552
SSN ICL Rules (SSN_ICL Rules) 3554
SSN_ICL2 3554
SSN_ICL3 3555
SSN Retargeting Rules (SSN_R Rules) 3557
SSN_R1 3557
SSN_R2 3558
SSN_R3 3558
SSN_R4 3559
SSN_R5 3562
SSN_R6 3564
SSN_R7 3565
SSN_R8 3567
SSN_R9 3570
SSN_R10 3571
SSN_R11 3573
SSN_R12 3576
SSN_R13 3578
SSN_R15 3579
SSN_R16 3580
SSN_R17 3581
SSN_R18 3582
SSN_R19 3584
SSN_R20 3585

SSN_R21 3586
SSN_R23 3586
Scan Chain Trace Rules (T Rules) 3587
T1 3588
T2 3588
T3 3589
T4 3593
T5 3597
T6 3599
T8 3600
T9 3600
T10 3601
T11 3601
T12 3601
T13 3602
T14 3602
T15 3602
T16 3603
T17 3603
T18 3604
T19 3604
T20 3604
T21 3605
T22 3605
T23 3606
T24 3606
T25 3607
T26 3608
T27 3609
Power-Aware Rules (V Rules) 3610
V1 3611
V2 3612
V3 3612
V4 3612
V5 3613
V6 3613
V7 3613
V8 3614
V9 3614
V10 3615
V11 3616
V12 3617
V13 3618
V14 3619
V15 3619
V16 3620
V17 3620
V18 3620
V19 3621

V20 . . . 3621
V21 . . . 3622
Timing Rules (W Rules) . . . 3623
W1 . . . 3624
W2 . . . 3624
W3 . . . 3625
W4 . . . 3625
W5 . . . 3625
W6 . . . 3626
W7 . . . 3626
W8 . . . 3627
W9 . . . 3627
W10 . . . 3627
W11 . . . 3628
W12 . . . 3628
W13 . . . 3629
W14 . . . 3629
W15 . . . 3629
W18 . . . 3630
W19 . . . 3630
W20 . . . 3631
W21 . . . 3631
W22 . . . 3632
W23 . . . 3632
W24 . . . 3632
W25 . . . 3633
W26 . . . 3633
W27 . . . 3634
W28 . . . 3634
W29 . . . 3635
W30 . . . 3635
W31 . . . 3636
W32 . . . 3636
W33 . . . 3637
W34 . . . 3638
W35 . . . 3638
W36 . . . 3639
W37 . . . 3639
W38 . . . 3640
W39 . . . 3641
W41 . . . 3642
W42 . . . 3642
W43 . . . 3643
X-bounding Rules (XB Rules) . . . 3645
XB1 . . . 3645
XB2 . . . 3646
Other DRC Messages . . . 3647
Oscillation Limitation . . . 3647
RAM Summary Results and Test Capability . . . 3647

Chapter 11
Data Models 3649
Hierarchical Design Data Model 3651
Module 3654
Instance 3668
Port 3683
port_bundle 3704
Pin 3710
pin_bundle 3720
Net 3728
net_bundle 3736
Bundle Examples 3742
pseudo_port 3745
Flat Design Data Model 3749
Flat Design Object Types 3750
Gate_pin 3750
ICL Data Model 3764
ICL Object Types 3766
icl_instance 3766
icl_module 3770
icl_pin 3781
icl_port 3784
ICL File Examples With Complex Ports 3806
Example of Complex Ports as Functional Clocks or Input Port Functions 3806
Example of Grouping ICL Port Definitions for Modeling 3808
ICL Model Attribute Name Examples 3811
Scan Data Model 3813
Scan Element Object Type 3813
Scan Chain Family Object Type 3820
Scan Mode Object Type 3820
Test Point Data Model 3821
Configuration Data Model 3823
config_csv_wrapper 3823
config_data_wrapper 3824
config_property 3825
config_repeatable_property 3825
config_wrapper 3826

Chapter 12
Attributes 3829
Attribute-Related Commands 3829
Attribute Types 3831
Pre-Defined Attributes 3831
User-Defined Attributes 3831
Attribute Value Types 3831
Attribute Filtering Equation Syntax 3832
Glob and Regular Expression Pattern Matching Syntax 3834
Hierarchical Name Matching With Escaped Identifiers 3838

Attribute Inheritance Behavior 3840
Inheritance Rules 3840

Chapter 13
Configuration-Based Specification 3843

DftSpecification Configuration Syntax 3846
DftSpecification 3848
EDT 3860
IjtagNetwork. 3927
InSystemTest 4060
EmbeddedBoundaryScan 4093
BoundaryScan. 4153
LogicBist 4217
LpctType3. 4250
MemoryBisr 4262
MemoryBist 4310
OCC 4356
SSN 4391
RtlCells. 4540
TopModuleConnections 4551
PatternsSpecification Configuration Syntax 4555
PatternsSpecification 4556
Patterns 4563
SimulationOptions 4568
AdvancedOptions 4574
TestStep 4579
TestbenchVerify 4668
ICLNetworkVerify 4669
ProcedureStep. 4674
SSNContinuityVerify 4685
EnableGroupInfo 4687
SdfInfo 4689
LoadBoardInfo 4692
TesterInterface 4694
Loopbacks. 4697
ACLoopbacks 4699
IjtagRetargetingOptions 4701
InSystemTest 4703
DefaultsSpecification Configuration Syntax 4707
DefaultsSpecification 4708
DefaultsSpecification/DftSpecification 4709
DynamicDftSignals 4712
StaticDftSignals 4713
IjtagNetwork. 4715
BoundaryScan. 4720
MemoryBisr 4727
MemoryBist 4732
DefaultsSpecification/PatternsSpecification. 4751

SignOffOptions . . . 4753
ManufacturingOptions . . . 4755
Metadata Configuration Syntax . . . 4757
Wrapper . . . 4759
Property . . . 4768
RepeatableProperty . . . 4772
DataWrapper . . . 4774
CsvWrapper . . . 4777
Id . . . 4779
Value . . . 4782
RowIdElement . . . 4787
DftSpecification and Complex Ports . . . 4790
Embedded Boundary Scan Flow With Complex Pad IO Ports . . . 4790

Chapter 14
Tessent Core Description . . . 4793

Scan . . . 4794

Chapter 15
Tessent Shell Data Base (TSDB) . . . 4807

root_directory . . . 4807
instruments . . . 4817
dft_inserted_designs . . . 4820
patterns . . . 4827
logic_test_cores . . . 4829
<design_name>.lbist_mode_<mode_name> . . . 4830
<design_name>.atpg_mode_<mode_name> . . . 4832
<design_name>.atpg_retargeting_mode_<mode_name> . . . 4833

Chapter 16
Instrument Connectivity Language (ICL) . . . 4835

ICL Overview . . . 4837
Unsupported ICL Language Elements . . . 4837
Common Features of the ICL Language . . . 4838
Numbers . . . 4838
Strings . . . 4839
Identifiers . . . 4839
Limited Identifier Mathematics . . . 4840
Concatenations . . . 4840
ICL Statement Descriptions . . . 4842
Syntax Conventions . . . 4843
AccessLink . . . 4844
AddressPort . . . 4853
Alias . . . 4855
Attribute . . . 4858
CaptureEnPort . . . 4860
Chain . . . 4862
ClockMux . . . 4865

ClockPort 4868
DataInPort 4870
DataInOutPort 4874
DataMux 4876
DataOutPort 4879
DataRegister 4882
Enum 4887
Instance 4890
IRSelectPort 4896
LocalParameter 4897
LogicSignal 4898
Module 4900
NameSpace 4902
OneHotDataGroup 4903
OneHotScanGroup 4907
Parameter 4909
ReadEnPort 4912
ResetPort 4914
ScanInPort 4916
ScanInterface 4918
ScanMux 4924
ScanOutPort 4927
ScanRegister 4930
SelectPort 4935
ShiftEnPort 4937
TCKPort 4939
TMSPort 4941
ToCaptureEnPort 4944
ToClockPort 4948
ToIRSelectPort 4953
ToResetPort 4955
ToSelectPort 4958
ToShiftEnPort 4961
ToTCKPort 4965
ToTMSPort 4967
ToTRSTPort 4969
ToUpdateEnPort 4971
TRSTPort 4975
UpdateEnPort 4976
UseNameSpace 4978
WriteEnPort 4979

Chapter 17
Adding Custom Plugin Functionality 4981
plugin directory 4981

Chapter 18
Generation of ScanDEF 4991
ScanDEF File Generation 4991
ScanDEF File Format 4992

Appendix A
Parameter File Format and Keywords 4997
What is a Parameter File? 5000
Parameter File Syntax 5001
Parameter File Keywords. 5002
A Note about the ALL_* Parameter File Keywords 5006
ALL_EDT_INTERNAL_USE_NUMBER_SHIFTS 5006
ALL_EXCLUDE_POWER_GROUND 5006
ALL_EXCLUDE_UNUSED 5007
ALL_FIXED_CYCLES 5008
ALL_FLATTEN_TIMING 5009
ALL_FULL_TIMEPLATES 5009
ALL_FULL_VARIABLES 5009
ALL_HEADER_COMMENT 5010
ALL_IDDQ_TESTER_CYCLE 5010
ALL_IGNORE_SIMULATION_ONLY 5011
ALL_MAX_LOOP 5011
ALL_MIN_SCAN_LOAD 5012
ALL_NO_CYCLE_OPT 5012
ALL_NO_LOOP 5013
ALL_NO_PATTERN_TYPE 5013
ALL_NONSCAN_CONSTANT 5013
ALL_ORIGINAL_INDEX 5014
ALL_OVERRIDE_STROBE_METHOD 5014
ALL_PRESERVE_FORCE_X 5015
ALL_SINGLE_QUOTED_SSN_ANNOTATIONS 5015
ALL_SSN_MAXIMUM_BUS_SPEED 5016
ALL_TIME_RESOLUTION 5017
ALL_USE_CYCLE_LOOP 5017
CTL_ONE_PAT 5017
CTL_NO_PROTO 5017
CTL_STIL_0 5018
CTL_STIL_1 5018
FTDL_DESIGNER 5019
FTDL_MAX_PAT_BLOCK 5019
FTDL_REVISION 5019
FTDL_TCOMMENT 5020
FTDL_TEST_NAME 5020
FTDL_TEST_NUM 5020
FTDL_TEST_SUFFIX 5020
FTDL_ZMODE_MES 5021
SIM_ANNOTATE_QUIET 5021
SIM_CHAIN_ERROR 5022

SIM_CLOCK_MONITOR 5022
SIM_COMPARE_SUMMARY 5022
SIM_COMPARE_X 5023
SIM_DELAY_SCAN_RELEASE 5024
SIM_DIAG_FILE 5025
SIM_DISTRIBUTED_PAT 5026
SIM_DUMPFILE_PATH 5026
SIM_EARLY_RELEASE 5027
SIM_EARLY_RELEASE_LAST 5027
SIM_EARLY_RELEASE_TIME 5028
SIM_FILL_BIDISCAN 5028
SIM_FORCE_Z_AT_CYCLE 5029
SIM_INCLUDE 5029
SIM_INSTANCE_ARGUMENTS 5029
SIM_INSTANCE_NAME 5030
SIM_KEEP_PATH 5030
SIM_MASK_FILE 5031
SIM_MIN_NAME_FILE 5031
SIM_MISCOMPARE_LIMIT 5032
SIM_MODULE_INCLUDE 5032
SIM_MODULE_NAME 5033
SIM_NAME_FILE 5033
SIM_NO_CHAINNAME_FILE 5033
SIM_NO_INTERNAL_COMPARE 5034
SIM_NO_MEASURE_IN 5034
SIM_PARALLEL_DELAY_SHIFT_CLOCKS 5034
SIM_PARALLEL_EARLY_FORCE_AND_MEASURE 5035
SIM_PARALLEL_MONITOR 5035
SIM_POST_SHIFT 5036
SIM_PRE_SHIFT 5037
SIM_PRECISION 5037
SIM_SERIAL_CHAIN 5037
SIM_SHIFT_DEBUG 5038
SIM_SSN_MAXIMUM_BUS_SPEED 5039
SIM_SSN_TIMING_CHECKS 5039
SIM_STATUS_MSG 5040
SIM_STRETCH_SCAN_RELEASE 5040
SIM_TIMEPLATE_COMM 5041
SIM_TMP_REG_LENGTH 5041
SIM_TOP_NAME 5041
SIM_UNCOMPRESS_COMMAND 5042
SIM_VECTOR_COMM 5042
SIM_VECTYPE_SIGNAL 5043
STIL_2005_SCAN 5045
STIL_ADD_TIME_0_FORCE_PREVIOUS_EVENT 5045
STIL_ADD_TIME_0_MEASURE_OFF_EVENT 5046
STIL_ALLOW_SCAN_BIDI_WAVEFORM 5046
STIL_ANNOTATION_INTERVAL 5047
STIL_CHAIN_CELLS 5048

STIL_CHANNEL_CELLS 5048
STIL_CLOCK_WFC_PULSE 5049
STIL_DUAL_MODE 5049
STIL_EXCLUDE_UNUSED_BUS_BITS 5050
STIL_FULL_CHAIN 5050
STIL_IDDQ_UPPERCASE 5051
STIL_MIN_QUOTE 5051
STIL_NESTED_MACRO 5052
STIL_NO_OPTIMIZE_VEC 5052
STIL_NO_SCANSTRUCTURE 5053
STIL_NO_SCANX 5053
STIL_NO_WFC_T 5054
STIL_NOMEASURE_CLOCK 5054
STIL_PAT_ANN 5054
STIL_PAT_CMT 5055
STIL_PAT_LAB 5055
STIL_QUAD_MODE 5055
STIL_QUOTE_ALL 5056
STIL_SCAN_ANN1 5056
STIL_SCAN_IN_OUT_USE_PSEUDO_PORT_NAME 5057
STIL_STRUCTURAL 5057
STIL_TI_TITLE 5059
STIL_TIMING_BLOCK_NAME 5060
STIL_TRIM_PULSE 5060
STIL_VECTOR_ANN 5061
STIL_VERG_ESC 5062
STIL_WFC_KEEP_N 5063
STIL_WFC_KEEP_Z 5063
STIL_WRAP_ANNOTATION 5064
STIL2005_ONE_FILE 5064
TITDL_CHAIN_SETTYPE 5065
TITDL_CUSTOMER 5065
TITDL_GROUP_TIMING 5065
TITDL_KEEP_SCANOUT 5066
TITDL_LIBRARY 5066
TITDL_PARTNUM 5066
TITDL_PSEUDO_PREFIX 5067
TITDL_REVISION 5067
TITDL_SET_DESCRIPTION 5068
TITDL_SETNAME 5068
TITDL_SETTYPE 5068
WGL_ADD_LASTX 5069
WGL_ALT_BIDI 5069
WGL_ALT_VECT_ANN 5070
WGL_ANNOTATION_INTERVAL 5070
WGL_EDGE_STROBE 5071
WGL_EXCLUDE_UNUSED_BUS_BITS 5071
WGL_FULL_CHAIN 5072
WGL_FULL_SCANGROUP 5073

WGL_GROUP_PIN 5074
WGL_INV_SC 5074
WGL_LOOP_ONLY 5075
WGL_NO_SCANX 5075
WGL_NOMEASURE_CLOCK 5075
WGL_ONE_ILLINOIS 5076
WGL_PATTERN_NAME 5076
WGL_TRIM_LEADING_P 5077
WGL_TRIM_PULSE 5077
WGL_VECTOR_ANN 5078
WGL_VERG_ESC 5078
WGL_VTRAN_PADSC 5079
WGL_WRAP_ANNOTATION 5079

Appendix B
HDL Limitations in the Tessent Shell Flow 5081

HDL Limitations in the Tessent Shell Flow for Verilog and SystemVerilog 5083
HDL Limitations in the Tessent Shell Flow for VHDL 5088
SystemVerilog Interfaces 5091
Multi-dimensional Instantiation of SystemVerilog Interfaces 5091
SystemVerilog Interfaces Cannot Be Edited 5091
Outside Connections on Interface Type Ports Cannot be Intercepted 5091
Insertion of an Instance of a Module With a SystemVerilog Interface Port Type 5092
SystemVerilog Generic Interface Port Declarations in the Root Module of a Child Block 5092
SystemVerilog Interfaces in Task or Function Invocation 5093
SystemVerilog Interfaces With Logic 5094
Modport Expressions 5094
Nested Modules 5094
Configurations for Verilog 5095
SystemVerilog Enumerations 5096
SystemVerilog Enum Methods 5096
SystemVerilog Enum Type in Port Declarations 5096
Complex Port Types for Verilog and SystemVerilog 5097
Editing of Ports and Nets of Complex Types 5097
Unpacked Dimensions and Other Complex Types in Port Declarations of the Root Module of a Design 5098
Complex Types in Top-Level Ports at the Chip Level 5098
Complex Object Permissions 5098
Functions, Tasks, and Procedural Statements for Verilog and SystemVerilog 5101
SystemVerilog Extern Modules 5101
Unrolled Generate Loops for Verilog and SystemVerilog 5102
Unrolling of Generate Loops Side Effects 5102
Insertion of Custom Logic Connecting to a Port or Net in an Unrolled Generate Loop . . 5105
Parameterized Designs for Verilog and SystemVerilog 5107
Parameter Overrides in the Instantiation of the Root Module of a Child Block 5107
Support for the Defparam Construct 5107
Implicit Port Syntax 5108

SystemVerilog Implicit Port Syntax in the Instantiation of the Root Module of a Child Block 5108
SystemVerilog 2009 5109
New Module Insertion in VHDL 5110
Insertion of VHDL Entities with Unconstrained Ports 5110
Insertion of VHDL Entities with Uninitialized Generics 5110
Complex Port Types for VHDL. 5111
Editing of Ports and Nets of Complex Types 5111
Array of Array and Record Types in Port Declarations in the Root Module of a Design . 5111
Complex Types in Top-Level Ports at the Chip Level 5111
Ports with Unconstrained Array Types in the Root Design Entity of a Child Block 5112
Procedures, Functions, and Process Statements for VHDL. 5112
Configurations for VHDL 5112
VHDL 2008 5112
Parameterized Designs for VHDL 5114
Generic Overrides in the Instantiation of the Root Design Entity of a Child Block 5114

Appendix C
Getting Help 5115

The Tessent Documentation System 5115
Global Customer Support and Success 5116

Index

Third-Party Information

List of Figures

Figure 3-1. Illustration of Various Clock Types 137
Figure 3-2. Merge Patterns Sets Into One Patterns Set 160
Figure 3-3. Merging Patterns for Top Level and Core at the Top Level 162
Figure 3-4. Example Usage of add_dft_clock_mux 177
Figure 3-5. Example Usage of add_dft_control_points Command 184
Figure 3-6. Multiplexing and Pipelining Logic for Output Connections 187
Figure 3-7. Enable Logic and Pipelining for Input Connections 188
Figure 3-8. Multiplexing for Destinations of Modal Connections 188
Figure 3-9. Timing Arcs for Scan In and Out Paths 190
Figure 3-10. Waveforms of Selected Dynamic DFT Signals 224
Figure 3-11. Holding Scan Cell Without Disable 236
Figure 3-12. Holding Scan Cell With Disable 236
Figure 3-13. Block-Level Port Association 237
Figure 3-14. Chip-Level Port Association 238
Figure 3-15. Example of a False Path to be Defined 256
Figure 3-16. Example Circuit for Defining False Paths 257
Figure 3-17. Static Fault 270
Figure 3-18. Transition Fault - Incompatible Clocks 271
Figure 3-19. Transition Fault - Compatible Clocks 272
Figure 3-20. Transition Fault - No Equivalent Pins 272
Figure 3-21. Transition Fault - Equivalent Pins 273
Figure 3-22. Example Scan Interfaces 281
Figure 3-23. Logical Connection Attributes 296
Figure 3-24. Logical Connection Example 297
Figure 3-25. MISR Placement 308
Figure 3-26. Logical Connection Example 335
Figure 3-27. Schematic Callouts 378
Figure 3-28. Schematic Callouts (Collapsed) 378
Figure 3-29. analyze_simulation_mismatches Example 468
Figure 3-30. Pin and Port Definition on an Instance 544
Figure 3-31. Design With IJTAG Host Tiling 564
Figure 3-32. Flow to Generate an Exclude List 577
Figure 3-33. create_layout With Extra Layout Hierarchy 591
Figure 3-34. create_layout With Extra Design Hierarchy 591
Figure 3-35. SSN Continuity Pattern - Internal SSH Paths Coverage 639
Figure 3-36. SSN With Multiple Datapaths 640
Figure 3-37. Example Design With Two SSN Datapath Configurations 641
Figure 4-1. Hierarchical Test Logic Insertion Example 765
Figure 5-1. Design Inheritance and Module Matching 920
Figure 5-2. Various Components of a Configuration Element 943

Figure 5-3. Circuit Illustrating the Usage of the get_fanins Command. 1012
Figure 5-4. Circuit Illustrating the Usage of the get_fanouts Command. 1016
Figure 5-5. Design Inheritance and Module Matching 1123
Figure 5-6. iApply Time Frame. 1296
Figure 5-7. Inverter Interception 1345
Figure 5-8. Intercepting an Input Pin From Inside An Instance 1416
Figure 5-9. Intercepting an Output Port From Inside the Current Design. 1417
Figure 5-10. Intercepting an Input Pin From Outside An Instance 1418
Figure 5-11. Intercepting an Output Pin From Outside An Instance. 1419
Figure 5-12. Timeplate for TCK Off State = 0 and TCK Ratio = 1 1429
Figure 5-13. Scan Frame Timing for TCK Off State = 0 and TCK Ratio = 1. 1429
Figure 5-14. Timeplate for TCK Off State = 0 and TCK Ratio = 2 1430
Figure 5-15. Scan Frame Timing for TCK Off State = 0 and TCK Ratio = 2. 1430
Figure 5-16. Timeplate for TCK Off State = 0 and TCK Ratio = 4 1431
Figure 5-17. Scan Frame Timing for TCK Off State = 0 and TCK Ratio = 4. 1431
Figure 5-18. Timeplate for TCK Off State = 1 and TCK Ratio = 1 1432
Figure 5-19. Scan Frame Timing for TCK Off State = 1 and TCK Ratio = 1. 1432
Figure 5-20. Timeplate for TCK Off State = 1 and TCK Ratio = 2 1433
Figure 5-21. Scan Frame Timing for TCK Off State = 1 and TCK Ratio = 2. 1433
Figure 5-22. Timeplate for TCK Off State = 1 and TCK Ratio = 4 1434
Figure 5-23. Scan Frame Timing for TCK Off State = 1 and TCK Ratio = 4. 1434
Figure 6-1. Design Views in a Bottom-up Flow 1494
Figure 6-2. TSDB logic_test_cores Directory Structure 1552
Figure 6-3. Case Analysis Example. 1560
Figure 6-4. Plugin Directory Contains XYZ Custom DRC Rules. 1601
Figure 6-5. Violation Detected by XYZ_S1 DRC. 1602
Figure 6-6. Combinational Paths Permitted and Rejected by XYZ_S2 1616
Figure 6-7. Violation Detected by XYZ_S3 1619
Figure 6-8. Forward Walk to Find All Destination Hits 1622
Figure 6-9. Backward Walk to Final All Source Hits 1622
Figure 6-10. Violations Detected by XYZ_S4_DRC 1624
Figure 6-11. Design Inheritance and Module Matching 1702
Figure 7-1. Design Inheritance and Module Matching 1952
Figure 7-2. report_memory_dft_summary Tcl Dictionary Structure 1972
Figure 7-3. Controller Tcl Dictionary Structure 1973
Figure 7-4. Inversion Flags With Library Cell Pins 2075
Figure 7-5. Inversion Flags Without Library Pins. 2075
Figure 7-6. report_scan_path Example 2099
Figure 7-7. set_clock_off_simulation Off and set_split_capture_cycle Off 2176
Figure 7-8. set_clock_off_simulation On and set_split_capture_cycle Off 2176
Figure 7-9. set_clock_off_simulation Off and set_split_capture_cycle On 2176
Figure 7-10. set_clock_off_simulation On and Set Split Capture On. 2177
Figure 7-11. Typical Contents of the synthesis_outdir Directory 2237
Figure 7-12. Example Output Directory Structure for Questa. 2246
Figure 7-13. Example Output Directory Structure for VCS 2249

Figure 7-14. Example Output Directory Structure for Incisive 2250
Figure 8-1. Global Versus Local Bus Simulation Analysis Example 2326
Figure 8-2. Half-Cycle Skew 2378
Figure 8-3. One-Cycle Skew 2379
Figure 8-4. Multiple-Cycle Skew 2379
Figure 8-5. Same Clocks Between Loads On 2382
Figure 8-6. Compatible Clocks Between Loads On 2382
Figure 8-7. domain_clock Alone 2384
Figure 8-8. False Path Intersection and Effect Cones 2588
Figure 8-9. Load/Unload and Capture Timing With SSN 2685
Figure 8-10. Timing When Independent Insertion Flow is Not Used 2691
Figure 8-11. Timing for Independent Insertion Flow 2691
Figure 8-12. Clock Selection Scheme 2847
Figure 9-1. Example of a Reconvergent Controlling Path. 3003
Figure 9-2. Flat Schematic Display After add_schematic_path Command. 3006
Figure 9-3. Result of trace_flat_model While Tagging Instances 3007
Figure 9-4. Writing Out the Modified Design in rtl vs no_rtl Modes 3045
Figure 9-5. Effect of Successive Design Editing Passes on the Design View. 3046
Figure 9-6. Example CompressBisrChain Configuration File. 3105
Figure 10-1. Example of Design Level 3171
Figure 10-2. Example of Primitive Level 3172
Figure 10-3. Data Reported for a Specific Gate 3174
Figure 10-4. Clock Cycle Terminology 3223
Figure 10-5. Example Clock Cone 3224
Figure 10-6. Example Effect Cone 3224
Figure 10-7. Pin in Both Cones 3225
Figure 10-8. Tessent Visualizer Flat Schematic Window View Showing Clock Cones 3225
Figure 10-9. Clock Source And Derived Clocks 3226
Figure 10-10. C1 Violation Example. 3229
Figure 10-11. C2 Rule Example Circuit 3233
Figure 10-12. C3 Violation Example. 3234
Figure 10-13. Example Timing That Allows Sink to Capture Source's New Data. 3235
Figure 10-14. C4 Violation Example. 3240
Figure 10-15. Example Where Actual Behavior Differs from Tool's Prediction 3240
Figure 10-16. C5 Rule Example Circuit 3245
Figure 10-17. C6 Rule Example Circuit 3247
Figure 10-18. C7 Rule Example Circuit 3249
Figure 10-19. C8 Rule Example Analysis 3250
Figure 10-20. C9 Rule DRC Analysis 3252
Figure 10-21. C10 Rule Example Circuit 3253
Figure 10-22. C22 DRC Example Design 1 3259
Figure 10-23. C22 DRC Example Design 2 3260
Figure 10-24. C27 Design Rule Violation 3265
Figure 10-25. No Violation of the C27 Design Rule 3266
Figure 10-26. Example DFT_C1 Violation in the Presence of a Blackbox 3285

Figure 10-27. Example DFT_C1 Violation on Differential Clocks 3287
Figure 10-28. Schematic and Flat Model View of a Differential Clock Path 3287
Figure 10-29. Example of DFT_C4 Violation. 3290
Figure 10-30. Example DFT_C6 Violation 3293
Figure 10-31. DFT_C9 Example of Type 1 Violation. 3297
Figure 10-32. Race Condition Check 3298
Figure 10-33. Tessent Visualizer Showing a DFT_S1 Error Location 3311
Figure 10-34. Example E4 Violation Trace in Flat Schematic Window of Tessent Visualizer 3316
Figure 10-35. Example E10 Violation Trace in Flat Schematic Window of Tessent Visualizer 3324
Figure 10-36. Blocked Decompressor Input 3338
Figure 10-37. Blocked Channel Input to Decompressor 3339
Figure 10-38. Output Channel and Compactor Connection Blocked 3341
Figure 10-39. Viewing F8 DRC Violation in Tessent Visualizer 3343
Figure 10-40. Tracing Back on the F8 X path 3344
Figure 10-41. Tri-State Driven to X by connection to VDD 3345
Figure 10-42. F8 DRC Violation Cause 3346
Figure 10-43. Blocked Scan Chain Path 3347
Figure 10-44. analyze_drc_violation F22 in Flat Schematic Window 3359
Figure 10-45. Scan Data Path to an External Subordinate Instrument 3414
Figure 10-46. Host ScanInterface Virtual Scan Path. 3414
Figure 10-47. Host ScanInterface Circular Scan Path 3415
Figure 10-48. Invalid Multi-TAP Configuration 3437
Figure 10-49. Valid Multi-TAP Configuration 3437
Figure 10-50. Example 3441
Figure 10-51. Failing Identification. 3450
Figure 10-52. Common Cell Structure 3450
Figure 10-53. Failing Identification. 3451
Figure 10-54. Example S3 Rule Violation. 3543
Figure 10-55. T3 Violation 3590
Figure 10-56. Tracing T3 Violation 3591
Figure 10-57. T4 Violation 3594
Figure 10-58. T4 Violation Showing States 3595
Figure 10-59. T4 Violation Traced Back Showing States 3595
Figure 10-60. T5 DRC Violation. 3598
Figure 10-61. T5 DRC Violation Traced Back 3598
Figure 10-62. Example of T25 Violation. 3607
Figure 10-63. W38 Reference Circuit 3640
Figure 10-64. RAM Summary Results and Test Capability Messages 3648
Figure 11-1. Tessent Shell Hierarchical Design Objects 3652
Figure 11-2. Tessent Shell Non-Hierarchical Design Objects. 3653
Figure 13-1. Module Hosting TDRs Inside EDT 3878
Figure 13-2. IJTAG Network With TDRs Hosted in a Hybrid EDT/LBIST Controller 3879
Figure 13-3. Hold-Mux Logic for edt_clock 3908

Figure 13-4. IJTAG Network Example Schematic 3929
Figure 13-5. Existing Compliance Enable Modules 3936
Figure 13-6. IJTAG Network Inserted Into a Tiled Design. 3937
Figure 13-7. DataInPorts Example Schematic. 3946
Figure 13-8. TDI Local Mux Shown in Die Closest to TDO in a 2.5D Package 3953
Figure 13-9. Sacrificial Pad Multiplexing Module 3954
Figure 13-10. HostScanInterface/Interface Example 1 Schematic 3958
Figure 13-11. HostScanInterface/Interface Example 3 Schematic 3961
Figure 13-12. DesignInstance Example Schematic 3965
Figure 13-13. ScanMux Internal Schematic When Used on a DR Scan Path 3967
Figure 13-14. ScanMux Internal Schematic When Used on a TAP Scan Path 3968
Figure 13-15. ScanMux Example Schematic 3971
Figure 13-16. Example Use of SecondaryHostScanInterface for Tiling 3978
Figure 13-17. SIB Internal Schematic 3981
Figure 13-18. Sib(sti) With Scan Isolation Chain 3983
Figure 13-19. Added Scan Isolation Logic for SIB(sti). 3984
Figure 13-20. Mini-OCC for Scan Tested TCK Domain With Capture Triggered on scan_en 3985
Figure 13-21. Mini-OCC for Scan Tested TCK Domain With Capture Triggered on capture_en 3985
Figure 13-22. SIB Example Schematic 3989
Figure 13-23. Update Stage Timing Diagram 3996
Figure 13-24. Detailed View of TDR Schematic 3999
Figure 13-25. DataOutPort Connection Multiplexing 4008
Figure 13-26. TAP FSM Diagram 4014
Figure 13-27. TAP Block Diagram 4015
Figure 13-28. Example Multi-TAP Configuration Schematic. 4017
Figure 13-29. Example of TDR Scan Selector 4018
Figure 13-30. Addition to TAP to Make it a PTAP. 4042
Figure 13-31. STAP Block Diagram 4043
Figure 13-32. TDO Loop Timing From Upper Die to Lower Die 4044
Figure 13-33. PTAP/STAP Network Inserted Within the Top Physical Blocks of a Multi-die Stack. 4048
Figure 13-34. PTAP/STAP Network Inserted into a Tiled Die. 4050
Figure 13-35. TPSP Connections Schematic. 4056
Figure 13-36. TPSP Timing Diagram 4056
Figure 13-37. TPSP Hardware Schematic 4057
Figure 13-38. Boundary Scan Chain Distributed Across a Tiled Design 4119
Figure 13-39. Example Design With Pads. 4129
Figure 13-40. Single Logical Group Example. 4130
Figure 13-41. Two Logical Groups Example 4130
Figure 13-42. Two Logical Groups Example1 with One Output per Enable Cells. 4131
Figure 13-43. Two Logical Groups Example2 with One Output per Enable Cells. 4131
Figure 13-44. Typical Boundary Scan Cell Inserted for an Inout Pad. 4136
Figure 13-45. Boundary Scan Cell as Observation and Control in Logic Test Modes 4137

Figure 13-46. Adding Auxiliary Input and Output Logic in Bscan Cell 4138
Figure 13-47. Path Used When AuxOut Logic Activated 4139
Figure 13-48. Forcing aux_out_en Low in Capture Mode to Observe Core Output 4139
Figure 13-49. Using AuxIn and Controlling to Core Signal in ATPG 4140
Figure 13-50. Bidirectional Port With AuxOut 4140
Figure 13-51. Extra Logic Added When a Port is Used Both Internally and Externally . . . 4141
Figure 13-52. Needed Resources From Third-Party TAP Controller 4155
Figure 13-53. Schematic of the Boundary Scan Interface Module 4161
Figure 13-54. Segmenting the Boundary Scan Chains into Logic Test Chains 4162
Figure 13-55. Example pin_order_file. 4165
Figure 13-56. Example Design With Pads. 4185
Figure 13-57. Single Logical Group Example. 4186
Figure 13-58. Two Logical Groups Example 4186
Figure 13-59. Two Logical Groups Example1 with One Output per Enable Cells. 4187
Figure 13-60. Two Logical Groups Example2 with One Output per Enable Cells. 4187
Figure 13-61. Typical Boundary Scan Cell Inserted for an Inout Pad. 4192
Figure 13-62. Boundary Scan Cell as Observation and Control in Logic Test Modes 4193
Figure 13-63. Adding Auxiliary Input and Output Logic in Bscan Cell 4194
Figure 13-64. Path Used When AuxOut Logic Activated 4195
Figure 13-65. Forcing aux_out_en Low in Capture Mode to Observe Core Output. 4195
Figure 13-66. Using AuxIn and Controlling to Core Signal in ATPG 4196
Figure 13-67. Bidirectional Port With AuxOut 4196
Figure 13-68. Extra Logic Added When a Port is Used Both Internally and Externally . . . 4197
Figure 13-69. package0 Bonding Configuration Example 4213
Figure 13-70. package1 Bonding Configuration Example 4214
Figure 13-71. package2 Bonding Configuration Example 4214
Figure 13-72. BISR Segment Order File 4264
Figure 13-73. Tiling BISR Ports With a BISR Controller. 4281
Figure 13-74. Tiling BISR Ports Without a BISR Controller 4283
Figure 13-75. System-Level BISR Tiling 4283
Figure 13-76. Memory BISR Chains in a Tiled Design 4284
Figure 13-77. Hierarchical Chip Using the Tiling Design Flow 4365
Figure 13-78. OCC Usage During the Internal Test Modes of All Tiles. 4366
Figure 13-79. OCC Usage During the External Test Modes of the Tiles 4367
Figure 13-80. Case 1 — clock_intercept_node Has Functional Source and fast_clock Not Specified 4386
Figure 13-81. No Functional Source and fast_clock Specified 4387
Figure 13-82. Case 3 - Functional Source and fast_clock Specified. 4388
Figure 13-83. SSN Example One 4398
Figure 13-84. SSN Example Two With Connecting Tiles 4399
Figure 13-85. SSN Network in a Tiled Design 4401
Figure 13-86. Schematic Showing Data Handoff Between Two CTS Regions 4422
Figure 13-87. Timing Diagram Showing Data Handoff Between Two CTS Regions 4423
Figure 13-88. Schematic Showing Loop Timing Path for Bus Data Out 4423
Figure 13-89. Waveforms Showing Loop Timing Path for Bus Data Out 4424

Figure 13-90. Output Data Slow Down Using BusFrequencyDivider Node 4425
Figure 13-91. SSN Clock Domain Crossing Inside Physical Region 4429
Figure 13-92. SSN Clock Domain Crossing Across Physical Regions. 4431
Figure 13-93. Clock Domain Crossing at the Input of the Receiving Block. 4432
Figure 13-94. BusFrequencyDivider With Input Retimed 4433
Figure 13-95. Relative Timing Path for the Input Datapath 4437
Figure 13-96. Waveform Showing Relative Timing Paths for Input Datapath (fast) 4437
Figure 13-97. Waveform Showing Relative Timing Paths for Input Datapath (slow) 4437
Figure 13-98. Packets per Phase Counter Value With delayed_only 4451
Figure 13-99. Packets per Phase Counter Value With early_or_delayed 4452
Figure 13-100. FIFO With frequency_ratio of Four 4455
Figure 13-101. Waveform for FIFO early_or_delayed Configuration 4456
Figure 13-102. Waveform for FIFO delayed_only Configuration 4457
Figure 13-103. Multiplexer Usage Example 4470
Figure 13-104. Data and Clock multiplexing example 4471
Figure 13-105. ScanHost Block Diagram 4488
Figure 13-106. ScanHost Bus Register Access Via IJTAG. 4489
Figure 13-107. Two SSH Nodes Used in Standard Mode. 4491
Figure 13-108. SSN Packet Formats When Not Using On-Chip Compare. 4491
Figure 13-109. Six SSH Nodes Used in On-Chip Compare Mode 4492
Figure 13-110. SSH On-Chip Compare Logic 4492
Figure 13-111. SSN Packet Formats When Using On-Chip Compare 4493
Figure 13-112. SSH Circuitry Creating Scan Control and Clock Signals. 4500
Figure 13-113. SSH Scan Datapaths Interacting With Scan Chains 4501
Figure 13-114. Using a Clock Shaper Cell To Generate Scan Clocks 4503
Figure 13-115. Using a Clock Shaper Cell To Generate Scan Clocks With Bypass Mode . 4504
Figure 13-116. ScanHost Node Sourcing test_clock With Following DFT Signal Clock Gater 4505
Figure 13-117. ScanHost Sourcing test_clock Reused for Legacy Bypass Mode. 4506
Figure 13-118. SSN Bus Clock Tree When Streaming Through the Bus 4507
Figure 13-119. ScanHost Bus Clock Tree When Streaming Through IJTAG 4508
Figure 13-120. TCK as an External Capture Clock. 4509
Figure 13-121. TCK as an External Capture Clock When Streaming Through IJTAG 4510
Figure 13-122. EDT Controllers Without SSN 4525
Figure 13-123. EDT Controllers With ScanHost and Bypass Mode. 4527
Figure 13-124. Extra Logic Inserted to Share I/Os Between SSH and Legacy Mode. 4528
Figure 13-125. ScanHost Sourcing test_clock With Distributed DFT Signal Clock Gaters. 4532
Figure 13-126. Clock Shaper Cell Application Schematic 4544
Figure 13-127. Clock Shaper Cell Application Timing. 4544
Figure 13-128. Timing Diagram of the Global tristate_enable_non_contacted Test 4595
Figure 13-129. Timing Diagram of the Local tristate_enable_non_contacted Test 4596
Figure 13-130. Fast BISR Muxing 4622
Figure 13-131. Example Chip With Two Cores and Complex Clocking 4684
Figure 15-1. Root Directory Structure. 4808
Figure 15-2. Example Chip Design Process in Different Sites 4810

Figure 15-3. First Insertion Pass: b1 Loading 4811
Figure 15-4. Modified b1 rtl View After First Insertion Pass 4812
Figure 15-5. Design Source Dictionary After First Insertion Pass 4813
Figure 15-6. Second Insertion Pass: b1 Loading 4814
Figure 15-7. Modified b1 rtl View After Second Insertion Pass 4815
Figure 15-8. Design Source Dictionary After Second Insertion Pass 4816
Figure 15-9. File Paths in Design Source Dictionary After Second Insertion Pass. 4817
Figure 18-1. Single Scan Mode Environment 4992
Figure 18-2. Multi Scan Mode Environment. 4992

List of Tables

Table 2-1. Conventions for Command Line Syntax 74
Table 3-1. Conventions for Command Line Syntax 97
Table 3-2. Commands A Through C 98
Table 3-3. Example Cell Value Changes With Different Constraints for Scan Patterns . . . 126
Table 3-4. Constraint Type Recommendations 127
Table 3-5. Example Scan Cell Value Changes With Different Constraints for Chain Patterns 127
Table 3-6. Mapping of Core Descriptions and Instances 162
Table 3-7. Mapping of Core Descriptions With Modules and Instances 163
Table 3-8. DFT Logic Inserted for Different add_dft_clock_enables Locations 169
Table 3-9. DFT Logic for add_dft_control_points Nodes 179
Table 3-10. Pre-Registered Static Global DFT Control Signal Names 201
Table 3-11. Pre-Registered Static Logic Test Control Signal Names 206
Table 3-12. Pre-Registered Static Scan Mode Signal Names 210
Table 3-13. Pre-Registered Dynamic DFT Signal Names 219
Table 3-14. ICL Port Types 277
Table 3-15. Logical Connection Attributes 295
Table 3-16. Attribute Value Combinations 371
Table 3-17. Suspect Signature Diagnosis Results 454
Table 3-18. Example 1 -threads Option 457
Table 3-19. Example 2 -threads Option 458
Table 3-20. Attribute Value Combinations 630
Table 4-1. Commands D Through F 646
Table 4-2. Fault Class Codes and Class Names 717
Table 5-1. Commands G-Q 884
Table 5-2. Types of Configuration Elements 945
Table 5-3. Available Partitions for Introspection 949
Table 5-4. ICL Objects and Applicable Options 1050
Table 5-5. Source Types 1078
Table 5-6. Events of Type “vector” 1169
Table 5-7. Events of Type “scan” 1171
Table 5-8. Events of Type “reset” 1174
Table 5-9. Events of Type “loop” 1174
Table 5-10. Events of Type “goto_state” 1175
Table 5-11. Events of Type “nop” 1175
Table 5-12. Calling get_pins -of_icl_ports in RTL Mode 1190
Table 5-13. Calling get_pins -of_icl_ports in No RTL Mode After Synthesis 1190
Table 5-14. Calling get_pins -use_path_matching_options in RTL Mode 1190
Table 5-15. Calling get_pins -use_path_matching_options in No RTL Mode After Synthesis 1190

Table 5-16. Calling get_pins -use_path_matching_options in No RTL Mode After Synthesis (Flattened Complex Ports) 1191
Table 5-17. Calling get_pins -use_path_matching_options in No RTL Mode After Synthesis With Bit-Blasting 1191
Table 5-18. Calling get_pins -use_path_matching_options in No RTL Mode After Synthesis with Bit-Blasting (Special Characters Replaced by Underscores) 1191
Table 5-19. Calling get_ports -of_icl_ports in RTL Mode 1201
Table 5-20. Calling get_ports -of_icl_ports in No RTL Mode After Synthesis 1201
Table 5-21. Calling get_ports -use_path_matching_options in RTL Mode 1201
Table 5-22. Calling get_ports -use_path_matching_options in No RTL Mode After Synthesis 1202
Table 5-23. Calling get_ports -use_path_matching_options in No RTL Mode After Synthesis with Bit-Blasting 1202
Table 5-24. Calling get_ports -use_path_matching_options in No RTL Mode After Synthesis with Bit-Blasting (Special Characters Replaced by Underscores) 1202
Table 5-25. Interception Based on Node Type 1341
Table 5-26. Permitted Cell Function Names 1343
Table 5-27. RunLoop Scaling Algorithm 1367
Table 5-28. Calculating tck_period, tck_ratio and tester_period 1428
Table 6-1. Commands Ra Through Report_i 1470
Table 6-2. Mask File Statements 1547
Table 6-3. Named Capture Procedure and Clock Control Handling Under Different Conditions 1556
Table 6-4. Mode values for the RTL Synthesis Proc 1599
Table 6-5. “-delay_distribution static” With -slack and -difference 1668
Table 6-6. “-delay_distribution actual” With -slack and -difference 1669
Table 6-7. Clock Compatibility Summary 1713
Table 6-8. Available Partition for Introspection 1728
Table 6-9. Available Information Displayed and Arguments 1777
Table 6-10. Fault Class Codes and Names 1850
Table 6-11. Name Conventions Used by report_faults -Cell_name 1854
Table 6-12. Reportable Gate Types 1882
Table 6-13. Clock Port Categories 1883
Table 7-1. Commands Report_j Through R 1926
Table 7-2. Notation Used in report_patterns Display 2027
Table 7-3. Default report_processors Information Display 2056
Table 7-4. Additional Data Displayed by -Verbose 2057
Table 7-5. Sequential Element Types 2070
Table 7-6. Library Cell Pins and Inversion Flag Position Values 2074
Table 7-7. Format Codes 2144
Table 7-8. Reason for Failed Identification 2200
Table 7-9. Untestable Faults that are Reclassified by reset_au_faults 2213
Table 8-1. Commands S 2272
Table 8-2. User-Specifiable Attributes 2300
Table 8-3. Default Clock Compatibility Summary at a Clock Domain Boundary 2377

Table 8-4. Contexts Available by License 2399
Table 8-5. Core Instance Parameters and Values by Instrument 2403
Table 8-6. Predefined Simulation Contexts For All Contexts Except dft -scan and dft -test_points 2434
Table 8-7. Predefined Simulation Contexts For dft -scan and dft -test_points Contexts Only 2435
Table 8-8. Default Extensions for set_design_sources 2457
Table 8-9. Pattern Sources 2577
Table 8-10. Extra Timing Margin With -inject_cycles 2635
Table 8-11. Valid TAP States 2636
Table 8-12. Annotation Parameter Values 2638
Table 8-13. Pin Functions That Can Be Renamed 2669
Table 8-14. WIRE Bus Contention Truth Table 2752
Table 8-15. AND Bus Contention Truth Table 2752
Table 8-16. OR Bus Contention Truth Table 2752
Table 8-17. Keywords for Pattern Types 2767
Table 8-18. Mask File Statements 2782
Table 8-19. Fault Classes for Relevant Coverage 2825
Table 8-20. Fault Sub-Classes for Relevant Coverage 2826
Table 8-21. Attribute Value Combinations 2839
Table 8-22. SDC File and -ADD_Hold_time_when_no_settings_in_sdc switch 2926
Table 8-23. X Handling for Level-Sensitive Ports 2968
Table 8-24. X Handling for Edge-Triggered Ports 2969
Table 9-1. Commands T Through Z 2997
Table 9-2. Difference Between RTL and Netlist Files 3045
Table 9-3. Single Module Per File Naming Convention 3061
Table 9-4. Name Conventions Used by write_faults -Cell_name 3088
Table 10-1. Clocking That Can Result in a C3 Signal Race 3234
Table 10-2. Clocking that Can Result in a C4 Signal Race 3239
Table 10-3. Error Messages Generated by the Clock Tracing Checks 3285
Table 10-4. Error Messages Generated by Differential Clock Checks 3286
Table 10-5. Error Messages Generated by the Clock Tracing Checks 3293
Table 10-6. Error messages Generated by DFT_C7 Checks 3294
Table 10-7. Error messages Generated by DFT_C8 Checks 3295
Table 10-8. Error Messages Generated by DFT_C9 Checks 3296
Table 10-9. Error and Warning Messages Generated by DFT_C10 3300
Table 10-10. Warning Messages Generated by DFT_C11 Checks 3307
Table 10-11. ICL Extraction DRCs 3373
Table 10-12. Port Connections 3377
Table 10-13. Port Types and Required Input Binding 3401
Table 11-1. Example 1: port_bundle “data_in[1][4].b” 3743
Table 11-2. Example 2: port “data_in[1][4].b[0]” 3743
Table 11-3. Example 3: port_bundle “data_in[1][4]” 3743
Table 11-4. Example 4: port_bundle “data_in[1]” 3744
Table 11-5. Example 5: port_bundle “data_in” 3744

Table 11-6. Built-in Primitives and Primitive Port Names 3758
Table 11-7. Built-In Attributes on ICL Instance Object Type 3767
Table 11-8. Built-In Attributes on ICL Module Object Type 3771
Table 11-9. Built-In Attributes on ICL Pin Object Type 3781
Table 11-10. ICL Names and Associated Attributes 3784
Table 11-11. Built-In Attributes on ICL Port Object Type 3785
Table 11-12. ICL Module corea (RTL View: 4 Ports; ICL Module: 6 Equivalent Ports) . . 3811
Table 11-13. ICL Module coreb (RTL View: 3 Ports; ICL Module: 5 Equivalent Ports) . . 3811
Table 11-14. ICL Module corea (RTL View: 4 Ports; ICL Module: 6 Equivalent Ports . . . 3811
Table 11-15. ICL Module coreb (RTL View: 3 Ports; ICL Module: 5 Equivalent Ports) . . 3812
Table 11-16. Built-in Attributes on Scan Element (SE) Object Type 3814
Table 11-17. Built in Attributes on Scan Chain Family Object Type 3820
Table 11-18. Built-in Attributes on Scan Mode Object Type 3821
Table 11-19. Built-In Attributes on Test Point Object Type 3821
Table 12-1. Commands for Manipulating Attributes 3829
Table 12-2. Attribute Value Types 3831
Table 12-3. POSIX Extended Regular Expression Metacharacters 3834
Table 12-4. Special Characters in Regular Expressions 3836
Table 12-5. Special Characters Set in Regular Expression 3836
Table 13-1. Configuration Data Editing and Introspection Commands 3843
Table 13-2. SyntaxConventions for Configuration Files 3844
Table 13-3. Valid cell_type Values and Usages 3855
Table 13-4. Illustration of Different Boundary Scan Implementations 4102
Table 13-5. BoundaryScanCellOptions list 4105
Table 13-6. Illustration of the “sample_only” Boundary Scan Cell Option 4110
Table 13-7. Connections To and From Internal Boundary Scan Cells 4149
Table 13-8. Illustration of Different Boundary Scan Implementations 4172
Table 13-9. BoundaryScanCellOptions list 4175
Table 13-10. Illustration of the “sample_only” Boundary Scan Cell Option 4180
Table 13-11. Connections To and From Internal Boundary Scan Cells 4205
Table 13-12. repair_mode_connection Repair Modes 4290
Table 13-13. Kill Clock Mode Truth Table 4359
Table 13-14. IJTAG Registers in ScanHost Nodes 4494
Table 13-15. ROM MISR Signature Source 4660
Table 13-16. DramOptions Parameter Values 4662
Table 15-1. Instrument Container Names 4818
Table 15-2. Format to Extension Mapping for the Interface View 4825
Table 16-1. SyntaxConventions for ICL Files 4843
Table A-1. Vector Types 5042
Table A-2. Translation Table 5044
Table A-3. STIL Special Keywords 5061
Table A-4. STIL Pattern Types 5062
Table A-5. STIL Vector Types 5062
Table B-1. Verilog and SystemVerilog Limitations 5083
Table B-2. VHDL Limitations 5088

Chapter 1
Introduction

Tessent Shell is an environment that provides access to all Tessent products.

Overview **59**

Product Introduction **62**

Tessent BoundaryScan 62
Tessent Diagnosis 63
Tessent FastScan and Tessent TestKompress. 64
Tessent IJTAG 66
Tessent LogicBIST 67
Tessent MemoryBIST. 67
Tessent MissionMode 68
Tessent Scan 69
Tessent ScanPro 70
Tessent SiliconInsight. 70
Tessent DesignEditor 71

Overview

Application commands are provided for Tessent Shell, as well as for the following products:

- Tessent BoundaryScan is a complete solution for the creation and integration of boundary scan cells and related control logic for embedded test and diagnosis of integrated circuit I/Os, as well as test and diagnosis of board-level interconnects between ICs. Tessent BoundaryScan provides a completely automated solution for adding standard boundary scan support to ICs of any size or complexity, reducing IC engineering development effort and improving time-to-market.
- Tessent Diagnosis is a sophisticated diagnosis tool used at different stages of test to help characterize and locate defects in manufactured chips. It is for users who need to shorten the time it takes to perform yield ramp, yield learning and customer return analysis. The unique advanced diagnosis capabilities it provides are also available in Tessent TestKompress.
- Tessent FastScan is a comprehensive combinational Automatic Test Pattern Generation (ATPG) system optimized for full-scan designs. It offers high speed and accurately measured high test coverage to guarantee your product quality and reliability.
- Tessent IJTAG is the Siemens EDA implementation of the IEEE 1687-2014 (IJTAG) standard and supports reading Instrument Connectivity Language (ICL), performing automatic retargeting of IJTAG-specific PDL commands such as iWrite and iRead, and

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

writing out the retargeted PDL patterns in many different ATE tester specific formats and as Verilog HDL simulation testbenches. See the *Tessent IJTAG User's Manual* for complete information.

- Tessent LogicBIST contains all of the functionality needed to implement the Hybrid TK/LBIST Flow for your design. The tool enables you to share the logic between EDT and LBIST controllers that can be configured to work either in EDT or LBIST mode. The product enables you to create a BIST-ready design, generates the hybrid IP, performs fault simulation, and creates top-level serial patterns to validate the LBIST mode.
- Tessent MemoryBIST provides a complete solution for at-speed testing, diagnosis, repair, debug, and characterization of embedded memories. The solution's architecture is hierarchical, enabling BIST and self-repair capabilities to be added to individual cores as well as at the top level.
- Tessent MissionMode provides a combination of automation and on-chip IP for enabling semiconductor chips throughout an automotive (or other safety-critical) electronics system to be tested and diagnosed at any point during the system's functional operation.
- Tessent SiliconInsight provides interactive capabilities for testing, debugging, and characterizing BIST-tested memories and logic, IEEE 1687 IJTAG instruments, and ATPG patterns. It also facilitates on-tester data collection and diagnosis for BIST-tested memories and logic.
- Tessent DesignEditor is a less expensive product that provides access only to design introspection and editing features. This product only provides access to the dft context (commands labeled "dft" or "all contexts").
- Tessent Scan is an internal scan synthesis tool that identifies and inserts scan and test circuitry into your design. For more information, refer to the *Tessent Scan and ATPG User's Manual*.
- Tessent ScanPro is a tool for inserting test logic to make the design scan and BIST ready. It includes the scan stitching capabilities that are part of Tessent Scan, but in addition includes capabilities for inserting test points to improve pattern counts and test coverage, X-bounding to prevent X's from propagating in the design, and inserts logic to isolate false and multicycle paths.
- Tessent TestKompress uses patented scan EDT™ (Embedded Deterministic Test) technology and many of the industry leading ATPG features of Tessent FastScan, to reduce scan data volume and test time by many times over Tessent FastScan ATPG. It is for users who need greater reductions in ATE memory and time requirements than are available with any other full-scan ATPG tool.
- Tessent Visualizer, which is built into all of these tools, is a schematic and data viewing utility. Tessent Visualizer adds the ability to graphically investigate and interact with

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

designs, making it easier to debug problems. For more information, see "Tessent Visualizer" in the *Tessent Shell User's Manual*.

In Tessent Shell, the term "context" refers to a broad category of functionality that often corresponds to a specific product and a license feature, such as Tessent FastScan. By setting the context, you indicate the type of task you want Tessent Shell to perform. For more information about Tessent Shell contexts, refer to the "Context and System Modes" section in the *Tessent Shell User's Manual*.

Note

Design editing commands are not available when using product licenses for Tessent FastScan and Tessent Scan. For more information about the design editing commands, refer to "Design Editing" in the *Tessent Shell User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Product Introduction

Tessent Shell is a Tcl shell environment and design data model that provides a unified Tcl command set and command naming conventions.

Tessent Shell also provides access to the following products:

Tessent BoundaryScan 62
Tessent Diagnosis 63
Tessent FastScan and Tessent TestKompress 64
Tessent IJTAG 66
Tessent LogicBIST 67
Tessent MemoryBIST 67
Tessent MissionMode 68
Tessent Scan 69
Tessent ScanPro 70
Tessent SiliconInsight 70
Tessent DesignEditor 71

Tessent BoundaryScan

Tessent BoundaryScan is a complete solution for the creation and integration of boundary scan cells and related control logic for embedded test and diagnosis of integrated circuit I/Os, as well as test and diagnosis of board-level interconnects between ICs. Tessent BoundaryScan provides a completely automated solution for adding standard boundary scan support to ICs of any size or complexity, reducing IC engineering development effort and improving time-to-market.

Tessent BoundaryScan supports standard 1149.1 boundary scan cells, 1149.1 custom boundary scan cells, and optionally 1149.6 boundary scan cells for differential I/O cells driving AC-coupled nets. It also provides a unique 1149.1-based solution for contactless testing of I/Os. For complete information, refer to the *Tessent BoundaryScan User's Manual*.

Tessent BoundaryScan Inputs

To use this flow, you must have either an RTL or a gate-level netlist with I/O pads already inserted into the design. For an RTL netlist, you must have the Tessent cell library or the pad library for the pad cells. For a gate-level netlist, you must have the Tessent cell library or the ATPG library for the standard cells, in addition to the Tessent cell library for the I/O pad cells. The pad library supported by Tessent BoundaryScan-LV is natively supported in Tessent Shell if the Tessent cell library for the I/O pad cells is not present.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tessent BoundaryScan Outputs

The tool creates and interconnects RTL-level boundary scan logic compliant with the IEEE 1149.1-2001 standard.

- **Instruction Support —** Full support of required IEEE standards 1149.1 instructions.
- **Extension Support —** Support of base extensions to IEEE 1149.1, such as the Device ID register.
- **Compliant Verilog —** Generation of Verilog (IEEE 1364-2001) that is compliant with Questa® SIM (Verilog), Synopsys' Design Compiler, and other industry synthesis tools.
- **RTL-Level Boundary Scan Generation —** Insertion and interconnection of boundary scan circuitry at the RTL level, moving generation of test circuitry to earlier in the design process.
- **Customized Boundary Scan —** Generation of default or user-customized boundary scan architectures.
- **Automatic Connection —** Automatic connection of boundary scan to internal scan logic.
- **Testbench Generation —** Generation of a testbench to test the boundary scan logic after interconnection with the core application logic.
- **Test Vector Generation —** Generation of boundary scan test vectors in a variety of ASIC vendor test data formats, as well as ASCII, binary, STIL, WGL and other pattern formats.
- **Setup File Generation —** Generation of ATPG setup files, for designs with generated boundary scan circuitry controlling internal scan circuitry.
- **Compliant BSDL —** Production of BSDL output that is compliant with IEEE standard IEEE 1149.1-2001 specification.

Tessent Diagnosis

Tessent Diagnosis-specific features include many capabilities.

Some of these capabilities are:

- Fault model independent analysis for suspect identification, suspect scoring, and classification.
- Logic and chain diagnosis.
- Direct diagnosis with compressed patterns.
- Links to Calibre® for physical view.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tessent Diagnosis Inputs

Tessent Diagnosis has the following inputs:

- **Design** — This must be a flattened design, created previously using the write_flat_model command in Tessent FastScan or Tessent TestKompress.
- **Test Patterns** — STIL, WGL, binary or ASCII version of the patterns applied on the tester.
- **Failure file** — This file contains ATE failure data presented in the correct Tessent failure file format for the diagnose_failures command.

Tessent Diagnosis Outputs

Tessent Diagnosis produces the following outputs:

- **Failure Diagnosis Report** — This is a human-readable diagnosis report that is displayed onscreen or written in ASCII format to a file. Additionally, the report can be written to a file in Comma Separated Value (CSV) format.

Tessent FastScan and Tessent TestKompress

Tessent FastScan and Tessent TestKompress share a great deal of functionality. Tessent TestKompress contains the entire Tessent FastScan feature set (the “patterns -scan” context), and in addition provides functionality for EDT IP creation (the “dft -edt” context).

Both products share the following features:

- Can be used within a Tessent flow or as a point tool in other design flows.
- Contain an internal high-speed fault simulator.
- Produce a number of standard test pattern data formats.
- Contain a powerful design rules checker.
- Read most standard gate-level netlists.
- Produce very high coverage test pattern sets for full-scan and scan-sequential designs. Scan-sequential designs contain well-behaved sequential scan circuitry, including non-scan latches, sequential memories, and limited sequential depth.
- Contain functionality for handling embedded RAM and ROM.
- Support the same pattern types and fault models.
- Provide the same test coverage.
- Provide the same diagnostics.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Contain the same command set, with the exception that Tessent TestKompress contains additional commands for EDT IP creation and use.

In addition to the previous features, Tessent TestKompress includes the following:

- Increases production throughput: scan test time is much shorter than any other scan methodology.

Tessent FastScan and Tessent TestKompress Inputs

Tessent FastScan and Tessent TestKompress have the following inputs:

- **Design** — The supported netlist formats are: gate-level Verilog, RTL Verilog, RTL System Verilog, and RTL VHDL.
- **Test Procedure File** — This file defines the operation of the scan circuitry in your design. You can generate the file by hand or by using the write_atpg_setup command in Tessent Scan.
- **Library** — This file contains model descriptions for all library cells used in your design.
- **Test Patterns** — This is a set of externally generated test patterns that you can use as the pattern source for simulation.

Tessent TestKompress has the following inputs:

- EDT IP Creation Phase:
  - **Design** — This must be a gate-level netlist with scan. The supported netlist format is Verilog.
  - **Test Procedure File** — This file defines the operation of the scan circuitry in your design. You can generate the file by hand or by using the write_atpg_setup command in Tessent Scan.
  - **Library** — This file contains model descriptions for all library cells used in your design.
- EDT Pattern Generation Phase:
  - **Design** — This must be a scan-inserted, gate-level netlist with EDT circuitry. The supported netlist format is Verilog.
  - **EDT Dofile** — Created by Tessent TestKompress in the IP Creation Phase, this file contains setup commands for the Pattern Generation Phase.
  - **EDT Test procedure file** — Created by Tessent TestKompress in the IP Creation Phase, this file contains test procedure steps for the Pattern Generation Phase.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **Library** — This file contains model descriptions for all library cells used in your design.

Tessent FastScan and Tessent TestKompress Outputs

Tessent FastScan and Tessent TestKompress produce the following outputs:

- **Test Patterns** — This file set contains test patterns in one or more of the supported simulator or ASIC vendor pattern formats. For more information on the available test pattern formats, refer to the write_patterns command reference page within this manual, or the "How to Save the Test Patterns" section in the *Tessent Scan and ATPG User's Manual*.
- **ATPG Information Files** — These files contain session information that you can save using various commands.
- **Fault List** — This is an ASCII file that contains internal fault information in the standard Tessent fault format.

Tessent TestKompress produces the following outputs:

- EDT IP Creation Phase:
  - **EDT IP Files** — These files contain the EDT IP described in Verilog or VHDL RTL, as well as connectivity and a black box representation of the core design.
  - **Design Compiler Synthesis Script** — This is a synthesis script, typically used as a template for synthesizing the EDT IP into the design.
  - **EDT Dofile** — This file contains setup commands for the EDT Pattern Generation Phase.
  - **EDT Test Procedure File** — This file contains test procedure steps for the Pattern Generation Phase.
  - **Bypass Files** — These files consist of a dofile and a test procedure file for optionally performing Tessent FastScan ATPG on the design with EDT.
- EDT Pattern Generation Phase:
  - **Test Patterns** — This file set contains test patterns in one or more of the supported simulator or ASIC vendor pattern formats. For more information on the available test pattern formats, refer to the write_patterns command description in this manual.

Tessent IJTAG

Tessent IJTAG is the Siemens EDA implementation of the IEEE 1687-2014 (IJTAG) standard for PDL command retargeting and ICL extraction.

For complete information, refer to IEEE 1687-2014 (IJTAG) term in the *Tessent Glossary* and to the *Tessent IJTAG User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tessent IJTAG Inputs

Tessent IJTAG has the following input:

- **ICL Files** — Instrument Connectivity Language (ICL) descriptions of your design.

Tessent IJTAG Outputs

Tessent IJTAG produces the following outputs:

- **Test Patterns** — Retargeted PDL patterns in many different ATE tester specific formats and Verilog HDL simulation testbenches.
- **Extracted ICL Files** — Top-level ICL expressing the interconnection of IJTAG instruments within a design netlist.

Tessent LogicBIST

Tessent LogicBIST contains all the features that are needed to implement Hybrid TK/LBIST Flow in a design.

The product includes the following features:

- Insertion of DFT into the design to make it BIST-ready. This involves scan stitching, x-bounding, and bounding of multicycle and false paths.
- Test point insertion to improve the random pattern testability of the design.
- Hybrid TK/LBIST generation and insertion into the incoming design. This includes sharing the logic between the EDT and LBIST controllers, thereby reducing the overall area overhead. Additionally, the tool also generates logic that enables you to do low power shift during LBIST.
- IJTAG network insertion and extraction, which is necessary to set up Logic BIST and observe the MISR signature.
- Fault simulation of the pseudo-random patterns applied during an LBIST session.
- Generation of top-level patterns that enables merging of patterns for multiple LBIST controllers.

Tessent MemoryBIST

Tessent MemoryBIST provides a complete solution for at-speed testing, diagnosis, repair, debug, and characterization of embedded memories. The solution's architecture is hierarchical, enabling BIST and self-repair capabilities to be added to individual cores as well as at the top level.

For complete information, refer to the *Tessent MemoryBIST User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tessent MemoryBIST Inputs

To insert memory BIST into your design, you must have either an RTL or a gate-level netlist as well as memory BIST libraries.

If the design contains standard cells, the Tessent cell library or ATPG library is also required. For other IP blocks that do not have either RTL or library models, the simulation model can be loaded using the read_verilog or read_vhdl command with the -interface_only option. This instructs the tool to ignore the internals of all modules specified in the filename argument and extract only the module port definitions and parameters.

Tessent MemoryBIST Outputs

Using Tessent MemoryBIST, you can perform the following:

- Test multiple memories using one memory BIST controller that has one or more BIST steps, where BIST steps are run in sequence.
- Test memories in parallel in one BIST step or in sequence in several BIST steps.
- Define one or more custom test memory algorithms that are hard coded into the memory BIST controller.
- Choose memory test algorithms from the Tessent library of algorithms to be hard coded into the memory BIST controller.
- Run the memory BIST controller for all steps with the specific algorithms assigned at generation time (default configuration).
- Run the memory BIST controller in diagnostic mode where you can freeze on a specific BIST step, specific memory test port, or specific error count.
- Select a hard-coded memory BIST algorithm to be applied to a specific memory BIST step at the tester.
- Select an algorithm from a library of algorithms to be applied to a specific memory BIST step.
- Define a custom algorithm at tester time to be applied to a specific memory BIST step.
- Perform repair analysis on memories implementing different redundancy schemes such as row only, column only, or row and column.

Tessent MissionMode

Tessent MissionMode enables communication between any external CPU bus and all on-chip test resources for system-controlled on-line test and diagnosis. The on-chip test resources can consist of any Tessent BIST or DFT capabilities (such as MemoryBIST, LogicBIST or EDT logic) or any third-party IJTAG-compliant instruments.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This product is part of the Mentor Safe tool suite that enables ISO 26262-compliant designs for advanced driver-assistance systems (ADAS) and autonomous driving capabilities for passenger cars, as well as for other safety-critical applications such as aerospace and medicine. The tool targets IC reliability by providing advanced self-test and monitoring capabilities once the IC is packaged and deployed in the device.

Tessent Scan

Tessent Scan tool-specific features include features that support scan insertion.

Specifically, Tessent Scan supports the following functionality:

- Support for full-scan identification and insertion.
- Support for Mux-scan methodology.
- Powerful design rules checking.
- Automatic generation of the scan setup dofile and the test procedure files for use downstream in your flow with Tessent FastScan and Tessent TestKompress ATPG products.
- Display of a variety of information—from design and debugging information to statistical reports for the test set you generate.

Tessent Scan Inputs

Tessent Scan has the following inputs:

- Design Netlist — In Verilog format.
- Test Procedure File — Required if you have existing scan circuitry in your design. The test procedure file defines the operation of existing scan circuitry. See also "How to Specify Existing Scan Information" in the *Tessent Scan and ATPG User's Manual*.
- DFT Library — Contains the model descriptions for all library cells used in your design, along with the model descriptions for all the scan replacement cells.
- **Command Dofile** — A text file containing a list of Tessent Scan commands. You can also enter these commands interactively.

Tessent Scan Outputs

Tessent Scan produces the following outputs:

- Design Netlist — A scan version of your design netlist that can be in Verilog format.
- ATPG Setup Files — The test procedure file, which defines the operation of the scan circuitry in your design, and a dofile for setting up the design and scan circuitry information for ATPG operations.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tessent ScanPro

Tessent ScanPro includes features that are necessary for scan insertion (part of Tessent Scan), as well as all the advanced DFT insertion features to facilitate scan testing and Logic BIST.

Specifically, Tessent ScanPro supports the following functionality:

- Includes all features that are part of Tessent Scan.
- Generation of test points for pattern reduction as well as test coverage improvement.
- Ability to identify and bound all Xs in the designs. This is needed to reduce the number of Xs that can propagate into the design and can affect compression as well as Logic BIST runs.
- Ability to read in SDC and add DFT to block transitions propagated through Multicycle Paths (MCPs) and False Paths (FPs) from being captured on scan cells.

Tessent ScanPro Inputs

These are the same as Tessent Scan.

Tessent ScanPro Outputs

These are the same as Tessent Scan, with the addition of the following:

- Scan Setup Files — This is the dofile and test procedure file that are required to perform incremental scan stitching of the test point flops.
- Test Point Dofile — This is the file that writes out all the test points that the tool is ready to insert into the design. This is needed when you want just to perform the analysis in the current session but would like to get back to insertion at a later time.

Tessent SiliconInsight

Tessent SiliconInsight provides interactive capabilities for testing, debugging, and characterizing BIST-tested memories and logic, IEEE 1687 IJTAG instruments, and ATPG patterns. It also facilitates on-tester data collection and diagnosis for BIST-tested memories and logic.

For complete information, refer to the *Tessent SiliconInsight User's Manual for Tessent Shell*.

Tessent SiliconInsight operates within the Tessent Shell environment. The tool uses an open ATE interface to interact with embedded instruments for both pre-silicon, desktop, and ATE applications. The open interface is called the Characterization and Debug Package (CDP). The CDP contains test patterns, procedural data and side files that enable access and control of any instrument at any hierarchy level.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For more detailed characterization activities, you can also control instruments such as power supplies and clock generators with interface buses implemented according to the General Purpose Interface Bus (GPIB) standard IEEE-488. To do this, connect a USB-to-GPIB adaptor to a GPIB instrument, which you then connect to the performance board.

Tessent DesignEditor

Tessent DesignEditor features contain a subset of Tessent Shell features.

The product includes the following features:

- Access to design introspection and editing features while consuming a less expensive product license.
- Access to the commands that can operate in the dft context (commands labeled “dft” or “all contexts”).

Tessent DesignEditor Inputs

Tessent DesignEditor has the following inputs:

- **Design** — The supported netlist formats are: gate-level Verilog, RTL Verilog, RTL System Verilog, and RTL VHDL.
- **Library** — This file contains model descriptions for all library cells used in your design.

Tessent DesignEditor Outputs

Tessent DesignEditor produces the following outputs:

- **Design** — The supported netlist formats are: gate-level Verilog, RTL Verilog, RTL System Verilog, and RTL VHDL.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 2
Tool Invocations

Tool invocation commands invoke tools that create, edit, manipulate, or diagnose test patterns for DFT applications. These commands are issued from the input window of your Linux workstation and, by default, open in a command line session.

Tool Invocation Command Descriptions **74**
create_skeleton_design 75
stil2tessent. 77
tessent 84
tessent -diagserver. 85
tessent -shell 88
tessent -siliconinsight 92
tessent -visualizer 93
tessent -yieldinsight 95

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tool Invocation Command Descriptions

The notational conventions used to describe the tool invocation commands are the same for all products. Do not enter any of the special notational characters (such as, {}, [], or |) when typing the command.

Table 2-1 describes the notational conventions used in this manual.

Table 2-1. Conventions for Command Line Syntax

Convention	Example	Usage
UPPercase	-STatic	Required argument letters are in uppercase; in most cases, you may omit lowercase letters when entering literal arguments, and you need not enter in uppercase. Arguments are normally case insensitive.
Boldface	set_fault_mode **Uncollapsed** | **Collapsed**	A boldface font indicates a required argument.
[ ]	exit [-force]	Square brackets enclose optional arguments. Do not enter the brackets.
Italic	dofile ***filename***	An italic font indicates a user-supplied argument.
{ }	add_fault_sites {**-ALl** | **-UNDEFINED_Cells** }[-VERBose]	Braces enclose arguments to show grouping. Do not enter the braces.
|	add_fault_sites {**-ALl** | **-UNDEFINED_Cells** }[-VERBose]	The vertical bar indicates an either/or choice between items. Do not include the bar in the command.
Underline	set_au_analysis **ON** | **OFf**	An underlined item indicates either the default argument or the default value of an argument.
…	add_clocks *off_state* ***pin_list*** …	An ellipsis follows an argument that may appear more than once. Do not include the ellipsis when entering commands.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_skeleton_design

Invokes Tessent TestKompress and creates files necessary to integrate Tessent TestKompress logic (TestKompress logic) into a design at the RTL stage.

Usage

create_skeleton_design -i *skeleton_design_input_file*
[-o *output_file_prefix*]
[-simulation_library]
[-design_interface *filename*]
[-module_name *module_name*]
[-help [-verbose]]

Description

Invokes Tessent TestKompress and creates files necessary to integrate Tessent TestKompress logic (TestKompress logic) into a design at the RTL stage.

This command requires a special design skeleton input file that you create manually. For more information about this file, see "Integrating Compression at the RTL Stage" in the *Tessent TestKompress User's Manual.*

Arguments

- -i *skeleton_design_input_file*

  A required switch and string pair that specifies the pathname of the skeleton design input file.

- -o *output_file_prefix*

  An optional switch and string pair that specifies a naming prefix for the output files as follows:

  ```
  <output_file_prefix>.v
  <output_file_prefix>.dofile
  <output_file_prefix>.testproc
  <output_file_prefix>.atpglib
  ```

  By default, "test" is used for the naming prefix.

- -simulation_library

  An optional switch that creates a Verilog simulation library from the specified cell library.

- -design_interface *filename*

  An optional switch and string pair that specifies the pathname of the file containing a Verilog description of the design interface. The Verilog description file is used when the TestKompress logic is placed externally to the design core.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -module_name *module_name*

  An optional switch and string pair that specifies the name of the generated skeleton design module. When this switch is not specified, the default module name is “skeleton_design_top”.

  This switch is useful when a design created using the skeleton flow contains multiple EDT blocks. Normally, this would result in all of the EDT blocks having the same name, skeleton_design_top_edt. You can use this switch during the skeleton design generation step to provide different names for different EDT blocks.

- -help

  An optional switch that lists all the invocation switches.

- -verbose

  An optional switch that, together with -help, lists all the invocation switches and a brief description of each.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

stil2tessent

A utility that converts STIL files to dofiles and test procedure files for Tessent FastScan and Tessent TestKompress. The converted files define clocks, scan chains, scan groups, and pin constraints. It also converts CTL files to the Tessent Core Description (TCD) format used by Tessent Scan. Another usage is for converting memory CTL descriptions to the TCD format used by Tessent MemoryBIST.

Usage

Usage 1

```
stil2tessent {-STil stil_filename [-TPf tpf_filename] [-DOfile dofile_name]
    [-ALias Min | All] [-CApture NOne | Single | NAmed]
    [-EDGe_strobe_processing  ON | OFf ]
    [-INFer_pulse_clock ON | OFf]
    [-EXec exec_name]
    [-WRITE_CORE_DESCription
        [-OUTPUT_DIRectory dir_path]
        [-TCD_SCAN_EXTension ext]
        [-CTL_mode mode_name]
        [-CORE_name name]
    ]
    [-LOgfile logfile_name [-REplace]]} |
    [-help | -Usage | -MANual | -Version]
    [-Verbose]
```

Usage 2: Memory CTL to TCD Format Conversion

```
stil2tessent {-STil stil_filename
    [-WRITE_CORE_DESCription
        [-OUTPUT_DIRectory dir_path]
        [-TCD_MEMORY_EXTension ext]
        [-TCD_SCAN_EXTension ext]
        [-SKIP_TCD_SCAN_WRAPPER name]
        [(-ENVIRONMENT_name | -CORE_name) name]
        [-CTL_mode mode_name]
        [-TCD_NAME_OVERRIDE name]
        [-TCD_MEMORY_OVERRIDE overrides]
    ]
    [-LOgfile logfile_name [-REplace]]} |
    [-help | -Usage | -MANual | -Version]
    [-Verbose]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Description

Usage 1

For Tessent FastScan and Tessent TestKompress, invokes stil2tessent and produces a dofile that defines clocks, scan chains, scan groups, and pin constraints. This tool also creates test procedure files with a timeplate and the standard test_setup, load_unload, and shift scan procedures.

This command translates STIL signal groups, which are used in timeplates or procedures, into "alias" statements in the test procedure file. It also produces a test procedure file timeplate definition for each WaveformTable in the STIL file, exactly matching the timing specified in the STIL file.

Any procedure or macro with a name that matches a Tessent procedure type (load_unload, master_observe) is translated into that procedure. Any procedure or macro in the STIL file that has "capture" in its name is translated into a named capture procedure when you use the "-capture named" switch.

Any other macro whose name does not match anything that Tessent uses is translated into an unused sub-procedure.

For Tessent Scan, invokes stil2tessent to generate a Tessent Core Description (TCD) file with a populated Scan wrapper when you use the -write_core_description switch. This TCD file, also known as tcd_scan, enables stitching of scan segments defined in the input CTL file.

For more information about translation details, refer to "Notes About Using the stil2tessent Tool" in the *Tessent Shell User's Manual*.

Note

Refer to the Core/Scan page for details of the generated tcd_scan file. IEEE 1450.6 (CTL) describes how to define "DataType Constant" pins. The generated tcd_scan file represents these pins with a TestEn wrapper.

If the CTL description specifies "InputProperty SynchLatch/SynchFF" for the "DataType ScanDataIn" signal or "OutputProperty SynchLatch/SynchFF" for the "DataType ScanDataOut" signal, the polarity of scan_in_clock and scan_out_clock is inverted with respect to the one recovered from the ScanMasterClock, CaptureClock, and LaunchClock statements in the CTL description.

Usage 2

Invokes the stil2tessent utility and converts a memory CTL description to the Tessent Core Description (TCD) format used by Tessent MemoryBIST. The IEEE 1450.6.2 Standard for Memory Modeling in Core Test Language (memory CTL) is an extension of the IEEE 1450.6 Core Test Language (CTL) that can be used to describe the test and repair related aspects of memories. The stil2tessent utility options described in Usage 2 are used to process the memory

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

CTL description and to write out the equivalent memory TCD format in preparation for MemoryBIST insertion.

Arguments

- **-STil *stil_filename***

  A required switch and string pair that specifies the name of the input STIL file. The supplied STIL file may be a STIL Procedure File (SPF) or a Core Test Language (CTL) file.

- **-TPf *tpf_filename***

  A required switch and string pair that specifies the name of the test procedure file to generate. If not specified, the *tpf_filename* is named by appending ".proc" to the STIL Test Procedure filename.

- **-DOfile *dofile_name***

  A required switch and string pair that specifies the name of the dofile to generate. If not specified, the *dofile_name* is named by appending ".dof" to the STIL Test Procedure filename.

- -ALias Min | All

  An optional switch and literal pair that produces the minimum required alias statements in the Tessent Procedure file or to produce alias statements for all signal groups found in the STIL Test Procedure file. The default is to produce the minimum required alias statements.

- -CApture NOne | Single | NAmed

  An optional switch and literal pair that adds information to the Tessent Procedure file. This switch either produces no capture procedures, a single default capture procedure, or multiple named capture procedures for each capture Macro found in the STIL Test Procedure File. The default is to produce no capture procedures.

- -EDGe_strobe_processing <u>OFf</u> | ON

  An optional switch and literal pair that specifies the type of strobe processing, edge strobe or window strobe, used to measure events.

  <u>ON</u>

  When you specify ON, the tool processes H and L waveform events in the SPF file to indicate edge strobe timing.

  <u>OFf</u>

  When you specify OFF, edge strobe timing is not created. This is the default

- -INFer_pulse_clock <u>ON</u> | OFf

  An optional switch and literal pair to specify that stil2tessent uses the generic pulse_clock statement for clocks whose timing edges are identical.

  <u>ON</u>

  When you specify ON and the timing edges for all of the clock edges in the SPF file are identical, the produced timeplates uses the pulse_clock statement. If the timing

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

edges are not identical for each clock, the produced timeplate uses individual pulse statements. This is the default

OFf

When you specify OFf, individual pulse clock statements are used for each clock.

- -EXec ***exec_name***

  An optional switch and string pair to specify the name of the STIL PatternExec used when the STIL file has more than one PatternExec block. An error is reported when this switch is missing and more than one PatternExec blocks exist in the STIL file.

- -WRITE_CORE_DESCription [-OUTPUT_DIRectory *dir_path*] [-TCD_SCAN_EXTension *ext*] [-SKIP_TCD_SCAN_WRAPPER *name*] [-CTL_mode *mode_name*] [(-ENVIRONMENT_name | -CORE_name) *name*] [-TCD_MEMORY_EXTension *ext*] [] [-TCD_NAME_OVERRIDE *name*] [-TCD_MEMORY_OVERRIDE *overrides*]

  An optional switch and options that write sub-chain definitions into the *tcd_scan* file, and memory test and repair configurations into the *tcd_mem_lib* file. If you specify this switch but do not provide the CTL block in the input file, the tool reports an error. Choose any of the following options:

  Note

  Refer to Tessent Core Description/Scan for details of generated tcd_scan file.

  IEEE1450.6 (CTL) allows you to define “DataType Constant” pins. They are represented using a TestEn wrapper in a generated *tcd_scan* file.

  -OUTPUT_DIRectory *dir_path* — An optional switch and string pair that specifies the destination directory where the tool writes the TCD files. By default, the tool writes the generated TCD files to the current directory. If the specified directory path does not exist, then the tool creates the directory.

  -TCD_MEMORY_EXTension *ext* — An optional switch and string pair that specifies the file extension of the generated memory TCD file. The default file extension is “*tcd_mem_lib*”. The output filename is created by appending the extension to the Environment block name.

  -TCD_SCAN_EXTension *ext* — An optional switch and string pair that specifies the file extension of the generated scan TCD file. The default file extension is *tcd_scan*. The tool creates the output filename by appending the extension to a core name or Environment block name.

  -SKIP_TCD_SCAN_WRAPPER *name* — An optional switch and string pair that specifies the name of a wrapper that should be skipped in the generated scan TCD file. The supported value for <*name*> is “TestEn”.

  -CTL_mode *mode_name* — An optional switch and string pair that specifies the test mode name corresponding to the scan chain information that the tool reads for Usage 1. For Usage 2, This pair specifies the CTLMode block for translation. If there is more than one CTLMode block in the Environment block, this switch is required to select the desired CTLMode block.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

-(ENVIRONMENT_name | CORE_name) *name* — An optional switch and string pair that specifies a core name or an Environment block from the available values for translation. If there is more than one Environment block in the input CTL file, this switch is required to select the desired Environment block.

-TCD_NAME_OVERRIDE *name* — An optional switch and string pair that is used to override the Environment name during the TCD file generation. The string value specified becomes the TCD core name and the TCD filename base.

-TCD_MEMORY_OVERRIDE *overrides* — An optional switch and list to specify certain top-level TCD memory properties during the TCD generation phase. The value of this switch must be a space-separated list of property names and values, in the form "*key1 value1 key2 value2...*". The properties that can be overridden are:

- DataOutHoldWithInactiveReadEnable
- MemoryHoldWithInactiveSelect
- MilliWattsPerMegaHertz
- RetentionTimeMax
- PipelineDepth
- MinHold

The only properties overridden are those specified with this switch, unspecified properties are not modified. For more information on these properties, refer to the Memory TCD wrapper description in *Tessent MemoryBIST User's Manual for Use With Tessent Shell*.

- -LOgfile *logfile_name*

  An optional switch and string pair that specifies the name of the file to write all session information.

- -REplace

  An optional switch that overwrites the -Logfile *logfile_name* if one by the same name already exists.

- -help

  An optional switch that displays a message that contains all the stil2tessent switches and a brief description of each.

- -Usage

  An optional switch that displays the usage syntax for the stil2tessent command.

- -MANual

  An optional switch that opens the Tessent Documentation System.

- -Version

  An optional switch that displays the version of the stil2tessent tool that you currently have available.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example illustrates Usage 1 and runs the stil2tessent tool on *design.spf*.

```
stil2tessent -stil design.spf -tpf new_design.tpf \
             -dofile new_dofile.do
```

Example 2

The following example illustrates Usage 1 and demonstrates invoking the tool for single-mode output:

```
stil2tessent -stil ../data/multi_mode.ctl \
             -ctl_mode scan_mode \
             -write_core_description
```

In the example, the tool writes the tool-generated files to the current directory.

Example 3

This example illustrates Usage 1 and demonstrates multi-mode output invocation. In the example, the tool writes the generated file as *./outDir/block.tcd*. This file contains a description of all modes containing valid scan chain data.

```
stil2tessent -stil ../data/multi_mode.ctl \
             -output_directory ./outDir \
             -tcd_scan_extension tcd \
             -core_name block \
             -write_core_description
```

Example 4

The following example illustrates Usage 1 and sources Tessent Shell tool-generated files:

```
set_context dft -scan
# (...) loading design
set_design_sources -format tcd_scan -Y ./ -extensions tcd_scan
```

Example 5

The following examples demonstrate Usage 2 for converting a memory CTL description to a TCD format using various settings that direct file location and naming. For this example, the memory CTL description is in the file *SYNC_1R1W_16x8.ctl* and contains a single Environment block named SYNC_1R1W_16x8. Invoking *stil2tessent* as shown generates the *SYNC_1R1W_16x8.tcd_mem_lib* TCD file in the current directory:

```
stil2tessent -stil SYNC_1R1W_16x8.ctl \
             -write_core_description
```

If you want to direct the generated TCD file to a subdirectory named *libraries*, the following command option is added:

```
stil2tessent -stil SYNC_1R1W_16x8.ctl \
             -write_core_description \
             -output_directory libraries
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The TCD file extension can be changed as needed. For this example, if the TCD file is to be written in the current directory with the filename *SYNC_1R1W_16x8.lib*, the following command is run:

```
stil2tessent -stil SYNC_1R1W_16x8.ctl \
             -write_core_description \
             -tcd_memory_extension lib
```

The TCD file and module name can also be changed. For this example, to convert the content of the memory CTL file described above to a TCD file named *SRAM16.tcd_mem_lib* that contains a TCD module named SRAM16, the following command is run:

```
stil2tessent -stil SYNC_1R1W_16x8.ctl \
             -write_core_description \
             -tcd_name_override SRAM16
```

Example 6

The following Usage 2 example demonstrates converting a specified memory CTL Environment block for TCD file conversion. Given a memory CTL description named *all_cores.ctl* that contains multiple Environment blocks, the following command converts the Environment block named SYNC_1R1W_16x8 to a TCD file:

```
stil2tessent -stil all_cores.ctl \
             -write_core_description \
             -environment_name SYNC_1R1W_16x8
```

Example 7

This Usage 2 example demonstrates converting a certain CTLMode in the memory CTL file to a TCD file. For this example, the memory CTL description is in the file *SYNC_1R1W_16x8.ctl* and contains multiple CTLMode blocks within a single Environment block. To convert the CTLMode block named MBIST to the TCD file format, the following command is run:

```
stil2tessent -stil SYNC_1R1W_16x8.ctl \
             -write_core_description \
             -ctl_mode MBIST
```

Example 8

This Usage 2 example demonstrates overriding top-level TCD properties in the TCD file conversion. For this example, the memory CTL description is in the file *SYNC_1R1W_16x8.ctl* and contains one Environment block named SYNC_1R1W_16x8. To convert the contents to a TCD file with PipelineDepth overridden to 2 and MinHold to 1.5, the following command is run:

```
stil2tessent -stil SYNC_1R1W_16x8.ctl \
             -write_core_description \
             -tcd_memory_override "PipelineDepth 2 MinHold 1.5"
```

The PipelineDepth and MinHold properties default to 0 when they are unspecified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

tessent

Invokes a Tessent tool in a command line session.

Usage

tessent {**-shell** [*options*] | **-diagserver** [*options*]| **-yieldinsight** [*options*]
| **-siliconinsight** {**-ate** | **-desktop**} [*option*] | **-visualizer**[*options*]}
| [-help]
| [-manual]

Description

Invokes a Tessent tool in a command line session.

Arguments

- **-shell** *options*

  A switch and options that invokes the Tessent Shell. See "tessent -shell" on page 88.

- **-diagserver** *options*

  A switch and options that invokes the Tessent Diagnosis server. See "tessent -diagserver" on page 85.

- **-yieldinsight** *options*

  A switch and options that invokes Tessent YieldInsight. See "tessent -yieldinsight" on page 95.

- **-siliconinsight -ate** | **-desktop** *option*

  A switch and literal pair that invokes Tessent SiliconInsight. See "tessent -siliconinsight" on page 92.

- **-visualizer** *options*

  A switch and options that invokes Tessent Visualizer. See "tessent -visualizer" on page 93.

- -help

  An optional switch that lists the switches for the tessent command.

- -manual

  An optional switch that opens the Tessent Documentation System.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

tessent -diagserver

Invokes the Tessent Diagnosis server.

Usage

tessent **-DIAGSERVER**
[-daemon [*daemon_id*]]
[-dlist]
[{-dstatus | -dexit | -dterminate} *daemon_id*]
[-HDB *history_db*]
[-DOFile *dofile_name*... [-HIStory]]
[-LOGfile *logfile_name* [-REPlace]]
[-Arguments *name=value* ...]
[-license_wait *minutes* | none | unlimited]
| [-HELP] | [-USAGE] | [-MANUAL] | [-VERSION]

Description

Invokes the Tessent Diagnosis server. See "Running Tessent Diagnosis Server" in the *Tessent Diagnosis User's Manual*.

Arguments

- **-DIAGSERVER**

  A required switch that invokes the Tessent Diagnosis server.

- -daemon [*daemon_id*]

  An optional switch that invokes the Tessent Diagnosis server in daemon mode. The tool automatically assigns a process ID. Using the optional *daemon_id*, you can specify a process ID for the daemon.

- -dlist

  An optional switch that lists running daemons, including the id, host, and logfile for each daemon.

- -dstatus *daemon_id*

  An optional switch and integer pair that queries the status of a daemon you identify with *daemon_id* and displays the status of the monitors and analyzers.

- -dexit daemon_id

  An optional switch and integer pair that instructs the daemon to exit after the tool finishes running any diagnosis jobs.

- -dterminate daemon_id

  An optional switch and integer pair that instructs the daemon to immediately exit without finishing running any diagnosis jobs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -HDB *history_db*

  An optional switch and string that specifies the name of the History Database (HDB). The HDB is an SQL database containing information in tables. See "History Database Schema" in the *Tessent Diagnosis User's Manual*.

  Caution

  The tool overwrites an HDB with no warning if you repeat the command using the same name. See "Server History" in the *Tessent Diagnosis User's Manual* for more information.

  The HDB supports session IDs. You can append to an existing HDB by adding a "+" in front of an existing HDB name to indicate append mode. Append mode opens the HDB using a new session ID.

- -DOFile *dofile_name*... [-HIStory]

  An optional switch and repeatable string pair that specifies the name of one or more dofiles to run upon invocation. As the tool is processing each dofile, the Tcl variable "tessent_user_argv0" is assigned the name of the current dofile. -HIStory is an optional switch that adds dofile commands to the command line history list. The tool ignores the -History switch if you issue it without the -Dofile switch.

- -LOGfile *logfile_name* [-REPlace]

  An optional switch and string pair that specifies the name of the file that you want Tessent Diagnosis to write all session information to. The default is to display session information to the standard output. The version banner that indicates the tool, version, date, and platform on which the log file was produced is included at the beginning of the file.

  An optional -REPlace switch overwrites the *logfile_name* if a log file of the same name already exists.

- -Arguments *name=value* ...

  An optional switch and repeatable name/value pair that specifies variable names and values in a list named "tessent_user_arg." No spaces are permitted between the name, the equal sign, and the value. This list can later be accessed by a dofile for the purpose of controlling parameters in the dofile. For more information, refer to "Example 2" on page 89.

- -LICense_wait *minutes* | none | <u>unlimited</u>

  An optional switch and integer, or switch and literal, that specifies the tool's response if the license is unavailable.

  Choose one of the following options:

  - *minutes* — A positive integer that specifies the number of minutes to wait for the license.
  - none — A literal that directs the Tessent tool to exit immediately if no license is available.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

  - unlimited — A literal that directs the Tessent tool to wait with no time limit for a license. This is the default.

- -HELP

  An optional switch that lists the switches for the tessent command.

- -USAGE

  An optional switch that lists the switches for the tessent command.

- -MANUAL

  An optional switch that opens the Tessent Documentation System.

- -VERSION

  An optional switch that displays the version of the tessent tool you specify.

Examples

See "Introduction to Tessent Diagnosis Server" in the *Tessent Diagnosis User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

tessent -shell

Invokes a Tessent Shell environment.

Usage

tessent **-SHell**
[-LOGfile *logfile_name* [-REPlace]]
[-DOFile *dofile_name*]
[-ARGuments {*name=value*}]
[-LICENSE_WAIT {*minutes* | none | unlimited}]
[-IGNORE_STARTUP_FILE]
| [-HELP] | [-USAGE] | [-MANUAL] | [-VERSION]

Description

Invokes a Tessent Shell command line environment.

Arguments

- **-SHell**

  A switch for invoking the Tessent Shell command line environment. For more information, refer to the "Tessent Shell Introduction" in the *Tessent Shell User's Manual*.

- -LOGfile *logfile_name* -REPlace

  An optional switch and string pair that specifies the name of the file to which you want Tessent Shell to write all session information. The default is to display session information to the standard output. The version banner that indicates the tool, version, date, and platform on which the log file was produced is included at the beginning of the file.

  The optional -REPLace switch overwrites the *logfile_name* if a log file of the same name already exists.

- -DOFile *dofile_name*

  An optional switch and repeatable string pair that specifies the name of one or more dofiles to run upon invocation. As the tool is processing each dofile, the Tcl variable "tessent_user_argv0" is assigned the name of the current dofile.

- -ARGuments *name=value*

  An optional switch and repeatable name/value pair that specifies variable names and values in a list named "tessent_user_arg." No spaces are permitted between the name, the equal sign, and the value. This list can later be accessed by a dofile for the purpose of controlling parameters in the dofile. For more information, refer to "Example 2" on page 89.

- -LICENSE_WAIT *minutes* | none | unlimited

  An optional switch and integer, or switch and literal, that specifies the tool's response if the license is unavailable.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Choose one of the following options:

- *minutes* — A positive integer that specifies the number of minutes to wait for the license.
- none — A literal that directs the Tessent tool to exit immediately if no license is available.
- unlimited — A literal that directs the Tessent tool to wait with no time limit for a license. This is the default.

- -HELP

  An optional switch that lists the switches for the tessent command.
- -USAGE

  An optional switch that lists the switches for the tessent command.
- -MANUAL

  An optional switch that opens the Tessent Documentation System.
- -VERSION

  An optional switch that displays the version of the tessent tool you specify.

Examples

Example 1

The following example invokes Tessent Shell, uses the logfile *myLogfile* replacing any existing logfile with the same name, and runs the dofile *myDofile*.

```
tessent -shell -logfile myLogfile -replace -dofile myDofile
```

Example 2

The following example invokes Tessent Shell and runs a dofile named *dofile2*. The command line also specifies three variable/value pairs that are placed in a Tcl list named "tessent_value_arg" and thus made available to the dofile. It shows how to use single and double quotation marks to set variables that contains spaces. The system shell does not perform the expansion environment variable of the text surrounded by single quotes thus preserving the $, unlike the text in quotation marks:

```
% setenv I2 3
% tessent -shell -dofile dofile2 -arguments depths="9 \
        $I2" comment='Cost $US' output_dir=myOutDir
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following is the listing of *dofile2*, which demonstrates how to use the -arguments switch to set the values of variables inside the dofile. This technique enables you to reuse dofiles without modification:

```
# dofile2
# Set the default values of the control parameters
  array set params {
     depths {}
     comment ""
     output_dir "outDir"
  }
  array set params $tessent_user_arg
  # Use some params
  puts "comment = $params(comment)"
  if { [llength $params(depths)] > 0} {
    # do whatever needs to be done in this case
    puts "depth list = $params(depths)"  }
  if { ![file isdirectory $params(output_dir)] } {
    puts "Creation of directory $params(output_dir)"
    file mkdir $params(output_dir)  }
```

Here is the transcript produced by the invocation of Tessent Shell.

```
% setenv l2 3
% tessent -shell -dofile dofile2 -arguments depths="9 $l2" \
    comment='Cost $US' output_dir=myOutDir
```

```
//                              ... Tool Banner ...
//
//  command: # dofile2
//  command: # Set the default values of the control parameters
//  command: array set params {
//              depths {}
//              comment ""
//              output_dir "outDir"
//           }
//  command: array set params $tessent_user_arg
//  command: # Use some params
//  command: puts "comment = $params(comment)"
                  comment = Cost $US
//  command: if { [llength $params(depths)] > 0} {
//             # do whatever needs to be done in this case
//             puts "depth list = $params(depths)"
//           }
//           depth list = 9 3
//  command: if { ![file isdirectory $params(output_dir)] } {
//             puts "Creation of directory $params(output_dir)"
//             file mkdir $params(output_dir)
//           }
//           Creation of directory myOutDir
```

Example 3

The following example shows how to create a single shell script that invokes and passes arguments to Tessent Shell, which then runs the commands in the shell script as a regular dofile. In order for the following to work, the shell script file must have executable permissions.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The contents of the shell script, called *tScript* in this example, are as follows.

```
#!/bin/sh
#\
exec tessent -shell -dofile "$0" -arguments ${1+"$@"}

set_transcript_style off
puts "$tessent_user_arg"
# add dofile commands here
exit
```

The Linux shell interprets the second line ('#\') as a comment and then runs the third line, which invokes Tessent Shell and passes the entire shell script as a dofile. (Tessent Shell tool's Tcl interpreter ignores the second and third lines of the shell script and then interprets all following lines as normal Tessent Shell commands.)

The -arguments switch on the third line enables you to pass an arbitrary number of argument/value pairs from the shell script invocation into a standard Tcl list called "tessent_user_arg" for later parsing and use by Tessent Shell. So, for example, when you invoke the shell script as follows:

% tScript myarg1=myvalue1 myarg2= myarg3=myvalue3

The three arguments and their values are passed to Tessent Shell, and the relevant part of the transcript would appear as follows:

```
//  command: #!/bin/sh
//  command: # exec tessent -shell -dofile "$0" -arg ${1+"$@"}
//  command: set_transcript_style off
myarg1 myvalue1 myarg2 {} myarg3 myvalue3
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

tessent -siliconinsight

Invokes Tessent SiliconInsight.

Usage

tessent **-siliconinsight** {**-ate** | **-desktop**} [-help]

Description

Invokes Tessent SiliconInsight in either ATE or Desktop mode.

Depending on what you are using Tessent SiliconInsight for, refer to the following documentation:

- **LV Flow** — "Invoking Tessent SiliconInsight" in the *Tessent SiliconInsight User's Manual for the LV Flow*.
- **Tessent Diagnosis** — "Diagnosis for Tessent LogicBIST Designs" in the *Tessent Diagnosis User's Manual*.

Arguments

- **-siliconinsight -ate** | **-desktop**

  A required switch and literal pair for invoking Tessent SiliconInsight. Choose from one of the following:

  -ate — Invokes Tessent SiliconInsight ATE.

  -desktop — Invokes Tessent SiliconInsight Desktop.

- -help

  An optional switch that shows the manual pages for the tool.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

tessent -visualizer

Invokes the Tessent Visualizer GUI.

Usage

tessent **-visualizer**
[-help]
[-version]
[-logfile *file*]
[-server *hostname*]
[-tcp_port *port*]
[-authorization_code *code*]
[-label *name*]
[-import_configuration *filename*]

Description

Invokes Tessent Visualizer.

See "Tessent Visualizer" in the *Tessent Shell User's Manual*.

Arguments

- **-visualizer**

  A required switch that invokes Tessent Visualizer.

- -help

  An optional switch that displays a usage message for Tessent Visualizer. Tessent Visualizer is not started.

- -version

  An optional switch that displays the Tessent Visualizer version. Tessent Visualizer is not started.

- -logfile *file*

  An optional switch and string pair that specifies the log filename.

- -server *hostname* -tcp_port *port*

  Optional switch and string pairs that specify the remote sever host name and TCP/IP port.

- -authorization_code *code*

  An optional switch and string pair that specifies a user-supplied six-digit client/server authorization code.

- -label *name*

  An optional switch and string pair that specifies a prefix to be added to the Tessent Visualizer window titles.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -import_configuration *filename*

  An optional switch and string pair that specifies a configuration file to be loaded during Tessent Visualizer startup. If the configuration file cannot be read, is invalid, or is from an incompatible version of Tessent Visualizer, an error message is displayed and Tessent Visualizer does not start.

  See "Saving and Restoring the Session State" in the *Tessent Shell User's Manual* for more information on using configuration files.

Examples

For specific examples, refer to "Invoking Tessent Visualizer" in the *Tessent Shell User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

tessent -yieldinsight

Invokes Tessent YieldInsight.

Usage

tessent **-YIELDINSIGHT**
[-GUI]
[-NOGUI]
[-LOGfile *logfile_name* [-REPlace]]
[-DOFile *dofile_name*]
[-ARGuments {*name*=*value*}]
[-LICENSE_WAIT {*minutes* | none | unlimited}]
| [-HELP] | [-USAGE] | [-MANUAL] | [-VERSION]

Description

Invokes Tessent YieldInsight.

See the "Overview of Tessent YieldInsight" in the *Tessent YieldInsight User's Manual* for complete information.

Arguments

- **-YIELDINSIGHT**

  A required switch for invoking Tessent YieldInsight.

- -GUI

  An optional switch for invoking the Tessent YieldInsight GUI. This is the default.

- -NOGUI

  An optional switch for invoking the tool in command-line mode. By default, Tessent YieldInsight invokes in GUI mode.

- -LOGfile *logfile_name* -REPlace

  An optional switch and string pair that specifies the name of the file to which you want Tessent Shell to write all session information. The default is to display session information to the standard output. The version banner that indicates the tool, version, date, and platform on which the log file was produced is included at the beginning of the file.

  The optional -REPLace switch overwrites the *logfile_name* if a log file of the same name already exists.

- -DOFile *dofile_name*

  An optional switch and repeatable string pair that specifies the name of one or more dofiles to run upon invocation. As the tool is processing each dofile, the Tcl variable "tessent_user_argv0" is assigned the name of the current dofile.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -ARGuments *name=value*

  An optional switch and repeatable name/value pair that specifies variable names and values in a list named "tessent_user_arg." No spaces are permitted between the name, the equal sign, and the value. This list can later be accessed by a dofile for the purpose of controlling parameters in the dofile. For more information, refer to "Example 2" on page 89.

- -LICENSE_WAIT *minutes* | none | unlimited

  An optional switch and integer, or switch and literal, that specifies the tool's response if the license is unavailable.

  Choose one of the following options:

  - *minutes* — A positive integer that specifies the number of minutes to wait for the license.
  - none — A literal that directs the Tessent tool to exit immediately if no license is available.
  - unlimited — A literal that directs the Tessent tool to wait with no time limit for a license. This is the default.

- -HELP

  An optional switch that lists the switches for the Tessent command.

- -USAGE

  An optional switch that lists the switches for the Tessent command.

- -MANUAL

  An optional switch that opens the Tessent Documentation System.

- -VERSION

  An optional switch that displays the version of the Tessent tool you specify.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 3
Command Dictionary (A-C)

The Command Dictionary describes the application commands for Tessent Shell. For quick reference, the commands appear alphabetically with each beginning on a separate page.

The command usage line syntax conventions are common to all products documented in this manual.

Table 3-1. Conventions for Command Line Syntax

Convention	Example	Usage
UPPercase	-STatic	Required argument letters are in uppercase; in most cases, you may omit lowercase letters when entering literal arguments, and you need not enter in uppercase. Arguments are normally case insensitive.
Boldface	set_your_mode **mode1** | **mode2**	A boldface font indicates a required argument.
[ ]	exit [-force]	Square brackets enclose optional arguments. Do not enter the brackets.
Italic	dofile ***filename***	An italic font indicates a user-supplied argument.
{ }	add_your_paths {***path_name*** | **-All**} [-max_paths *number*]	Braces enclose arguments to show grouping. Do not enter the braces.
|	add_your_paths {***path_name*** | **-All**} [-max_paths *number*]	The vertical bar indicates an either/or choice between items. Do not include the bar in the command.
Underline	set_your_switch **ON** | **OFf**	An underlined item indicates either the default argument or the default value of an argument.
…	add_your_clock *off_state* ***pin_list*** …	An ellipsis follows an argument that may appear more than once. Do not include the ellipsis when entering commands.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Command Descriptions

This section describes the Tessent Shell commands alphabetically, A through C. The beginning of each command description shows the contexts and modes that support the command and provides the following information:

Context: Lists each context and sub-context that supports the command. “All contexts” means that the command is supported in all available contexts. “Unspecified” means that you do not have to set a context to use the command.

Mode: Lists each mode that supports the command. “All modes” means that the command is supported in all available modes. In some cases, a particular mode supports the command only within certain contexts, and this is noted in parentheses after the mode type.

Table 3-2. Commands A Through C

Command	Description
add_atpg_constraints	Restricts all patterns placed into the internal pattern set according to the user-defined constraints.
add_atpg_functions	Creates an ATPG function that you can then use when generating user-defined ATPG constraints.
add_bist_capture_range	Associates a named ATPG capture procedure with a subset of the BIST patterns.
add_black_boxes	Defines instances of Verilog modules or cell library models as black boxes and sets the constrained value on output or bidirectional black box pins.
add_bsdl_extraction_user_instruction	Specifies a user instruction and an optional user register to be added in the BSDL file.
add_cdp_test	Creates a new test in the CDP.
add_cell_constraints	Constrains scan cells (also non-scan cells for CX, TX, DX, and SX constraints) to a constant value.
add_cell_models	Specifies DFT library models for user-defined test points, system-generated test points, and system-generated test logic.
add_chain_masks	Masks the load, capture, and unload values on the specified scan chains during simulation.
add_clocks	Adds scan or non-scan clocks to the clock list for proper scan operation.
add_config_element	Adds a configuration element in the configuration data.
add_config_message	Adds error, warning, or information messages to configuration elements.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-2. Commands A Through C (cont.)

Command	Description
add_config_tab	Adds one configuration tree tab to the Config Data Browser window.
add_control_points	Specifies user-defined control points during test point insertion.
add_core_instances	Adds a core instance to the design by associating the core description currently in memory with the specified core instances in the design.
add_design_partition	Defines a partition of a design.
add_dft_clock_enables	Defines a pin or port of the current design or a port of a sub-module as a clock enable signal that, when activated, disables the clock gating on a clock branch.
add_dft_clock_mux	Instructs the tool to assume the presence of a clock multiplexer during pre-DFT DRC and to insert the multiplexer during DFT insertion as part of the process_dft_specification command.
add_dft_control_points	Instructs the tool to assume the presence of the DFT control logic during pre-DFT design rule checking, and to insert the controlling logic during DFT insertion as part of the process_dft_specification command.
add_dft_modal_connections	A command used to insert modal connections from specified output data source nodes or to input data destination nodes typically used to connect the EDT channel input and output pins from multiple cores to and from a limited set of input and output ports.
add_dft_signals	Adds static and dynamic DFT signals used to control various aspect of the DFT logic.
add_drc_violation	Adds a DRC violation to the list of DRC violations for a given DRC rule.
add_edt_blocks	Creates an arbitrary identifier for an EDT block and applies certain subsequent commands only to that block.
add_edt_configurations	Sets up EDT logic with two compression configurations.
add_false_paths	Defines false paths in the design. False paths are paths that cannot be activated in the design's functional (at-speed) mode of operation. You would typically determine them as a part of static timing analysis, prior to ATPG.
add_fault_sites	Automatically adds fault site definitions for library cells that do not have UDFM definitions.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-2. Commands A Through C (cont.)

Command	Description
add_faults	Adds faults to the current fault list, discards all patterns in the current test pattern set, and sets all faults to undetected.
add_gate_to_rtl_mapping	Adds rules for post-synthesis/post-layout to RTL instance name-mapping for DEF files.
add_icl_ports	Specifies top design ports that are added in the ICL file generated during ICL extraction.
add_icl_scan_interfaces	Adds the specified scan interfaces to the new ICL top module that is created during ICL extraction.
add_icl_ssn_datapaths	Specifies the SSN datapath identifiers that Tessent Shell will associate with certain ICL ports of the new ICL top module during ICL Extraction.
add_iddq_exceptions	Enables you to specify individual sites where the tool does not apply the Iddq restrictions.
add_ijtag_logical_connection	Creates a new logical connection path through the current design by specifying the source and destination of the logical path.
add_input_constraints	Constrains primary inputs to specified values.
add_layout_core_information	In hierarchical layout-aware diagnosis, adds core instance information to an existing core-level LDB.
add_lfsr_connections	Connects an external pin or an internal pin to a Linear Feedback Shift Register (LFSR).
add_lfsr_taps	Adds the tap configuration to a Linear Feedback Shift Register (LFSR) or a PRPG of type cellular automata (CA).
add_lfsrs	Adds LFSRs for use as PRPGs or MISRs.
add_lists	Adds pins to the list of pins on which to report.
add_loadboard_loopback_pairs	Creates a loopback connection from specified output ports to specified input ports.
add_misrs	Specifies the MISR structure and initial seed.
add_nofaults	Places nofault settings either on pin pathnames, pin names of specified instances, or modules.
add_nonscan_instances	Sets a user-specified is_non_scannable attribute on the specified design objects.
add_notest_points	Prevents test logic insertion in the specified regions.
add_observe_points	Specifies user-defined observe points during test point insertion.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-2. Commands A Through C (cont.)

Command	Description
add_output_masks	Ignores any fault effects that propagate to the primary output pins you name.
add_pade_logical_connection	Creates a new logical connection path through the current design by specifying the source and destination of the logical path.
add_pattern_trigger	Adds a trigger for additional equipment for measurement at a specific pattern location. This command applies only to OpalKelly adaptors using smart mode.
add_primary_inputs	Adds a primary input (PI) also known as an input pseudo_port to the specified pins or nets.
add_primary_outputs	Adds primary outputs (PO) to the specified net.
add_processors	Enables the tool to run multiple processors in parallel on multiple machines to reduce runtime.
add_read_controls	Specifies the off-state value for specified RAM read control lines.
add_register_value	Creates a static or dynamic variable through which you specify a register identifier, *value_name*, and, optionally, the state string, *value_string*.
add_rtl_to_gates_mapping	Specifies custom rules for RTL to post-synthesis and post-layout name mapping for registers, instances, and ports.
add_scan_chains	Specifies a name for a pre-existing scan chain within the design.
add_scan_groups	Specifies a group name for a set of pre-existing scan chains in a design.
add_scan_instances	Adds specified instances to the scannable instance list.
add_scan_mode	Specifies a chain scan mode, which defines one or more scan chains.
add_scan_segments	Adds one or more existing scan segment(s).
add_schematic_callout	Add an informational callout to a Tessent Visualizer schematic window.
add_schematic_connections	Displays design objects in the Tessent Visualizer schematic window and enables you to trace visually through the design.
add_schematic_objects	Displays provided objects in the specified Tessent Visualizer schematic window.
add_schematic_path	Display the specified path in one of the Tessent Visualizer schematic windows.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-2. Commands A Through C (cont.)

Command	Description
add_simdut_fault	Injects a stuck-at-0 fault or a stuck-at-1 fault into the specified signal for SimDUT simulation.
add_simulation_context	Creates a new user-defined simulation context.
add_simulation_forces	Forces one or more *gate_pin* objects to the specified value(s).
add_synchronous_clock_group	Defines a set of clocks that are synchronous (compatible) with each other.
add_tcl_variables_to_flat_model	Adds the specified Tcl variables to the flat model file.
add_tied_signals	Adds a value to floating signals or pins.
add_to_collection	Adds objects to a base collection and returns the new collection without affecting the original base collection.
add_wave_generator_pins	Displays provided pin objects in the Tessent Visualizer Wave Generator tab.
add_write_controls	Specifies the off-state value for specified RAM write control lines.
analyze_atpg_constraints	Checks the ATPG constraints you created for their satisfiability or mutual exclusivity.
analyze_bus	Analyzes the specified bus gates for contention problems.
analyze_compression	Analyzes the compression for an application in two steps.
analyze_control_signals	Identifies and optionally defines the primary inputs of control signals.
analyze_drc_violation	Generates a schematic in the schematic windows of the portion of the design involved with the specified rule violation number.
analyze_fault	Performs an analysis to identify why a fault is not detected and optionally displays the relevant circuitry in Tessent Visualizer.
analyze_graybox	Identifies the instances and nets to be included in the graybox netlist by automatically setting their in_graybox attributes to true. Subsequently, a “write_design -graybox” command writes out the graybox netlist for the marked instances and nets.
analyze_icl_suspect	Context: patterns -silicon_insight -ijtag
analyze_layout_hierarchy	Analyzes the layout LEF/DEF hierarchy and stores the analysis results in a layout hierarchy database.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-2. Commands A Through C (cont.)

Command	Description
analyze_restrictions	Performs an analysis to automatically determine sources of the problem from a failed ATPG run.
analyze_scan_chains	Distributes scan elements into new scan chains.
analyze_simulation_mismatches	Analyzes simulation mismatches that occur between test patterns and a parallel testbench simulation performed by an external timing simulator.
analyze_test_points	Performs test point analysis and generates a list of test points for the tool to insert.
analyze_wrapper_cells	Identifies shared and dedicated wrapper cells for the primary I/O ports.
analyze_xbounding	Performs X-bounding analysis.
annotate_diagnosis	Adds DFM and RCD data, if available, to existing diagnosis reports.
append_to_collection	Adds one or more objects to a collection referenced by the specified ***var_name***.
apply_specification_defaults	Automatically applies all defaults specified in the DefaultsSpecification/DftSpecification and DefaultsSpecification/PatternsSpecification wrappers to wrappers that were created manually.
assert_icl_sib	Context: patterns -silicon_insight -ijtag
catch_output	Runs the specified tool command line, and may be used to prevent command errors from aborting the enclosing dofile or Tcl proc.
check_design_rules	Transitions the tool from setup mode to analysis mode.
check_synthesis	Checks the status of synthesis that was previously launched by the run_synthesis command.
check_testbench_simulations	Checks the status of simulations that were previously launched by the run_testbench_simulations command.
close_layout	Closes the LDB. Any subsequent diagnosis you perform is non-layout aware.
close_pattern_set	Finalizes and closes the currently open pattern set.
close_tsdb	Makes the contents of a previously opened TSDB directory invisible to the tool.
close_visualizer	Saves display data and closes Tessent Visualizer.
collect_cdp_test_failure_logs	Saves failure logs for a CDP test to a directory.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-2. Commands A Through C (cont.)

Command	Description
compare_collections	Compares the contents of two collections.
compress_layout	Compresses a LDB.
compress_patterns	Compresses patterns in the current test pattern set.
convert_design_path_format	Converts a list of design paths to the specified dot or slash separator.
copy_collection	Duplicates the contents of an existing collection, resulting in a new collection. The base collection remains unchanged.
copy_module	Creates an exact copy of a design module and assigns it a new name so that it can then be used as part of a create_instance and replace_instances.
copy_simulation_context	Copies the simulation values and forces from one simulation context to another.
create_bisr_segment_order_file	Creates the BisrSegmentOrderSpecification configuration data wrapper and extracts the list and ordering of BISR chain segments.
create_capture_procedures	Creates named capture procedures based on clock sequences either derived from patterns or specified at the command line.
create_connections	Connects pin, net, or port objects.
create_custom_bisr_register	Creates a customized BISR register that is manually instantiated by the user in the functional design and is used to control functional logic.
create_dft_specification	Creates the DftSpecification(*design_name*,*id*) configuration wrapper and returns the newly created wrapper object so that it can be stored in a variable and used to customize the created specification.
create_diagnosis_patterns	Creates targeted patterns for logic failure or scan chain failure iterative diagnosis.
create_feature_statistics	Calculates the root cause deconvolution (RCD) constants from the LDB for RCD diagnosis and analysis.
create_flat_model	Creates a primitive gate simulation representation of the design. The command also flattens the hierarchy of the design so that the design can be viewed in the Tessent Visualizer Flat Schematic window.
create_icl_flush_pattern	Context: patterns -silicon_insight -ijtag
create_icl_setup_pattern	Context: patterns -silicon_insight -ijtag

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-2. Commands A Through C (cont.)

Command	Description
create_icl_verification_patterns	Specifies the creation of a structural pattern to verify the proper operation of the IJTAG network.
create_initialization_patterns	Creates RAM initialization patterns and places them in the internal pattern set.
create_instance	Instantiates an instance of a module mod_spec inside a design module in the current design.
create_layout	Generates a diagnosis tool-compatible Layout Database directory (LDB) from LEF/DEF input files.
create_memory_repair_flow	Incorporates a Repair Flow Custom Test into the patterns specification.
create_module	Creates a new design module with no ports.
create_net	Creates a net inside an instance of a design module in the current design.
create_open_socket_failure	Creates emulated open-socket failure files to use for validating reverse mapping top-level failures to the core in Tessent Diagnosis.
create_patterns	Performs automatic high-speed test generation and pattern compression.
create_patterns_specification	Generates a pattern specification for the specified usage. The usage is either signoff or manufacturing. The tool supports IJTAG and MemoryBIST patterns.
create_pin	Creates a pin on an instance of a design module.
create_port	Creates a port on the specified design module.
create_reversible_patterns	Generates reversible test patterns.
create_scan_chain_family	Creates a scan_chain_family object, which controls how one or more scan chains are allocated/connected during distribution.
create_scan_segment	Creates a spec scan segment that enables you to force sequences of scan elements to be stitched together within scan chains.
create_silicon_insight_setup_specification	Generates a default Tessent SiliconInsight setup specification.
create_ssn_continuity_patterns	Creates an SSN continuity pattern to test the internal path of the ScanHost from input to output.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_atpg_constraints

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Restricts all patterns placed into the internal pattern set according to the user-defined constraints.

Usage

add_atpg_constraints {**0** | **1** | **Z**} {***pin_pathname*** | ***gate_id#*** | {***function_name*** |
{**-Cell** ***cell_name*** {***pin_name***…}}}… [-Instance {*object_expression*…}]
[-MODUle {*module_name* …}]}… [-DYnamic | -STatic]
[-NOCapclock_check | -Load_only]

Description

When a simulated pattern is rejected, a message is issued that indicates the number of rejected patterns and the first gate at which the failure occurred. You can control the severity of the violation with the set_contention_check command. If you set the checking severity to Error, the simulation is terminated if it rejects a pattern due to a user-defined constraint. You can analyze the simulation data up to the termination point by using the report_gates command with the Error_pattern option.

User-defined constraints are used when using Tessent FastScan or Tessent TestKompress to generate test patterns with deterministic test generation methods. When test patterns are generated randomly using Tessent FastScan or Tessent TestKompress, the use of the user-defined ATPG constraints is not ensured; however; non-conforming random patterns are rejected.

If you change an ATPG constraint for a single internal set of patterns, pattern compression continues using the new constraints, which can cause the good patterns to be rejected. Therefore, you should remove all ATPG constraints before compressing the pattern set.

Note

If you constrain a pin by directly creating an ATPG constraint to the *pin_pathname*, and then create another constraint that indirectly creates a different constraint, the constraint that directly specified the *pin_pathname* (overriding the global ATPG cell constraint) is used.

The -Dynamic switch lets you change the ATPG constraints any time during the ATPG process, affecting only the fault simulation and capture that occurs after the constraint changes. Any subsequently simulated patterns that fail to meet the current constraints are rejected during fault simulation.

Dynamic ATPG constraints do not affect DRC because of their temporary nature. Static ATPG constraints are unchangeable in analysis mode, ensuring that DRC must be repeated if they are changed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- **0 |1 |Z**

  A literal that restricts the named object to a low state, high state, or high impedance state, respectively. You must choose one of the three literals to indicate the state value to which you want the tool to constrain the specified object.

  The following lists the four objects on which you can place the constraint. You can use any number of the four argument choices in any order.

- ***pin_pathname***

  A repeatable string specifying the pathname of a top-level pin or a library model pin on which you are placing the constraint. Pathnames of pins on intermediate hierarchy modules are not supported.

- ***gate_id#***

  A repeatable integer specifying the gate identification number of the gate you want to constrain.

- ***function_name***

  A repeatable string specifying the name of a function you created with the add_atpg_functions command. If you place a constraint on an ATPG function that you generated with the add_atpg_functions -Cell command, all cells affected by that ATPG function are also constrained. You can delete all these constraints using the *function_name* argument with the delete_atpg_constraints command.

- **-Cell *cell_name*{*pin_name*}**

  A repeatable switch and string pair specifying the name of a DFT library cell and name of a pin on that cell. You can repeat the *pin_name* argument if there are multiple pins or nets on a cell that you need to constrain. If you use the -Cell option, an ATPG constraint is placed on every occurrence of that cell within the design.

- -Instance *object_expression*

  An optional switch and repeatable string pair that places ATPG constraints inside the specified list of instances. You can use regular expressions, which may include any number of embedded asterisk (*) and question mark (?) wildcard characters. You can only use this switch when using the -Cell switch or when a function name is specified.

- -MODUle *module_name*

  An optional switch and repeatable string pair that places the ATPG constraints inside all instances of the specified module.

- -DYnamic

  An optional switch that satisfies only the ATPG constraints during the ATPG process and not during DRC. You can change these constraints during the ATPG process; therefore, DRC does not check these constraints. This is the default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -STatic

  An optional switch that satisfies the ATPG constraints you are defining during all its processes. You can only add or delete static ATPG constraints when you are in Setup mode, ensuring the static ATPG constraints are used for all ATPG analyses during design rules checking. DRC checks for any violations of ATPG constraints during the simulation of the test procedures (rule E12).

- -NOCapclock_check

  An optional switch that suppresses checking of specified ATPG constraints after the capture clock. By default, ATPG constraints are checked at the same time as bus contention. If contention checking is performed after the capture clock, ATPG constraints are also checked after the capture clock. You cannot use this switch and -Load_only at the same time.

  In some situations, you may have some ATPG constraints that do not need to be checked after the capture clock; although, you may want other constraints and bus contention to be checked after the capture clock. In this case, you can use the -Nocapclock_check switch on certain constraints to suppress checking of those constraints after the capture clock.

- -Load_only

  An optional switch specifying the Atpg constraint only applies to the first cycle after shift. When using Tessent FastScan or Tessent TestKompress, the values loaded into the scan chain and the forced PI values satisfy the constraint; however, once the leading edge of the first clock occurs, the constraint no longer applies (as if it were temporarily deleted until the next load, then re-added). Constraints added using the -Load_only option are satisfied by each scan load of a RAM sequential pattern and by each scan load of a clock sequential multiple load pattern. A clock sequential multiple load pattern occurs when you issue the set_pattern_type command with the -Multiple_load switch on, and the -Sequential switch set to 2 or more. Note that you cannot use the -Load_only and -NOCapclock_check switches at the same time.

Examples

Example 1

The following example creates a user-defined ATPG function and then uses it when creating ATPG pin constraints:

```
add_atpg_functions and_b_in and /i$144/q /i$141/q /i$142/q
add_atpg_constraints 0 /i$135/q
add_atpg_constraints 1 and_b_in
```

Example 2

The following example creates a user-defined ATPG function and then uses it when creating ATPG constraints. The ATPG constraints are added to the instances inside core1 only.

```
add_atpg_functions mux_1hotsel SELECT1 -cell imux3sux2 S2 S1 S0
add_atpg_constraints 1 mux_1hotsel -instance core1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

The following example adds ATPG constraints to all occurrences of the specified cell inside the instances with pathnames that begin with “/core1/u19/reg”.

```
add_atpg_constraints 1 -cell imux3sux2 S1 -instance /core1/u19/reg*
```

Example 4

The following example shows how to use wildcards in the second level of hierarchy.

```
add_atpg_constraints 1 -cell imux3sux2 S1 -instance /core1/*/reg*
```

Related Topics

add_atpg_functions

delete_atpg_constraints

report_atpg_constraints

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_atpg_functions

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Creates an ATPG function that you can then use when generating user-defined ATPG constraints.

Usage

add_atpg_functions ***function_name*** ***type*** {***pin_pathname*** | ***gate_id#*** | ***existing_function_name*** | {**-Cell** ***cell_name*** {***pin_name*** | **-Function** ***type pin_name***...}...}}...

Description

You can specify any combination of pin pathnames, gate identification numbers, and previously user-defined functions. You can precede any object with the tilde (~) character to indicate an inverted input with respect to the function. If you specify an input pin pathname, the tool automatically converts it to the output pin of the gate that drives that input pin.

Arguments

- ***function_name***

  A required string that specifies the name of the ATPG function you are creating. You can use this *function_name* as an argument to the add_atpg_constraints command.

- ***type***

  A required argument specifying the operation that the function performs on the selected objects. The choices for the *type* argument, from which you can select only one, are as follows:

  And — The output of the function is the same as for a standard AND gate.

  Or — The output of the function is the same as for a standard OR gate.

  Equiv — The output of the function is a high state (1) if all its inputs are at a low state (0), or if all its inputs are at a high state (1). So, the function's output is a low state if there is at least one input at a low state and at least one input at a high state.

  Select — The output of the function is a high state (1) if all its inputs are at a low state (0) or if one input is at a high state and the other inputs are at a low state. So, if there are at least two inputs at a high state, the function's output is at a low state.

  SELECT1 — The output of the function is a high state (1) if one input is at a high state and the other inputs are at a low state (0). So, the function's output is a low state if there are at least two inputs at a high state or all inputs are at a low state.

  The following lists the objects on which the function operates. You can use any number of the argument choices in any order.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ***pin_pathname***

  A repeatable string that specifies the pathname to the pin on which you are placing the function. If you specify an input pin name, it is automatically replaced with the output pin of the gate that drives that input pin.

- ***gate_ID#***

  A repeatable integer that specifies the gate identification number.

- ***existing_function_name***

  A repeatable string that specifies the name of another function you created with the add_atpg_functions command. The *existing_function_name* argument cannot be the same as any pin name in the design. This string cannot be used with the -Cell option.

- **-Cell *cell_name*** {***pin_name*** | **-Function *type pin_name***}

  A repeatable switch with a corresponding pair of strings that specify the name of a DFT library cell or pin on that cell. You can repeat the *pin_name* argument if you need to constrain multiple pins or nets on a cell.

  The -Function switch is a repeatable switch with corresponding strings that specify the function type and a list of specific pins on cell *cell_name* that are arguments to the function. You can repeat the *pin_name* argument if you need to constrain multiple pins. This switch enables you to define a maximum of two layers of functions with the -Cell switch. Normally, you can define two functions, f1 and f2, and then add a third function, f3, that refers to f1 and f2. However, this is not permitted when you use the -Cell switch.

  For information on how this switch is used, see Example 2.

  If you use the -Cell option, an ATPG function is placed on every occurrence of that cell within the design.

Examples

Example 1

The following example creates an ATPG function and then uses it in an add_atpg_constraints command:

```
add_atpg_functions and_b_in and /i$144/q /i$141/q /i$142/q
add_atpg_constraints 1 and_b_in
```

Example 2

The following example generates multiple levels of ATPG functions when using the -Cell switch.

For every instantiation of the OR4 cell, the following function is implemented:

```
[ ( A AND B ) OR ( C OR D ) ]
```

where A, B, C, and D are inputs of the OR4 cell. Normally, you can define two functions, f1 and f2, then add a third function, f3, that refers to f1 and f2. However, when you use the -Cell switch

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

with the add_atpg_functions command, this is not possible. You can use the -Function switch to define up to two levels of functions using the -Cell switch. The syntax is as follows:

```
add_atpg_functions top_function or -cell OR4 -function and A B -function or C D
```

This implemented the previously mentioned function, which can be confirmed with the report_atpg_functions command.

Example 3

The following example assigns the two Q outputs of scan cells /u11 and /u12 to always have the same assigned state after loading.

```
add_atpg_functions CELL_INCLUSIVE equiv /u11/Q /u12/Q
add_atpg_constraints 1 CELL_INCLUSIVE
```

Example 4

The following example causes one input to always be the opposite from the other. The tilde (~) character is the negation symbol.

```
add_atpg_functions CELL_EXCLUSIVE equiv ~ /u71/Q /u72/Q
add_atpg_constraints 1 CELL_EXCLUSIVE
```

Related Topics

add_atpg_constraints

delete_atpg_functions

report_atpg_functions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_bist_capture_range

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Associates a named ATPG capture procedure with a subset of the BIST patterns.

Usage

add_bist_capture_range ***atpg_capture_proc_name start_value stop_value***

Description

Hybrid TK/LBIST Flow Usage

The add_bist_capture_range command is added once per NCP. Once all the NCPs are applied, the tool restarts from the first NCP.

Additionally, the command is automatically added to the logicBIST fault simulation dofile written out during IP generation. For usage information, refer to "LogicBIST Fault Simulation and Pattern Creation" in the *Hybrid TK/LBIST Flow User's Manual*.

Arguments

- ***atpg_capture_proc_name***

  A required string that specifies the name of the ATPG capture procedure.

- ***start_value***

  A required argument that specifies the beginning of a contiguous range for a group of capture procedures. The tool typically generates the start and stop values in the BIST controller synthesis phase, and you should not change the values.

- ***stop_value***

  A required argument that specifies the end of a contiguous range for a group of capture procedures. The tool typically generates the start and stop values in the BIST controller synthesis phase, and you should not change the values.

Examples

This example shows the association of NCPs with capture ranges corresponding to the specification - clk1_twice 34 percent, clk2_twice 33 percent, and clk3_twice 33 percent:

```
add_bist_capture_range clk1_twice 0 87
add_bist_capture_range clk2_twice 88 171
add_bist_capture_range clk3_twice 172 255
```

Related Topics

delete_bist_capture_ranges

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_bist_capture_ranges

set_lbist_controller_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_black_boxes

Context: all contexts

Mode: setup

Defines instances of Verilog modules or cell library models as black boxes and sets the constrained value on output or bidirectional black box pins.

Usage

add_black_boxes {{{-**Instances** ***ins_pathnames*** | -**Modules** ***module_names***}
[0 | 1 | X | Z]} [-Pin *pinname* {0 | 1 | X | Z}]…} | {-**Auto** [0 | 1 | X | Z]}
[-FAUlt_boundary | -NOFAUlt_boundary | -NO_Boundary]

Description

A black box is an instance that has no defined model or that you want to treat as though it has no model.

You can blackbox a single instance, or every instance of a particular module or cell library model that you do not want included in the netlist used by the tool. You can also use this command to automatically blackbox instances of nonexistent modules or cell library models.

When reading in a Verilog netlist, a warning message is issued when an instance is found that references a model (Verilog module or cell library model) that is not in the netlist or cell library. A referenced model for which a module or cell library model of the same name is not found in the netlist or cell library is considered to be undefined. You must blackbox instances of undefined models; otherwise, an error message is issued when you attempt to perform certain tasks such as leave Setup mode or enter a create_flat_model command. The message lists the names of referenced models that were not found in the netlist or cell library. You can also list the names using “report_black_boxes -undefined”.

Use add_black_boxes with the -Auto switch to automatically blackbox all instances of models that are considered to be undefined. By default, instances that are automatically blackboxed drive Xs on their outputs. Faults that propagate to the black box inputs are classified as ATPG_untestable (AU). You can use the add_black_boxes command to change the output values.

Issuing multiple add_black_boxes commands in succession, the first with the -Module switch and subsequent ones with -Instance, globally blackboxes all instances of a particular module or cell library model while selectively blackboxing particular instances of the same module or cell library model with slightly different pin values.

Tip

For optimal performance, issue all add_black_boxes commands after reading the design and library but before issuing any other commands.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In dft -rtl context only, the tool automatically blackboxes modules or instances detected as Synopsys DesignWare modules. The actual blackbox operation is done when the tool transitions from setup to analysis mode using the set_system_mode command. When the tool exits analysis mode, the tool automatically resets the black box definitions.

Arguments

- **-Instances *ins_pathnames***

  A switch and string pair that specifies to blackbox the objects specified by the ***ins_pathnames*** argument. The ***ins_pathnames*** argument can be a Tcl list of one or more instances or a collection of one or more instances.

  Note

  Blackboxing implemented using the -Instances switch always overrides blackboxing implemented with the -Modules switch.

- **-Modules *module_names***

  A switch and string pair that specifies to blackbox every instance of the Verilog module or cell library model specified by the ***module_names*** argument. The ***module_names*** argument can be a Tcl list of one or more modules or a collection of one or more modules.

- **-Auto**

  A switch that blackboxes any netlist instance that references a Verilog module or cell library model that is not in the netlist or in the cell library.

  Note

  To properly blackbox instances for Tessent FastScan MacroTest, use the -Instance or -Module switch. The -Auto switch is not intended for use with Tessent FastScan MacroTest. To ensure success, you must also define at least the module (or entity or cell library model) pins, including names, directions, and width of vectors.

  Note

  To prevent unintentional black-boxing of new undefined modules in future executions of your scripts, it is recommended to not use the -auto switch in your dofile.

- 0 | 1 | X | Z

  An optional literal that specifies pin tie values. When used with the -Instance or -Module switch, it specifies the tie value for any pin not explicitly defined by the -Pin switch. When used with -Auto, it specifies the tie value for every pin. If you specify no value for this option, then the set_tied_signals command's value is the default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Pin *pinname* {0 | 1 | X | Z}

  An optional, repeatable switch, string, and literal that specifies the tie value of a particular pin. Use this switch to explicitly define the tie value for a pin, overriding for that pin the global tie value in effect for the -Instance or -Module switch.

- -FAUlt_boundary

  A switch that keeps pin pathnames at the boundaries of all blackboxed instances and enables boundary pins to become fault sites. This is the default for `celldefine modules and library models.

- -NOFAUlt_boundary

  A switch that keeps pin pathnames at the boundaries of all blackboxed instances, but does not permit boundary pins to become fault sites. This is the default for blackboxed instances that are not `celldefine modules or library models.

- -NO_Boundary

  A switch that does not keep pin pathnames at the boundaries of all blackboxed instances and does not permit boundary pins to become fault sites.

Note

Do not include this switch when you use the add_black_boxes command to blackbox a macro for Tessent FastScan MacroTest. The Tessent FastScan macrotest command must be able to find pin pathnames at the macro boundary.

Examples

Example 1

The following example creates a black box for a module named “core,” with each output driven by a tie0 logic gate. The example then overrides, for the single instance “/core1” of the module “core,” the output value of pin1, so it is driven by a tie1 logic gate instead of a tie0. All other instances of the “core” module still have outputs driven by tie0 logic.

```
add_black_boxes -modules core 0
add_black_boxes -instances /core1 -pin pin1 1
```

Example 2

The following example shows the tool’s warning message for instances of undefined models. The example then creates black boxes for these instances.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Warning: Undefined modules were found.
// Before using "set_system_mode" or "create_flat_model", you must either
// define the missing modules using "read_verilog" and/or
// "read_cell_library", or use the following command to treat them as
// black boxes:
// add_black_boxes -modules { \
//   and02 \
//   ao2 \
// }
// You can also use "add_black_boxes -auto" to black box all undefined
// modules but it is recommended that you do not add this command to your
// dofile. Doing so may unintentionally black-box new undefined modules
// in future runs.
```

add_black_boxes -auto

Example 3

The following example shows two commands that keep the pin pathnames at the boundary of each instance of the blackboxed module named “macro1.” Either command enables Tessent FastScan MacroTest to function properly:

add_black_boxes -modules macro1 -fault_boundary

add_black_boxes -modules macro1 -nofault_boundary

Example 4

The following example creates blackboxes for the collection of modules returned by the get_modules command. The get_modules command returns only those modules that have been marked by the user with the user-defined attribute mark_black_box.

add_black_boxes -modules [get_modules -filter mark_black_box]

Example 5

Similar to Example 4, this example operates on instances that have a user-defined attribute mark_black_box.

add_black_boxes -instances [get_instances -filter mark_black_box]

Example 6

The following example creates blackboxes by assigning the collection of modules returned by the get_modules command to a variable. The add_black_box command then operates on the collection contained in the to_black_box variable.

set to_black_box [get_modules -filter mark_black_box]
add_black_boxes -modules $to_black_box

Related Topics

delete_black_boxes

report_black_boxes

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_bsdl_extraction_user_instruction

Context: dft (with no subcontext)

Mode: setup

Specifies a user instruction and an optional user register to be added in the BSDL file.

Usage

add_bsdl_extraction_user_instruction ***name*** ...
-instruction_codes *instruction_codes*
[-bsdl_visibility {public | private}]
[-user_register {*user_register* | *user_register*[*int*]} [-capture_value *capture_value*]]

Description

The user instruction ***name*** is listed in the INSTRUCTION_OPCODE section of the BSDL file. If the ***name*** is specified as being private, it is also listed in the INSTRUCTION_PRIVATE section.

If a user register associated with ***name*** is specified, it is listed in the REGISTER_ACCESS section of the BSDL file. User registers specified with this command must be IEEE1149.1 standard registers such as BOUNDARY, BYPASS, or DEVICE_id, or be defined with a register name with the register size in brackets (user_register[int]).

The opcodes for the user instruction must be defined directly using the -instruction_codes option.

Arguments

- ***name***

  A repeatable identifier used to specify the user instruction name. These user instruction names are listed in the INSTRUCTION_OPCODE section of the BSDL file and must be valid BSDL names.

- **-instruction_codes *instruction_codes***

  A required switch and binary list pair that specifies the binary opcodes associated with the ***name*** user instruction. The ***instruction_codes*** option is a sized binary list consisting of 1s and 0s; for example “[list 2’b11 2’b00]”. It must match the size of the instruction register.

- -bsdl_visibility public | private

  An optional switch and value pair that specifies whether the instruction ***name*** is to be used only by the manufacturer or to be available publicly. The default value is public. If set to private, the ***name*** is listed in the INSTRUCTION_PRIVATE section of the BSDL file.

- -user_register *user_register* | *user_register*[*int*]

  An optional switch and string pair that specifies a user register associated with the user instruction ***name***. The value *user_register* specifies the name of the user register and must be a valid BSDL name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The specified *user_register* is listed in the REGISTER_ACCESS section of the BSDL file. if *user_register* is not specified, then bsdl_visibility must be set to private.

- -capture_value *capture_value*

  An optional switch and string that specifies the capture value. The value is a sized binary string consisting of 1s, 0s, and Xs; for example “8’b0000X1X1”. It must match the size of the register. You can only use this switch when using the -user_register switch.

Related Topics

delete_bsdl_extraction_user_instruction

report_bsdl_extraction_user_instructions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_cdp_test

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Creates a new test in the CDP.

Usage

add_cdp_test ***test_name*** [-pattern *pattern_file*] [-support_pattern_files *support_pattern_file* ...]

Description

Creates a new test in the CDP. The test is based on the specified pattern file.

The newly created test enables you to perform two test operations:

- Run the pattern file to determine the pass/fail status.
- Run the pattern file and collect failing data.

Arguments

- ***test_name***

  A required string that specifies the name of the test.

- -pattern *pattern_file*

  An optional switch and string pair that specifies the directory path for a standalone SVF or STIL test pattern file that is associated with the test.

- -support_pattern_files *support_pattern_file* ...

  An optional switch and string pair that specifies one or more pattern files that are associated with the pattern file specified by the -pattern switch.

Examples

The following example adds a test pattern file named *pat.svf* to the CDP and creates a test named *test1* for the *pat.svf* pattern file.

```
add_cdp_test test1 -pattern ./patterns/pat.svf
```

Refer to "Running Non-Tessent IJTAG Instruments" in *Tessent SiliconInsight User's Manual for Tessent Shell* for a sample usage scenario.

Related Topics

delete_cdp_test

execute_cdp_test

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_cell_constraints

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Constrains scan cells (also non-scan cells for CX, TX, DX, and SX constraints) to a constant value.

Usage

add_cell_constraints {**-Drc C6**} | {{{***chain_name cell_position***} | ***pin_pathname*** |
{**-CHain** ***chain_name***} | {**-CLock** ***clock_name*** [-Type {Latch | Flop}]}
-Instances *instance_name ...*} {**C0** | **C1** | **CX** | **OX** | **XX** | **TX** | **DX** | **SX** | **C0DX** | **C1DX**}
[-All_patterns | -Scan_patterns]}

Description

For all contexts, you can identify a particular scan cell either by specifying a scan chain name along with the cell's position in the scan chain, or by specifying an output pin pathname that connects to a scan memory element. The constraint value that you specify is placed at either the output pin or the scan cell MASTER. The "patterns -scan" context additionally provides the -Chain and -Clock switches for identifying groups of scan cells to receive a particular constraint.

The rules checker audits the correctness of the data that defines the constrained scan cells immediately after scan cell identification. The checker identifies all invalid scan cell constraints and an error condition occurs.

In the case of scan cells with improper controllability or observability, rather than rejecting these circuits, you can constrain (or mask) their controllability or observability.

Note

Adding constraints can lower test coverage.

Arguments

- **-Drc C6 ("patterns -scan" context)**

  A switch and literal pair that specifies to add a DX cell constraint to each latch and flip-flop that results in a C6 violation during DRC. Because this switch could have a big impact on the test coverage, we strongly recommend that you use the "set_simulation_options -C6_mask_races on" command to resolve C6 violations that occur when either the data port is faster than the clock port, or when both ports have almost the same delay and it is indeterminate as to which port is faster. For more description of these issues, see the -C6_mask_races argument description for the set_simulation_options command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ***pin_pathname***

  A string that specifies the name of one of the following types of pins:

  - An output pin of the scan cell.
  - An output pin directly connected through only buffers and inverters to the output of a scan memory element. The scan memory element is set to the value that you specify such that the pin is at the constrained value.
  - An output pin of a non-scan cell if you are applying a CX, TX, DX, or SX constraint.

  Except as noted above, the pin pathname must resolve to a scan memory element. Buffers and inverters may reside between the pin and the memory element.

  You can also use an introspection command to generate a Tcl collection of pins. For example:

  ```
  add_cell_constraints [get_gate_pins GO_ID_REG_reg_*/q] OX
  ```

- ***chain_name cell_position***

  A string pair that specifies the name of the scan chain and the position of the cell in the scan chain. The scan chain must be a currently-defined scan chain and the position must be an integer where 0 is the scan cell closest to the scan-out pin. You can determine the position of a cell within a scan chain by using the report_scan_cells command.

  The MASTER memory element of the specified scan cell is set to the value that you specify; there is no inversion. However, the tool may invert the output pin of the scan cell if there is anything between it and the MASTER memory element if inversion exists between the MASTER and the scan output pin of the scan cell only.

- **-CHain *chain_name* ("patterns -scan" context only)**

  A switch and string pair that places cell constraints on all cells in the specified scan chain, except any that are already constrained; this switch does not override a pre-existing cell constraint on a cell. This switch is valid in non-Setup system modes only.

- **-CLock *clock_name* ("patterns -scan" context only)**

  A switch and string pair that adds cell constraints to all latches and flip-flops controlled by the clock, clock_name. If the cell that includes the latch or flip-flop is already constrained, this switch does not override the pre-existing constraint. This switch is valid in non-Setup system modes only.

  Note

  This switch only supports constraint types OX, DX and SX.

- -Type Latch | Flop **("patterns -scan" context only)**

  An optional switch and literal pair that limits the -Clock switch's action to adding cell constraints on just the latches or just the flip-flops controlled by the clock, clock_name. This switch is valid in non-Setup system modes only.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Instances *instance_name* ...

  An optional switch and string pair that specifies one or more instance names upon which to add cell constraints. This switch supports all types of constraints for scan cells inside the specified instance.

  This switch also supports all types of constraints for non-scan cells and transparent latches in the scan path inside the specified instance, except for the following cases:

  - If the state element is learned as TIE-0 or TIE-1, you cannot add any cell constraint.
  - If the DRC learns the state element with loading value 0 or 1 when switching to capture, you cannot add cell constraints C1 and C1DX to the state element with loading value 0, or add constraints C0 and C0DX to the state element with loading value 1.
  - If the DRC analysis classifies the state element as TIE-X during capture, you cannot add C0, C1, C0DX, and C1DX cell constraints to this state element.
  - If the DRC analysis classifies the state element as a transparent latch during capture, you cannot add C0, C1, C0DX, or C1DX cell constraints to this state element.
  - You cannot add OX cell constraints to non-scan cells or transparent latches in the scan path.

- -Scan_patterns **("patterns -scan" context only)**

  An optional switch that applies the cell constraint only when creating scan test patterns.

- -All_patterns **("patterns -scan" context only)**

  An optional switch that applies the cell constraint when creating all patterns (scan and chain test patterns). Note that this switch is not valid with the -Drc switch in the same command line. This is the default.

  Note

  "dft -edt" context — if you specify many binary constraints, the tool may not be able to create chain test patterns that satisfy all the constraints.

- Constraint Values

  C0

  A literal that constrains the scan cell to load value 0 only.

  C1

  A literal that constrains the scan cell to load value 1 only.

  CX

  A literal that specifies to simulate the loaded scan cell or non-scan value as unknown (uncontrollable).

  OX

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

A literal that specifies to simulate the unloaded scan cell value as unknown (unobservable).

XX

A literal that constrains the scan cell to be both uncontrollable and unobservable.

TX

("patterns -scan" context only) A literal that constrains the scan cell or non-scan cell to TIE-X during test generation and simulation. Additionally, the logic value X is loaded into and unloaded from the scan cell.

DX

("patterns -scan" context only) A literal that specifies for the loaded value to remain until the cell (scan or non-scan) is disturbed by any on clock. When the cell is disturbed, the output value of the cell becomes X for subsequent cycles until the next scan load initializes it to a new value. The cell unloads an X if disturbed, but if not disturbed unloads the loaded value. The tool does not use a scan cell constrained to DX as an observation point.

If you use "add_cell_constraints -drc c6", the tool applies a DX constraint automatically to each latch and flip-flop that results in a C6 violation during DRC. See the description of the "-Drc C6" argument for more information.

SX

("patterns -scan" context only) Similar to DX except that when the cell is disturbed by any on clock, the output value of the cell becomes X only if the captured value is different from the loaded (or initialization) value. The tool does not use a scan cell constrained to SX as an observation point.

Note

If "set_drc_handling d8 warning" is in effect for DRC, the tool automatically applies an SX cell constraint to each D8 failing master cell. This prevents D8-related simulation mismatches when you verify the patterns in a timing based simulator, but may reduce coverage. If you use "set_drc_handling d8 note" for DRC, the tool does not apply SX cell constraints for D8 violations, which ensures test coverage is not affected. But, you should use note handling only for cell architectures you know do not produce mismatches.

C0DX

("patterns -scan" context only) A literal that constrains the scan cell to both load value 0 only and to keep the loaded value until the cell is disturbed by any on clock (C0 and DX). After the clock disturbance, the unload data is X (masked).

C1DX

("patterns -scan" context only) A literal that constrains the scan cell to both load value 1 only and to keep the loaded value until the cell is disturbed by any on clock (C1 and DX). After the clock disturbance, the unload data is X (masked).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-3 shows how cell constraints affect the cell during load-capture-unload. The table shows the unloaded (expected) value of the cell given three possible scenarios during capture:

- **Cell Not Disturbed** — All of the cell's clock/set/reset ports are off.
- **Cell Disturbed (same value captured)** — At least one of the cell's clock/set/reset ports are not off (either on or X), and the new stable capture value is the same as the old value in the cell. Because the tools perform a timeless simulation, the cell's new capture value may not be predictable for this scenario (for example, if a timing exception path drives this cell).
- **Cell Disturbed (different value captured)** — At least one of the cell's clock/set/reset ports is not off (either on or X), and the new stable capture value is different from the old value in the cell.

Note that the table uses ‘a’ and ‘b’ to represent different Boolean values (for example, if a=1, then b=0 or X).

Table 3-3. Example Cell Value Changes With Different Constraints for Scan Patterns

Cycle:	**Load**	**Capture**			**Unload**
Constraint	**Loaded Value (initialized value)**	**Cell Not Disturbed**	**Cell Disturbed (same value captured)**	**Cell Disturbed (diff. value captured)**	**Unloaded Value**
No constraint	a	a	a	b	same as capture value
C0	0	0	0	b	same as capture value
C1	1	1	1	b	same as capture value
CX	X	X	X	b	same as capture value
OX	a	a	a	b	X
XX	X	X	a	b	X
TX	X	X	X	X	X
DX	a	a	X	X	same as capture value
SX	a	a	a	X	same as capture value
C0DX	0	0	X	X	same as capture value
C1DX	1	1	X	X	same as capture value

To help determine the appropriate constraint type for your situation, refer to Table 3-4.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-4. Constraint Type Recommendations

Loading value reliable?	Unloading value reliable?	Capture value reliable?	Constraint
No	No	No	TX
No	No	Yes	XX
No	Yes	Yes	CX
Yes	No	Yes	OX
Yes	N/A	Yes if no clock is pulsed	DX
Yes	N/A	Yes if same value is captured	SX

Table 3-5 shows, for chain patterns, the different load and unload values of a scan cell with different constraints applied. Notice for chain patterns that CX, XX and TX are equivalent, and DX and SX are the same as no constraint.

Table 3-5. Example Scan Cell Value Changes With Different Constraints for Chain Patterns

Constraint	Loaded Value	Unloaded Value
No constraint	a	a
C0	0	0
C1	1	1
CX	X	X
OX	a	X
XX	X	X
TX	X	X
DX	a	a
SX	a	a
C0DX	0	0
C1DX	1	1

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example applies one constraint for use during the generation of scan patterns and one constraint for use during the generation of all patterns. Then it displays a list of all the constrained scan cells:

```
add_cell_constraints chain1 5 c0
add_cell_constraints /reg_d2_/Q OX -all_patterns
report_cell_constraints
```

```
// constraint chain   cell   cell_instance constraint    pattern scope
//    value   name  position      name     properties
// ---------- ----- -------- ------------- ---------- -------------------
//     CO     chain1    5                  (Dynamic)  Scan Patterns only
//     OX     chain4    7    /reg_d2_/Q    (Dynamic)  Chain + Scan Patt's
```

Example 2

The following example constrains all the remaining scan cells in the chain1 scan chain to a constant one:

```
add_cell_constraints -chain chain1 c1
```

Example 3

The following example solves C6 rule violations resulting from DRC:

```
add_cell_constraints -drc c6
```

Example 4

The following example adds an OX constraint to all flip-flops controlled by clock /clk1:

```
add_cell_constraints -clock /clk1 -type flop OX
```

Related Topics

delete_cell_constraints

report_cell_constraints

report_scan_cells

report_scan_chains

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_cell_models

Context: dft -scan, dft -test_points

Mode: setup, analysis

Specifies DFT library models for user-defined test points, system-generated test points, and system-generated test logic.

Usage

add_cell_models ***dftlib_model*** **-Type**
{**BUFfer** | **INVerter** | **AND** | **OR** | **NAND** | **NOR** | **XOR** | **CLOCK_BUFfer**
| {**MUX** ***mux_select mux_in0 mux_in1***}
| {**DFF** ***clock data_in*** [-POSEDGE_clock | -NEGEDGE_clock]}
| {**LATCH** ***enable data_in*** [-ACTIVE_HIGH_clock | -ACTIVE_LOW_clock]}
| {**SCAN_CELL** ***clock data_in*** [-POSEDGE_clock | -NEGEDGE_clock]}}

Description

When adding test logic circuitry, multiple models from the DFT library are used. The cell_type attribute in the library model description specifies the components available for use as test logic. You can use the add_cell_models command instead of defining the cell_type attribute in the library model description.

For more information on the design library attribute, see cell_type in the *Tessent Cell Library Manual*.

Note

The add_cell_models command does not recognize `celldefine cells as library cells and issues an error stating that these cells are not found in the library.

Arguments

- ***dftlib_model***

  A required string that specifies the name of a cell model in the DFT library used for test logic, buffer tree, lockup cell, or test point insertion.

- **-TYPE**

  A required switch and argument pair that specifies a model type. Options include the following:

  INVerter — A literal specifying an inverter cell.

  AND — A literal specifying an AND cell.

  BUFfer — A literal specifying a buffer cell.

  OR — A literal specifying an OR cell.

  NAND — A literal specifying a NAND cell.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

NOR — A literal specifying a NOR cell.

XOR — A literal specifying an exclusive OR cell.

CLOCK_BUFfer — A literal specifying a clock buffer cell.

MUX ***mux_select mux_in0 mux_in1*** — A literal and three strings specifying a 2-1 multiplexer and the names of the selector pin and both data pins.

SCAN_CELL ***clock data_in*** [-POSEDGE_clock | -NEGEDGE_clock] — A literal and two strings specifying a mux-DFF scan cell with the following input pins: clock, data, scan in, and scan enable

You must specify the names of the clock and data pins only of the DFT library cell model. You may also include an optional switch to specify the inversion on the clock pin. This option works in conjunction with the add_tied_signals command.

DFF ***clock data_in*** [-POSEDGE_clock | -NEGEDGE_clock] — A literal and two strings specifying a D flip-flop with input pins clock and data. You must specify the names of the clock and data pins only of the DFT library cell model. This option works in combination with the add_tied_signals command. If you are defining this model for use with lockup cells, you may also use an optional switch to specify the overall inversion on the clock pin.

LATCH ***enable data_in*** [-ACTIVE_HIGH_clock | -ACTIVE_LOW_clock] — A literal and two strings specifying a D latch with input pins enable and data. You must specify the names of the enable line and the data pin only of the DFT library cell model. If you are defining this model for use with lockup cells, you may also use an optional switch to specify the overall inversion on the clock pin. If you use this command to add an active_low latch, you must specify "add_cell_models *dftlib_model* -type latch *enable data* -active_low_clock". The default is -active_high_clock.

Examples

The following example shows a typical use of test logic involving the set, reset, and clock pins on sequential elements (flip-flops). The tool can usually ensure controllability of sequential elements with model types of And, Or, and, Mux.

```
add_clocks 0 clk
set_test_logic -set on -reset on -clock on
set_system_mode analysis
…
add_cell_models and2 -type and
add_cell_models or2 -type or
add_cell_models mux21h -type mux si a  b
```

Related Topics

delete_cell_models

report_cell_models

set_test_logic

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_chain_masks

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup and analysis, when EDT is off

Mode: setup only, when EDT or LBIST is on

Mode: setup only for -Load_value switch

Mode: setup only for -instance -block_chain_index_list switch pair

Mode: setup only for -cell_constraints switch

Masks the load, capture, and unload values on the specified scan chains during simulation.

Usage

Available when EDT is Off

add_chain_masks ***chain_name...*** [-Cell_constraints {OX | XX | TX}] [-UNload_value X]

Available when EDT is On

add_chain_masks {***chain_name...*** | **-INStance** ***instance_name*** **-Block_chain_index_list** ***block_chain_index_list*** ...} [-Cell_constraints {OX | XX | TX}] [-USed_chains {OFf | ON}] [-Load_value {0 | 1}] [-UNload_value {X | 0 | 1}]

LogicBIST Simulation

add_chain_masks ***{chain_name...*** | **-INStance** ***instance_name*** **-Block_chain_index_list** ***block_chain_index_list*** ...} [-Cell_constraints {OX | XX | TX}] [-Load_value {0 | 1}] [-UNload_value {0 | 1}]

Description

The add_chain_masks command enables you to mask specified scan chain values from the test results. This command also enables you to specify a circuit response (0/1) to the compactor.

You can use this command to specify that custom masking logic exists between the scan chain outputs and the compactor (when EDT is ON), between the scan chain outputs and output pins (when EDT is OFF), or on the input of the scan chains. You must create and appropriately control the custom masking logic.

Logic BIST Simulation

For logic BIST simulation, this command enables you to specify the scan chains that are masked during fault simulation and their load and unload values. This is useful to get around design issues or broken scan chains post manufacturing. If you use the hybrid TK/LBIST controller, then this command is used to specify the scan chains that need to be masked and the constrained load values for those chains. The tool automatically creates the data to be loaded into the static masking register to implement the masking. On the other hand, if you use your own LBIST controller, then this command enables you to specify the chain masking values corresponding to the hardware that you have added.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When you use a single scan chain mask register to mask a group of scan chains, the add_chain_masks command automatically applies the masking information from the specified masked chain to all other scan chains in the group.

Pattern Generation with EDT On

When using the Hybrid EDT/LBIST controller in EDT mode, and you use a single scan chain mask register to mask a group of scan chains, the add_chain_masks command automatically applies the masking information from the specified masked chain to all other scan chains in the group.

Pattern Generation with EDT IP TCD and Unused Chains

The EDT IP TCD flow enables you to specify that a chain is present but inactive and not to be traced. You specify these inactive (*unused*) chains by using a core instance (using -Instance) and the list of chain indices on the EDT controller instead of specifying chain names. You identify the inactive chains with the -used_chains switch. The -used_chains switch refers to the chain(s) references and is used as follows:

- -used_chains OFf — Tracing of the scan chain(s) is disabled.
- -used_chains ON — Tracing of the scan chain(s) is enabled (default).

If the chain is referenced using a core instance, then specification of -used_chains off/on is mandatory.

Arguments

- ***chain_name...***

  Required, repeatable string that specifies the name of a scan chain in the current design.

- **-INStance *instance_name***

  Required switch and value that specifies an EDT core instance. Must be specified as a set with -Block_chain_index_list. When this set is used, the masking applied to the specified chain is OX.

- **-Block_chain_index_list *block_chain_index_list***...

  Required switch and value that specifies a list of chain indices on the EDT controller. Must be specified as a set with -INStance. When determining chain indices for this argument, the first chain on the specified instance is counted as “1”. When this set is used, the masking applied to the specified chain is OX.

- -Cell_constraints {OX | XX | TX}

  Optional switch and literal pair that specifies the cell constraints and observe values for the masking of the specified scan chain(s).

  OX — Load and capture values are applied, and the fault observation is not included in the results.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

XX — Load and unload values are not applied, and values are captured normally for all scan cells in the scan chain for multiple-cycle tests.

TX — All scan cell values are tied as X.

- -Load_value {0 | 1}

  Optional switch and literal pair that specifies the value that is applied to the (internal) scan chain input. Options are:

  0 — Specifies the load value is 0.

  1 — Specifies the load value is 1.

  For EDT, this command enables you to load a value (0/1) at the input of the specified (internal) scan chains. It is assumed that you insert the hardware that constrains the loading value.

  For LBIST, this command is used to specify the value (0/1) to be loaded at the input (internal) of the scan chains. The value loaded should be consistent with the hardware that has been added to constrain the loading value at the scan chain inputs.

  If there are cell constraints added on a scan chain and also "add_chain_masks -load_value" is applied on the same chain, the tool issues an error. EDT Finder is disabled automatically when using "-load_value" for EDT.

- -UNload_value {X | 0 | 1}

  Optional switch and literal pair that specifies the unload value used to calculate the circuit response. Options include:

  X — Specifies the unload value is X. This value is valid with EDT On or Off. This value is not valid for Logic BIST simulation.

  0 — Specifies the unload value is 0. This value is valid only for EDT On, and for Logic BIST simulation.

  1 — Specifies the unload value is 1. This value is valid only for EDT On, and for Logic BIST simulation.

- -USed_chains {OFf | ON}

  Optional switch and literal pair that specifies whether the tool traces chain references. This switch is only supported with EDT core instances. Refer to "add_core_instances" on page 157. Choose from the following:

  OFf — Tracing of the scan chain(s) is disabled.

  ON — Tracing of the scan chain(s) is enabled (default).

  The following usage conditions apply:

  - If a bypass chain only includes unused chains, then the tool does not add the chain.
  - If a bypass chain includes a mix of used and unused chains, the tool issues a warning.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- If all bypass chains have unused chains such that there are no bypass chains left to add, the tool issues an error message.

If one or more chains are defined as unused, a message indicating the number of unused scan chains and that they are not traced is shown in the summary of the scan tracing when the system transitions to analysis mode (see "Example 3" on page 134). Any cells belonging to those unused chains (and not part of other traced chains) are treated as non-scan cells because they are not traced.

For further details on using this switch in the TCD flow, refer to the section "Pattern Generation with EDT IP TCD and Unused Chains" on page 132.

Examples

Example 1

The following example does not apply load values (no controllability) or unload values (no observability) and uses all the cells from *chain1* during a multicycle test. The 0 unload value is used to calculate the circuit response for the external channels.

add_chain_masks chain1 -cell_constraints xx -unload_value 0

Example 2

The following example shows the add_chain_masks in the pattern -scan context. The command masks the scan chain input by loading a 0, and masks the scan chain output by unloading a 0 on chains chain1 and chain2.

add_chain_masks chain1 chain2 -load_value 0 -unload_value 0

report_chain_masks

```
//  Masked scan chain  Load value  Unload value  Cell constraints
//  -----------------  ----------  ------------  ----------------
//        chain1            0            0              OX
//        chain2            0            0              OX
```

Example 3

In the following example, this command masks chain index 2 of the instance corea_rtl2_tessent_edt_c1_inst, which originally contains 15 scan chains:

**add_chain_masks -instance corea_rtl2_tessent_edt_c1_inst -block_chain_index_list 2 \
-used_chains off -unload_value 0**

The scan chain tracing summary indicates that only 14 chains are traced and 1 chain is not traced:

set_system_mode analysis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
...
//  ---------------------------------------------------------------------------
//  Begin EDT Finder analyses.
//  ---------------------------------------------------------------------------
//  Finding EDT logic.
//  Finding internal scan chains.
//  EDT Finder completed, EDT blocks=1, scan chains=14, CPU time=0.01 sec.
//  ---------------------------------------------------------------------------
//  Chain = corea_rtl2_tessent_edt_c1_inst__chain_1 successfully traced with
//  scan_cells = 45.
//  Chain = corea_rtl2_tessent_edt_c1_inst__chain_3 successfully traced with
//  scan_cells = 45.
...
//  Chain = corea_rtl2_tessent_edt_c1_inst__chain_13 successfully traced with
//  scan_cells = 45.
//  Chain = corea_rtl2_tessent_edt_c1_inst__chain_14 successfully traced with
//  scan_cells = 45.
//  Chain = corea_rtl2_tessent_edt_c1_inst__chain_15 successfully traced with
//  scan_cells = 42.
//  626 scan cells have been identified in 14 scan chains. 1 scan chain is defined
//  as unused and thus not traced.
```

Related Topics

add_scan_chains

delete_chain_masks

report_chain_masks

reset_di_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_clocks

Context: all contexts

Mode: setup

Adds scan or non-scan clocks to the clock list for proper scan operation.

Usage

For synchronous source clocks:

add_clocks [*off_state*] ***port_list*** [-PULSE_Always | -PULSE_In_capture] [-Label *label*] [-dft_inject_node *name*] [-no_dft_connections]

add_clocks [*off_state*] ***pin_list*** [-PULSE_Always | -PULSE_In_capture] [-Label *label*] [-PSEudo_port_name *name*] [-CAPture_only] [-silent] [-dft_inject_node *name*] [-no_dft_connections]

add_clocks [*off_state*] ***port_list*** [-FREQ_Multiplier *int* -PULSE_Always] [-Label *label*] [-dft_inject_node *name*] [-no_dft_connections]

For asynchronous source clocks:

add_clocks ***port_list*** {-PEriod *time* [s | ms | us | ns | ps | fs]} [-FREQ_Multiplier *int*] [-Label *label*] [-dft_inject_node *name*] [-no_dft_connections]

add_clocks ***pin_list*** {-PEriod *time* [s | ms | us | ns | ps | fs]} [-PSEudo_port_name *name*] [-FREQ_Multiplier *int*] [-Label *label*] [-silent] [-dft_inject_node *name*] [-no_dft_connections]

For generated clocks (for ICL extraction and memory BIST pre-DFT DRC) patterns -ijtag and dft (with no sub context):

add_clocks ***pin*** -REFERENCE {*pin* | *port* | *pseudo_port*} [-REFERENCE_Inv {*pin* | *port* | *pseudo_port*}] [-FREQ_Multiplier *int*] [-FREQ_Divider *int*] [-PSEudo_port_name *name*] [-Label *label*] [-dft_inject_node *name*] [-no_dft_connections]

For adding a clock branch (for ICL extraction and memory BIST pre-DFT DRC):

add_clocks ***pin*** [-Branch] [-Label *label*]

Description

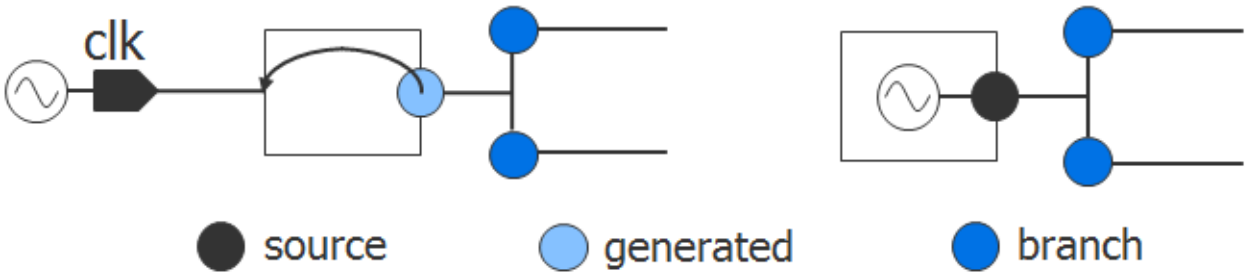
Declares a pin or a port as a clock. The defined clock may be of type source or a generated source. It may also be of type branch. The permitted types of clocks you can define depends on the current context you are using.

A clock is defined as being an asynchronous source when it is defined with a -period option. A clock is defined as generated when it has a -reference option. A clock is defined as a branch when it has the -branch option. Otherwise, the clock is considered a synchronous source. The various clock types are illustrated in Figure 3-1. The source clock represented with the black symbols are shown as asynchronous sources but they could also be synchronous sources. An asynchronous clock is free running and pulses with an arbitrary period. A synchronous clock

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

pulses or is kept idle under the control of the tester or in the ATPG simulator. The idle state is controlled using the off_state value.

Figure 3-1. Illustration of Various Clock Types



For ATPG and scan DRC, only synchronous clocks are supported. Branch clocks are ignored and generated clocks are converted into synchronous clocks by ignoring the specified -reference, -freq_divider and -freq_multiplier options. You can constrain a clock port to its off-state in order to suppress its use as a capture clock during the ATPG process.

For ICL extraction and memory BIST pre-DFT DRC, all clock types are supported. This enables you to fully describe the complete functional clock network as it pertains to the different instruments and test circuitry as well as by the various memories.

During ICL extraction, clock sources, generated clocks and branch clocks are represented with ICL constructs. Clock sources defined on ports of the current design are reported as ClockPorts with the tessent_clock_periods attribute capturing its period. The value of the tessent_clock_periods attribute is “all <period>”. This is to be ready for the time when the tool enables a clock to have a different period for different test modes. Today, there is only period described in ICL and it applies to all modes.

```
ClockPort <name> {
   Attribute tessent_clock_periods = "all <real>ns";
}
```

The clocks defined on internal pins with the add_clocks command are captured in ICL using a module named <design_name>_add_clock_<int> as shown here:

```
Module <design_name>_add_clock_<int> {
   // Created by ICL extraction
   Attribute tessent_clock_type = "source|generated|branch";
   ClockPort <input_name>;
   ToClockPort <output_name> {
      Source <input_name>;
      FreqMultiplier <int>;
      FreqDivider <int>;
   }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Those ICL modules are created only if the internal clock pin is found:

- When ICL Extraction traces from a module that is matched to an ICL module.
- When ICL Extraction traces backwards from a primary output that has been declared as a ToClockPort by means of "add_icl_ports *portname* -type to_clock".
- When ICL Extraction traces from another internal clock pin that has been found before.

In short, those ICL Modules representing the internal clocks are created only if there is something described in ICL to which the internal clock is connected.

The clock label is stored using the attribute tessent_clock_domain_labels when instantiating the ICL module. The *pin_name* value reflects the pin name in the design relative to the current design. This information is useful to generate SDC as the clock pin in the design is identified in the ICL instance.

```
Instance add_clock_inst_<int> Of <design_name>_add_clock_<int> {
  InputPort ClockIn = <source_name>;
  Attribute tessent_clock_domain_labels = "<label_name> {<pin_name>}";
}
```

When issuing the set_current_design in the dft context, source or generated clocks modeled in ICL are automatically inferred as add_clocks. This is done so that you do not have to declare them manually in order to perform the memory BIST pre-DFT DRC rules Scan Cell Data Rules (D Rules) to DFT_C5. Those clocks are automatically deleted when entering the patterns -ijtag context so that ICL extract sees your real ICL modules. To have your clocks automatically extracted during design elaboration, the ICL module matching the design module must have the syntax illustrated below where a ToClockPort exists with a Source pointing to a ClockPort on the same module. You use the Period element to model internal oscillators and the FreqMultiplier and FreqDivider elements to model PLLs and clock dividers. If your PLLs and clock dividers are programmable, you can easily enable their programming using IJTAG if you follow the description found in the ProcedureStep section.

```
Module <module_name> {
   ClockPort <input_name>;
   ToClockPort <output_name> {
      Source <input_name>;
      FreqMultiplier <int>;
      FreqDivider <int>;
      Period <int><time_units>;
   }
}
```

Use the report_clocks command after having run the set_current_design command in the dft context to show the inferred clocks, if any were inferred. They have the "inferred_icl_clock<int>" naming convention, as shown below. Use the report_ijtag_instances command to report the design instance for which an ICL module was matched. Use the "set_design_sources -format" ICL command to modify the search path for ICL files.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
User-defined Clock (1):
=========================

  Generated Clock
  ================
   -------------------  -----  --------------  ---------  ---------------
   Name                 Label  Reference       FreqMult   Pin(s)
   -------------------  -----  --------------  ---------  ---------------
   inferred_icl_clock0  ck0    clk_div/clk_in    1 / 4    clk_div/clk_out
```

Arguments

- *off_state*

  An optional literal that specifies the off state of the clock. This option is only used for synchronous clock sources as asynchronous and generated clocks cannot be stopped. The off state of a branch clock is inferred from the off state of the clock in its fan-in. The choices are:

  0 — A literal specifying the off-state value is 0 (default value).

  1 — A literal specifying the off-state value is 1.

  Note

  The tool ignores the off_state value when specified for asynchronous and generated clocks.

- ***pin_list***

  A repeatable string that lists the internal pins to define as clocks. The strings can include any number of asterisk (*) and question mark (?) wildcard characters and they must match pin names in the current design.

- ***port_list***

  A repeatable string that lists the primary input ports to assign as clocks. This string can include any number of asterisk (*) and question mark (?) wildcard characters and they must match port names on the current design.

- -PSEudo_port_name *name*

  An optional switch and string pair that specifies the name for a new primary input clock pin that drives all of the internal pins specified with the pin_list argument. The name argument must be a simple name that begins with an alpha character and can contain only alphanumeric characters and underscore characters ('_'). Wildcards are not permitted. In ATPG application, you can use this option to map a list of internal clock pins to a single pseudo port. In all other application, the pin_list can only include one pin name when specifying the -pseudo_port_name option so that each internal clock is associated with a unique pseudo port name.

- -CAPture_only

  An optional switch specifying that the corresponding pseudo port is only treated as a source clock during capture. For test procedures other than capture, the tool treats this clock pin as

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

an internal pin; therefore, its simulation value is decided by its driver in the netlist. An event on a capture_only clock in a non-capture procedure results in a P61 warning during DRC. This setting does not extend the capture_only behavior to equivalent clocks.

- -PULSE_Always

  An optional switch that identifies the specified pin as a pulse-always clock. A pulse-always clock is pulsed during every cycle of the pattern set.

  During ATPG, the clocks you specify as pulse-always are permitted to also pulse with any other clock, ignoring the specified clock restriction setting defined by the set_clock_restriction command. Pulse-always clocks are automatically pulsed in the external mode of named capture procedures (if present). For more information on named capture procedures, refer to "Capture Procedures (Optional)" in the *Tessent Shell User's Manual*. Unless you define an asynchronous free-running clock, you must specify the clock in the internal mode of named capture procedures in order to simulate the effect of the clock pulses.

  An Always Block is automatically created in the procedure file if it does not already exist, and the pulse-always clock is pulsed in this block.

  Note

  SSN bus clocks are not permitted to be specified as pulse-always. If you specify an SSN bus clock as pulse-always, the tool reports an SSN_R23 error during DRC.

- -PULSE_In_capture

  An optional switch that identifies the specified pin as a pulse-in-capture clock. This information is only used during ATPG and ignored in all other applications. A pulse-in-capture clock is pulsed in every cycle of the capture window. The clock you specify also pulses with any other clock, ignoring the specified clock restriction setting defined using the set_clock_restriction command.

- -PEriod *time_value* [s | ms | us | ns | ps | fs]

  An optional switch and real number pair that specifies the period of a free-running asynchronous clock. The time_value is a real number followed by an optional time unit that defaults to "ns." Note that ATPG and scan DRC treat asynchronous free-running clocks as X values during the capture cycles.

  The period of an asynchronous free-running clock affects the timescale and precision of the Verilog testbench. For more information, refer to the description of the "set time scale" statement in the *Tessent Shell User's Manual*.

- -Label *label*

  An optional switch and string pair that specifies an arbitrary label for a specified clock. This switch is not valid if the pin_list or port_list values have more than one element. The label name must be a single string that begins with a letter and consists only of letters, numbers, and underscores. Wildcards are not permitted.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The clock label that can optionally be specified with the add_clocks command only works with the following commands: get_clocks, get_clock_option, set_clock_options, and delete_clocks. For all other commands, you must still use the actual clock pin, port, pseudo-port name, or collection returned by the get_clocks command.

If you do not specify a label, the tool automatically assigns a label using the following algorithm:

```
if clock is a port
   if clock name begins with "\" (escaped)
     go to "Choose CK# label step"
   else if port is a bus
      take the scalar name followed with _# corresponding to the bit
       index as the label name
       if the label name does not pass the uniqueness check
          go to "Choose CK# label step"
   else if the port name begins with [a-z][A-Z], and only contains
       letters, numbers and _,
       if it passes the uniqueness check
         use the port name as the clock label name
       else
         go to "Choose CK# label step"
If the clock is on a pseudo port
  if the port name begins with [a-z][A-Z], and only contains
         letters, numbers and _,
    if it passes the uniqueness check
      use the port name as the clock label name
    else
      go to "Choose CK# label step"

Choose CK# label step:
   Take ck#, where # is chosen as the next available integer such
   that "ck#" is not an already used clock label and is not an
   existing top-level port name or an existing pseudo port name.
   The only time the label "CK#" is permitted to be used when a
   port or a pseudo port by the same name already exists is when
   the clock is defined on that port or that pseudo port itself.
```

- -dft_inject_node *dft_inject_node*

  An optional switch and string pair that specifies where the DFT object affecting the clock (for example, a TCK injection multiplexer) should be inserted along the clock path. You can introspect the status of this switch with the "get_clock_option -dft_inject_node" command and switch.

- -no_dft_connections

  A Boolean switch that instructs the tool to not infer DFT connections to this clock. When choosing the clock for a dedicated wrapper cell or a test_point in the insert_test_logic command, the clock used by the majority of the flip-flops in the fan-in or fanout of the node is normally used. If that clock was added with the -no_dft_connections switch, it is not chosen and instead the next majority clock is used. If all clocks the fan-in or fanout were

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

added with the -no_dft_connections switch, the largest clock domains in the design without this switch are selected.

- -FREQ_Multiplier *int*

  An optional switch and integer pair that specifies an integer to reflect the multiplication factor the clock has with respect to its specified -reference value.

- -FREQ_Divider *int*

  An optional switch and integer pair that specifies an integer to reflect the division ratio the clock has with respect to its specified -reference value. You can also use this option with the -period option to define how a clock frequency is divided from a base period. The tool adds special logic in the Verilog testbench to generate all clocks that have the same base period and make sure they remain coherent during simulation.

- -REFERENCE {*pin* | *port* | *pseudo_port*} (**"dft (no sub-context)" and "patterns -ijtag" only**)

  A switch and string pair that specifies the clock that is the reference of the clock. The multiplication and division relationship between the clock and its reference is described using the -freq_multiplier and -freq_divider options.

  Cascaded clocks should first declare the source clock, followed by the reference clocks.

- -REFERENCE_Inv {*pin* | *port* | *pseudo_port*} (**"dft (no sub-context)" and "patterns -ijtag" only**)

  An optional switch and string pair that specifies the second primary input port if the reference pin is a differential pin.

- -Branch

  An optional switch that defines a pin or a port as a clock branch. You can use this switch in conjunction with the -Label switch.

  A clock branch shares a source with one or more clocks but it is not skew-balanced with the other clocks that share the common source. The -branch switch is useful for describing branches in a clock tree that are balanced within themselves but not with the other branches of the tree. Refer to Figure 3-1 for a graphical representation of a clock branch.

- -silent

  An optional switch that suppresses the transcription of notes the tool issues.

Examples

Example 1

The following example adds two external pulse-always clocks with off states of 1:

```
SETUP> add_clocks 1 I2 I3 -pulse_always
SETUP> report_clocks
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
User-defined Clocks (2):
Sync and Async Clocks
=====================
 ----  ---------  ------------  --------
 Name  Off State   Constraints  Internal
 ----  ---------  ------------  --------
'I2'       1      Pulse always     No
'I3'       1      Pulse always     No
```

Note that the report_clocks command does not report a label name because, in the case of ports, the tool generates a label name that is identical to the port name.

Example 2

The following example adds an internal clock with an off state of 0. The add_clocks command automatically creates a PI at the new clock pin.

SETUP> add_clocks 0 u4/u2/CLK

```
//  Warning: Primary input 'u4/I6' is added at pin '/u4/u2/CLK'
```

SETUP> report_clocks

```
User-defined Clock (1):
=========================

  Sync and Async Source Clock
  ============================
   -----  -----  ---------  -----------  --------  ---------
    Name  Label  Off State  Constraints  Internal  Pin(s)
   -----  -----  ---------  -----------  --------  ---------
   u4/I6   ck0       0                      Yes    u4/u2/CLK
```

SETUP> report_primary_inputs

```
USER:    '/u4/I6' (internal pin)
SYSTEM:    '/I1'
...
```

Note that the report_clocks command reports the label name that the tool automatically generates for internal pins.

Example 3

The following example adds two internal clocks and merges them together under a pseudo_port name of port_1:

SETUP> add_clocks 0 u4/u2/CLK u5/u2/CLK -pseudo_port_name port_1

```
//  Note: Primary input 'port_1' is added, merging 2 pins.
```

SETUP> report_clocks

```
User-defined Clock (1):
=========================
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Sync and Async Source Clock
============================
 ------  ---------  -----------  --------  ------------------ ---------
 Name    Off State  Constraints  Internal  Other Properties     Pin(s)
 ------  ---------  -----------  --------  ------------------ ---------
 'port_1'      0                    Yes    Merged internal pin u4/u2/CLK
                                                                u5/u2/CLK
```

Example 4

The following example adds an external pulse-in-capture clock with a label name:

SETUP> add_clocks 0 I2 -pulse_in_capture -label clock_2

SETUP> report_clocks

```
User-defined Clock (1):
=========================

  Sync and Async Source Clock
  ============================
   ----  -------  ---------  ----------------  --------
   Name   Label   Off State     Constraints    Internal
   ----  -------  ---------  ----------------  --------
   'I2'   clock_2      0      Pulse in capture     No
```

Example 5

The following example specifies two asynchronous free-running clocks that are coherent. The overall period for the coherent group is 1000ns. The clock “aclk” pulses 20 times during that period while “bclk” pulses 30 times during that period, both with a 50 percent duty cycle:

SETUP> add_clocks aclk -period 1000 ns -freq_multiplier 20

SETUP> add_clocks bclk -period 1000 ns -freq_multiplier 30

SETUP> report_clocks

```
User-defined Clocks (2):
=========================

  Sync and Async Source Clocks
  ============================
   ----  ---------  -----------  --------  --------------  --------------------
   Name  Off State   Constraints  Internal       Period      Freq. Multiplication
   ----  ---------  -----------  --------  --------------  --------------------
   'aclk'      -      Asynchronous     No     1000.00ns              20
   'bclk'      -      Asynchronous     No     1000.00ns              30
```

The annotations and comments to document the asynchronous free-running clocks in the pattern files occur in the header section of the pattern file right after the block of comments/annotations

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

that are labeled "Begin_Verify_Section." The annotations are grouped in a block with a begin and end annotation. Here is an example of the WGL block:

```
{  Begin_async_clocks }
{     aclk = period: 1000.00ns, divider: 20 }
{     bclk = period: 1000.00ns, divider: 30 }
{  End_async_clocks }
```

All other pattern formats use the identical block structure and syntax within the annotation or comment. The only differences are using the correct annotation or comment punctuation or statements surrounding this block.

Example 6

The following example adds an internal clock with a non-integer multiplication factor:

SETUP> add_clocks clk -period 40 ns

SETUP> add_clocks /path/to/pll/o -reference clk -freq_mult 610 -freq_div 25 -label clk610

The primary input clock has a frequency of 25 MHz. The internal clock has a frequency of 610 MHz. Therefore, the multiplication factor is 610/25, or 24.4.

The value for the -freq_multiplier argument must be an integer, so you use -freq_multiplier and -freq_divider together to specify the internal clock's frequency.

The following provides an equivalent result:

SETUP> add_clocks /path/to/pll/o -reference clk -freq_mult 122 -freq_div 5 -label clk610

Example 7

The following example works in "patterns -ijtag" only. The following commands produce an internal half-speed clock on the ICL module "chip" during ICL extraction:

set_design_sources -format icl -y icl_data
set_current_design chip
add_clocks { pll_block/clock1 } -reference clock1 -freq_divider 2
set_system_mode analysis

The resulting ICL created by the ICL extraction looks as follows:

```
Module chip {
   // Created by ICL extraction
   ClockPort clock1;
   …
   Instance block1_I1 Of block1 {
      InputPort ClkA = add_clock_inst_0.ClockOut;
       …
   }
   Instance add_clock_inst_0 Of chip_add_clock_0 {
      InputPort ClockIn = clock1;
   }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// instanced as chip.add_clock_inst_0
Module chip_add_clock_0 {
   // Created by ICL extraction
   ClockPort ClockIn;
   ToClockPort ClockOut {
      FreqDivider 2;
      Source ClockIn;
   }
}
```

Related Topics

analyze_control_signals

delete_clocks

get_clocks

get_clock_option

report_clocks

set_clock_options

set_clock_restriction

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_config_element

Context: unspecified, all contexts

Mode: all modes

Adds a configuration element in the configuration data.

Usage

add_config_element [name] [-in_wrapper *wrapper_object*] [-copy_from *config_object_spec*]
[-before *config_object_spec* | -after *config_object_spec* | -first | -last]
[-type wrapper | data_wrapper | csv_wrapper | property] [-value *value_list*]
[-replace]

Description

The command returns a single element collection that contains the newly-created element.

The behavior of the command is different when adding an element inside a wrapper that fully defines the permitted child elements versus one that permits arbitrary elements. The permitted content of a wrapper is controlled by its metadata description. Wrappers of type single and simple properties exist automatically as soon as their parent wrapper is added. Therefore, these elements cannot be added using this command as they are automatically added when their parents are added. You use the set_config_value to specify their value.

You can add elements inside a parent wrapper that does not constrain its content (that is, its metadata has the property AllowUnknown set to on), but you need to add everything inside it and define the type of object using the -type property. Refer to Example 2 for an example that creates a configuration tree with no metadata.

Arguments

- *name*

  An optional string that specifies the name of the configuration element being added. When using the -in_wrapper option, the name is the relative pathname with respect to the specified wrapper. When not using the -in_wrapper option, the name is the complete pathname with respect to the root of the partition. The parent wrapper of the element must exist before the element can be added to it. For example DftSpecification(moda,rtl) must exist to add the MemoryBist wrapper into it using the “add_config_element DftSpecification(moda,rtl)/MemoryBist” command. The added element must not already exist unless the -replace option is used, in which case the existing element is deleted before being added.

- -in_wrapper *wrapper_object_spec*

  An option value pair that specifies that the configuration element is to be added inside a specific wrapper. When the -in_wrapper option is specified, the name option defines a name relative to the specified destination wrapper. The *wrapper_object_spec* can be the complete name of a wrapper relative to the root of the partition or a collection containing a single wrapper element as returned by get_config_elements or “get_config_value -object”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The default partition defines wrappers by the name of tmp(id) that permit unknown elements to be added inside them. If you want to read an arbitrary configuration wrapper into the tool that does not match any defined syntax, you can use the "add_config_element tmp(id)" command to create an empty tmp wrapper. The id can be any arbitrary label and you can create as many tmp wrappers as you need. Once created, you can then use the "read_config_data file -in_wrapper tmp(id)" command to load arbitrary configuration data inside the tmp(id) wrapper. Refer to Example 2 for an example of this usage.

- -copy_from *config_object_spec*

  An optional value pair that specifies that the added element is a copy of another element. When the permitted content of the destination parent wrapper is defined with Wrapper, the metadata definition of the source and destination element must be identical (that is, both definitions must reference the same metadata definition). If this is not the case, the tool issues an error message. You can omit the name of the added element when using both the -copy_from and the -in_wrapper options. In this case, the parent wrapper of the copied element and the destination wrapper must be different. If you do not specify the name option, the element is added into the destination wrapper using the leaf_name of the copied object.

- -before *config_object_spec*

  An optional value pair that specifies a configuration element that the new configuration element precedes. When you use the -before option with a collection containing a configuration element object, you cannot use the -in_wrapper option because the parent wrapper is extracted from the object. Also, the name option must be a leaf_name with no hierarchical component.

  When the *config_object_spec* value is a name, it must be a name relative to the object specified with the -in_wrapper or -copy_from options. If you do not specify either option, it is a name relative to the root of the partition.

  The -before option is mutually exclusive to the -after, -first, and -last options. When you specify none of these four options, the tool uses the -last option.

- -after *config_object_spec*

  An optional value pair that specifies a configuration element after which to add the new configuration element. When you use the -after option with a collection containing a configuration element object, you cannot use the -in_wrapper option because the parent wrapper is extracted from the object. Also, the name option must be a leaf_name with no hierarchical component.

  When the *config_object_spec* value is a name, it must be a name relative to the object specified with the -in_wrapper or -copy_from options. If you do not specify either option, it is a name relative to the root of the partition.

  The -after option is mutually exclusive to the -before, -first, and -last options. When you specify none of these four options, the tool uses the -last option.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -first

  An optional switch that specifies to add the new configuration element before any other element in the parent wrapper.

  The -first option is mutually exclusive to the -before, -after, and -last options. When none of these four options is specified, -last is assumed.

- -last

  An optional switch that specifies to add the new configuration element after all other elements in the parent wrapper.

  The -first option is mutually exclusive to the -before, -after, and -first options. When none of these four options is specified, -last is assumed.

- -type wrapper | data_wrapper | csv_wrapper | property

  An optional value pair that is only needed and permitted when adding a configuration element for which no Wrapper exists. When adding an element for which metadata exists, the -type option is not permitted because the type of the object is specified in the metadata. When adding an element for which no metadata exists, the -type value defaults to “wrapper” when the -value option is not specified and to “property” when the -value option is specified.

  Refer to Example 2 for an example of its usage when adding an element without metadata.

- -value *value_list*

  An optional switch and value pair that is used when adding a configuration element that needs a value. The -value option is only used when adding a repeatable property such as the connection property in the Tdr/DataOutPorts wrapper. The supplied value needs to be a well-formatted Tcl list so as to set a value that contains white spaces and is similar to the following:

  ```
  -value [list "a b c"]
  ```

- -replace

  An optional switch that suppresses the error that is normally generated when adding a configuration element with a name that conflicts with an existing element. The error only occurs if the element is defined as not repeatable in the metadata. Instead of an error, the existing element is first deleted.

Examples

Example 1

The following example creates a DftSpecification using the add_config_element command. The generated configuration data is then reported.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_transcript_style input_only
set spec    [add_config_element DftSpecification(moda,ex1)]
set ijtag [get_config_elements IjtagNetwork -in_wrappers $spec]
set host [add_config_element HostScanInterface(ijtag) -in $ijtag]
set mbist   [add_config_element MemoryBist -in $spec]
set_config_value ijtag_host_interface -in $mbist HostScanInterface(ijtag)
set ctl     [add_config_element Controller(c1)  -in $mbist]
set step    [add_config_element Step -in $ctl]
set mem_int [add_config_element MemoryInterface(m1) -in $step]
set_config_value memory_library_name -in $mem_int ram12x4
set mem_int [add_config_element ReusedMemoryInterface(m2) -in $step]
set_config_value reused_interface_id -in $mem_int m1
set step    [add_config_element  Step -in $ctl]
set mem_int [add_config_element MemoryInterface(m3) -in $step]
set_config_value memory_library_name -in $mem_int ram12x8
set mem_int [add_config_element ReusedMemoryInterface(m4) -in $step]
set_config_value reused_interface_id -in $mem_int m3
report_config_data DftSpecification(moda,ex1)
```

```
DftSpecification(moda,ex1) {
   IjtagNetwork {
    HostScanInterface(ijtag) {
    }
   }
   MemoryBist {
    ijtag_host_interface : HostScanInterface(ijtag);
    Controller(c1) {
      Step {
        MemoryInterface(m1) {
          memory_library_name : ram12x4;
        }
        ReusedMemoryInterface(m2) {
          reused_interface_id : m1;
        }
      }
      Step {
       MemoryInterface(m3) {
          memory_library_name : ram12x8;
        }
        ReusedMemoryInterface(m4) {
          reused_interface_id : m3;
        }
      }
    }
  }
}
```

Example 2

The following examples adds a tmp wrapper that enables anything inside it to show how arbitrary configuration data can be created. Notice the use of the -type options when adding a data row and a csv row. The -type option is not needed when adding a wrapper or a property because it automatically defaults to wrapper or property based on the presence of the -value option.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set tmp1  [add_config_element tmp(1)]
set n_wpr [add_config_element NormalWrapperExmaple(abc) -in $tmp1]
add_config_element -in $n_wpr prop_example -value [list a b c]
set d_wpr [add_config_element DataWrapperExample -in $n_wpr]
 # -type is needed otherwise it would default to property as
 # -value is not specified
add_config_element -in $d_wpr element1 -type property
add_config_element -in $d_wpr element2 -value 32ns
set c_wpr [add_config_element CsvWrapperExample -in $n_wpr]
add_config_element -in $c_wpr -value [list a b c]
add_config_element -in $c_wpr -value [list d e]
add_config_element -in $c_wpr -value [list f g h] -after <0>
report_config_data $tmp1
```

```
tmp(1) {
  NormalWrapperExmaple(abc) {
    prop_example : a, b, c;
    DataWrapperExample {
      element1;
      element2 : 32ns;
    }
    CsvWrapperExample {
      a, b, c;
      f, g, h;
      d, e;
    }
  }
}
```

Related Topics

delete_config_element

get_config_elements

move_config_element

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_config_message

Context: all contexts

Mode: all modes

Adds error, warning, or information messages to configuration elements.

Usage

add_config_message **-config_object *config_object message***
[-error | -warning | -info | -note] [-display]

Description

Adds error, warning, or information messages to configuration elements.

Arguments

- **-config_object *config_object***

  A required value that specifies the configuration object to which to add the message specified by the message string. The ***config_object*** value is the name of a configuration element or a collection containing a single configuration element object.

- ***message***

  A required string value that specifies the text being added.

- -error

  An optional Boolean option that specifies that the messages to be added are of type error.

- -warning

  An optional Boolean option that specifies that the messages to be added are of type warning.

- -info

  An optional Boolean option that specifies that the messages to be added are of type info.

- -note

  An optional Boolean option that specifies that the messages to be added are of type note.

- -display

  An optional switch that specifies to also display the error, warning, or information messages in the transcript. Without this option, the message is silently added to the configuration object.

  If the file_path_name and line_number attributes are set on the configuration object, then the message displays the pathname to the file and the line number in the file where the configuration element is defined. For example:

```
// Warning: /Core(SYNC_1RW_512x64x2)/Memory/OperationSet
//          Read from file: ./SYNC_1RW_512x64x2.lib, line number: 8
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example adds an error message to a property element. The -display option specifies to also report the error message in the transcript window.

```
add_config_message -config_object /tmp(1)/propertyA -error -display \
"The value A is inconsistent with the design level '[get_design_level]'."
```

Example 2

The following example illustrates the differences between -error, -warning, -info and -note as to how the messages are displayed in the transcript window:

```
// Error: /Core(SYNC_1R1W_16x8)/Memory
//        The memory uses a checkerboard test algorithm SMarchCHKBvcd and has no
//        column address bits. Memories with checkerboard algorithm and no column
//        address bits must have BitGrouping property set to the data word size.
//        The BitGrouping value of this memory is changed to 8.

// Warning: /Core(SYNC_1R1W_16x8)/Memory
//          The memory uses a checkerboard test algorithm SMarchCHKBvcd and has no
//          column address bits. Memories with checkerboard algorithm and no
//          column address bits must have BitGrouping property set to the data
//          word size. The BitGrouping value of this memory is changed to 8.

//Core(SYNC_1R1W_16x8)/Memory
//          The memory uses a checkerboard test algorithm SMarchCHKBvcd and has no
//          column address bits. Memories with checkerboard algorithm and no
//          column address bits must have BitGrouping property set to the data
//          word size. The BitGrouping value of this memory is changed to 8.

// Note: /Core(SYNC_1R1W_16x8)/Memory
//       The memory uses a checkerboard test algorithm SMarchCHKBvcd and has no
//       column address bits. Memories with checkerboard algorithm and no column
//       address bits must have BitGrouping property set to the data word size.
//       The BitGrouping value of this memory is changed to 8.
```

Related Topics

delete_config_messages

get_config_messages

report_config_messages

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_config_tab

Context: all contexts

Mode: all modes

Adds one configuration tree tab to the Config Data Browser window.

Usage

add_config_tab [-partition *partition*] ***wrapper***

Description

Creates a configuration tree tab in the Config Data Browser window and adds a wrapper to it.

Arguments

- -partition *partition*

  An optional switch and value pair that specifies the partition containing the configuration wrapper that the tool adds to the Config Data Browser. When this option is not specified, the tool uses the default partition.

- ***wrapper***

  A value that specifies the name of the configuration wrapper. The value is either the name of a wrapper or a collection that contains exactly one configuration wrapper. If the wrapper is not a root wrapper, the command adds the root wrapper and selects the specified wrapper in the tab.

Examples

This example displays a tcd_memory wrapper in the Config Data Browser window.

```
add_config_tab -partition tcd /Core(RAM12x8)
```

Related Topics

delete_config_tabs

display_specification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_control_points

Context: dft -test_points

Mode: setup, analysis

Specifies user-defined control points during test point insertion.

Usage

add_control_points **-location** ***pin/port_spec*** **-type** {**AND** | **OR**} [-clock *clock_pin/port*] [-enable *enable_pin/port*]

Description

Use this command to specify your own control points during test point insertion.

Note

After running the add_control_points command, you must run the analyze_test_points command before running the analyze_scan_chains, analyze_wrapper_cells, or analyze_xbounding commands.

Arguments

- **-location** ***pin/port_spec***

  A required switch and string that specifies the pathname to the control point location.

  The ***pin/port_spec*** can be a Tcl list of one or more pins/ports or a collection of one or more pin/port objects.

- **-type {AND | OR}**

  A required switch and literal pair that specifies the gate type to be inserted at the control point location.

- -clock *pin/port_spec*

  An optional switch and string pair that specifies the path name of the clock signal used to clock the control point. This can be a collection or list, but should have only one element.

- -enable *pin/port_spec*

  An optional switch and string pair that specifies the path name of an existing net used to enable or disable the control point. This can be a collection or list, but should have only one element.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example adds a type OR control point and a type AND control point. For the AND control point it also specifies the path name of the clock signal used to clock the control point:

```
add_control_point -location gc[1] -type OR
add_control_point -location gd[1] -type AND -clock gclk4
report_test_points
```

Example 2

The following example specifies the path name of an existing net that is to be used to enable or disable the control points. For the AND control point it also specifies the path name of the clock signal used to clock the control point. Then it proceeds to insert the control points into the netlist:

```
set_test_point_insertion_options -control_point_enable cp_enable
add_control_point -location ga[1] -type OR -clock buf1/Z
report_test_points
...
set_system_mode analysis
...
insert_test_logic
```

Related Topics

add_observe_points

delete_test_points

report_test_points

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_core_instances

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: setup

Prerequisites: A core description must currently be in memory. You must first run the read_core_descriptions command.

Adds a core instance to the design by associating the core description currently in memory with the specified core instances in the design.

Usage

add_core_instances [-core *core_name*] [-mode *mode_name*]
{**-modules *module_objects*** | **-instances *instance_objects*** | **-current_design**}
[-design_id *design_id*] [-parameter_values {*parameter_list*}] [-silent]

Description

Running this command associates a core instance description currently in memory with the specified core instances in the design. The tool automatically sets up the core instances for the next steps in the flow by automating tasks such as creating cutpoints at the outputs of the core. If the core description is not in memory, you can read it in using the read_core_descriptions command. You can report the available core descriptions using the report_core_descriptions command.

You can specify the cores with which to associate the core description by providing either absolute instance pathnames to the cores or uniquified module names, or by using the -current_design switch.

You must generate any Tessent instruments such as EDT, OCT, LPCT, or SSH by using the DftSpecification. The tool can only match ungrouped cores if they were generated with the add_persistent_buffers_in_scan_resource_instruments : on | off ; property set to on.

Restriction

Do not use the add_core_instances command to add SSN ScanHost instruments. The ScanHost instances are already inferred from ICL, and the active ScanHost instances are automatically identified.

Arguments

- -core *core_name*

  An optional switch and Tcl list that specifies the core name of the core description from which the instantiation should take place. If the -core switch is not specified, the tool attempts to match a core description against the module name of the specified modules or instances. If the tool does not find a direct match for a core description and a module or instance, the tool attempts to identify a match using the tool's default module matching options that are consistent with what synthesis tools typically use. For information on the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

default module matching options and how they can be changed, see the description for the set_module_matching_options -suffix_pattern_list option.

If the tool is unable to match a specified module or instance, it reports an error with a reminder that you can explicitly specify which core description to use with the -core switch.

- -mode *mode_name*

  An optional switch that specifies the mode of the core. This switch is required if the core has multiple modes.

- **-modules *module_objects***

  A required switch and string pair that specifies a Tcl list of one or more object names (hierarchical names) or a collection of one or more objects. The core instances are created for each instantiation of those objects in the design.

- **-instances *instance_objects***

  A required switch and string pair that specifies a Tcl list of one or more object names (hierarchical names) or a collection of one or more objects. The core instances are created for each instance in the list.

- **-current_design**

  A required switch that specifies to add the core instance with respect to the current design. This switch is mutually exclusive with the -modules and -instances switches. The mode of the core instance being added by the -mode switch must be different than the mode of the current design specified using the set_current_mode command.

- -design_id *design_id*

  An optional switch and string pair that specifies searching any opened Tessent Shell Data Base (TSDB) for Tessent Core Descriptions that match the *design_id* value.

  If you omit the -design_id switch and string pair and the added instance has the design_id attribute set, then the tool performs the matching based on the design_id attribute value. The design_id attribute is inherited by instance objects from their modules. Additionally, when you use "read_design -design_id *design_id*", the *design_id* attribute is set automatically for these modules and instances.

  If you do not set the *design_id* and there is only one TCD in the opened TSDBs that matches the core name, the tool loads the *design_id* in memory and adds the core instance. If the TCD is already loaded into memory the -design_id switch has no effect and the matching is preformed based on the core and module name.

- -parameter_values {*parameter_list*}

  An optional switch and parameter list that specifies instrument configuration parameters and their values. The parameter list is one or more parameter value pairs. For example:

```
SETUP> add_core_instances -core cpu_edt -modules cpu_edt \
       -parameter_values {edt_bypass off}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The list includes the subset of those parameters that are configurable in the specified instrument. Table 8-5 in the documentation for set_core_instance_parameters shows the core parameters and their instruments.

The parameters for core instances created with this command can be reported with the report_core_instance_parameters command. Their parameters may be modified with the set_core_instance_parameters command.

- -silent

  An optional switch that specifies to suppress warning messages.

Examples

Example 1

The "Retargeting Example" in the*Tessent Scan and ATPG User's Manual* demonstrates the use of this command in a design context.

Example 2

The following example demonstrates the use of the -current_design switch to merge two modes of a core; in this case, the "core" is the top level. In the example, patterns are generated separately for two sub-blocks of a design from the top-level pins. Sub-Block1 is tested as mode M1 of the top level, and Sub-Block2 is tested as mode M2 of the top level. The result, shown in Figure 3-3 on page 162, is the merging of the two pattern sets originally generated for each individual mode into a single pattern set applied at the top level.

The first dofile reads in the top level design and writes out the patterns and core description files for mode M1 targeting Sub-Block1:

```
// Invoke tool to generate patterns for each mode of top_level
set_context pattern -scan
read_verilog top_level.v                    // Read netlist of top_level

// Mode M1 consists of the Sub-Block1 design; the user must blackbox Sub-Block2 and apply
// constraints to access Sub-Block1
set_current_mode M1 -type internal
...
set_system_mode analysis
...
create_patterns
write_patterns top_levelM1.ascii
write_core_description top_levelM1.tcd
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

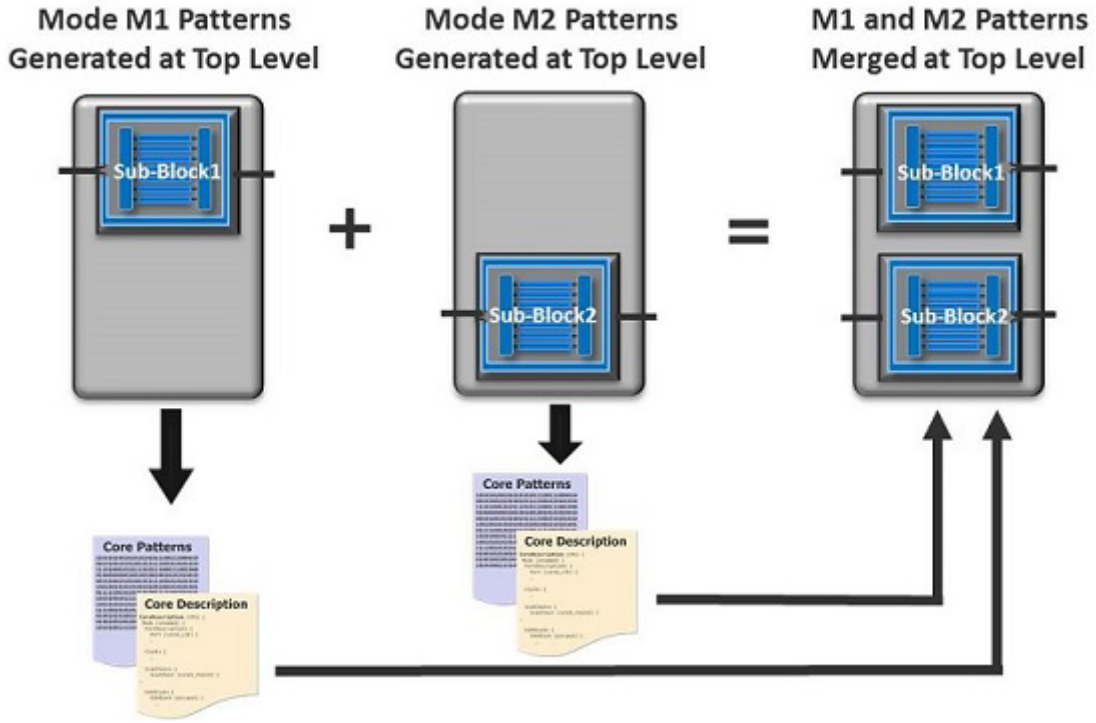
The second dofile reads in the design and writes out the patterns and core description files for mode M2 targeting Sub-Block2:

```
// Mode M2 consists of the Sub-Block2 design; the user must blackbox Sub-Block1 and apply
// constraints to access Sub-Block2
set_current_mode M2 -type internal
...
set_system_mode analysis
...
create_patterns
write_patterns top_levelM2.ascii
write_core_description top_levelM2.tcd
```

The third dofile merges the two separately generated top-level pattern sets (modes M1 and M2) into a single pattern set:

```
// Invoke the tool again to retarget patterns to top level
set_context patterns -scan_retargeting
...
read_core_description top_levelM1.tcd
read_core_description top_levelM2.tcd
add_core_instance -core top_level -mode M1 -current_design
add_core_instance -core top_level -mode M2 -current_design
set_system_mode analysis

// Mode extracted from ASCII file
read_patterns top_levelM1.ascii
read_patterns top_levelM2.ascii
write_patterns …
```

Figure 3-2. Merge Patterns Sets Into One Patterns Set



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

The following example demonstrates the use of the add_core_instances -current_design switch to merge mode M1 of the current design and mode internal of CoreB. In the example, one pattern set is generated at the top level (mode M1 of the top level) to test Sub-Block1, and a second pattern set is generated at the core level of CoreB. The result, as shown in Figure 3-3, is the integration of the two pattern sets, originally generated for mode M1 of the top level and mode internal of CoreB, into the top level.

The first dofile reads in the top level design and writes out the patterns and core description files for mode M1 targeting Sub-Block1.

```
// Invoke tool to generate patterns for mode M1 of the top level
set_context pattern -scan
read_verilog top_level.v                    // Read netlist of top_level

// Mode 1 consists of the Sub-Block1 design; the user must blackbox any other blocks not
// targeted, and apply constraints to access Sub-Block1
set_current_mode M1 -type internal
...
set_system_mode analysis
...
create_patterns
write_patterns top_levelM1.ascii
write_core_description top_levelM1.tcd
```

The second dofile reads in the CoreB design and writes out the patterns and core description files.

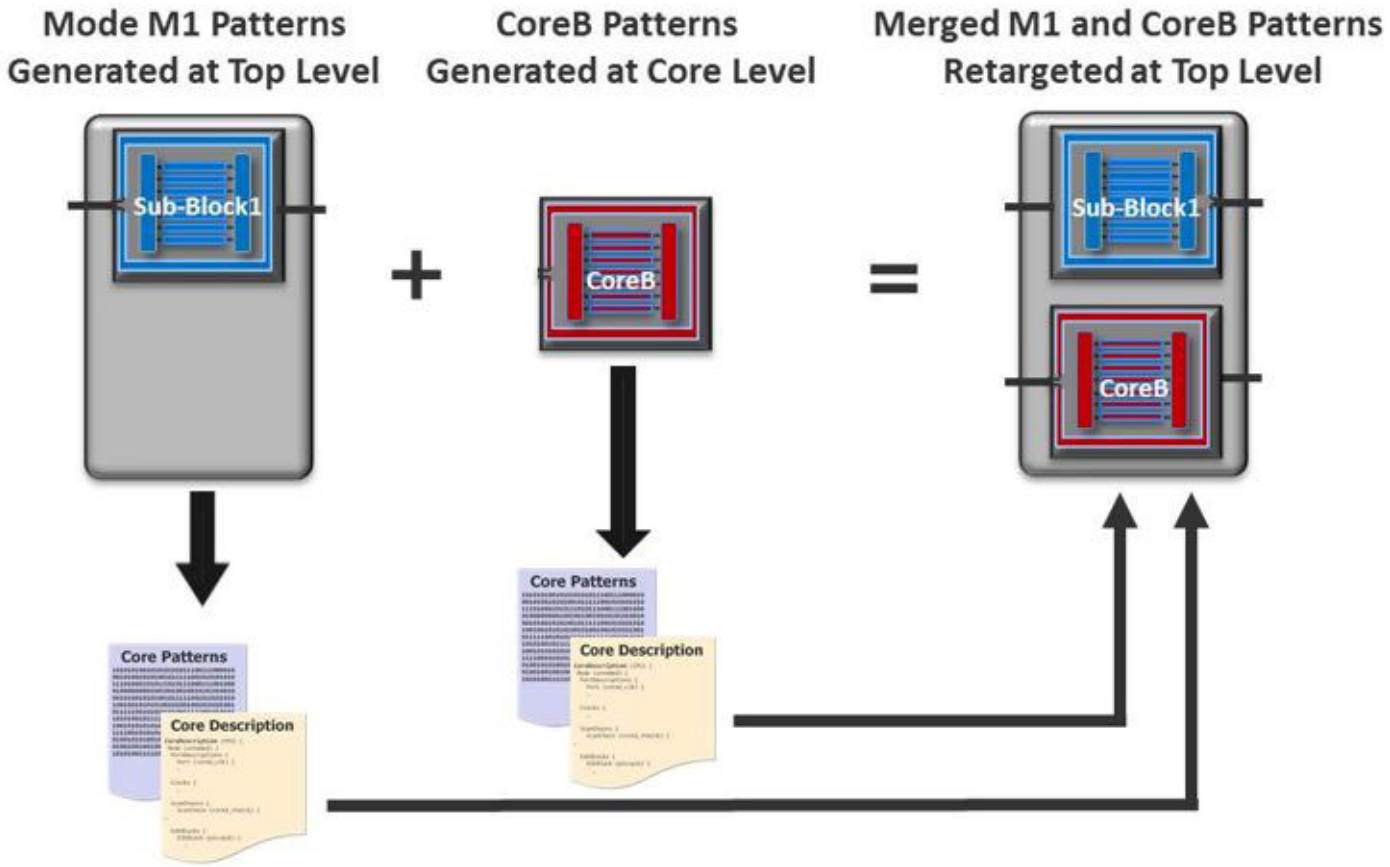
```
set_system_mode setup
delete_design
read_verilog CoreB.v
set_current_mode internal -type internal
...
set_system_mode analysis
create_patterns
write_patterns CoreB.ascii
write_core_description CoreB.tcd
```

The third dofile merges the top level pattern set (mode M1) with the retargeted CoreB patterns.

```
// Invoke tool to retarget patterns to top level
set_context patterns -scan_retargeting
...
read_core_description top_levelM1.tcd
read_core_description CoreB.tcd
add_core_instance -core top_level -mode M1 -current_design
add_core_instance -core CoreB -mode internal -instances  /CoreB_i
...
set_system_mode analysis
read_patterns top_levelM1.ascii
read_patterns CoreB.ascii
write_patterns  ….
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 3-3. Merging Patterns for Top Level and Core at the Top Level



Example 4

The following example shows the four core descriptions in memory that are matched and then used to add core instances:

SETUP> report_core_descriptions

```
Name                  Mode Name       Type
------------------    -----------     -----------
* my_design           retargeting     retargeting
  m1231               internal        internal
  my_module_name_1    internal        internal
  my_module_name_2    internal        internal
  my_module_name_3    internal        internal

Note: * denotes current design
```

The corresponding design modules have the same name as the core descriptions. Table 3-6 shows the mapping between the core descriptions and the design instances:

Table 3-6. Mapping of Core Descriptions and Instances

Core	Instance
m1231	inst_4

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-6. Mapping of Core Descriptions and Instances (cont.)

Core	Instance
my_module_name_1	inst_1
my_module_name_2	inst_2
my_module_name_3	inst_3

The following command adds the core instances:

SETUP> add_core_instances -instance inst_*

```
// Added core instance 'inst_4'.
// Added core instance 'inst_1'.
// Added core instance 'inst_2'.
// Added core instance 'inst_3'.
```

Example 5

The following example shows the four core descriptions in memory that the tool attempts to match and use to add core instances. The matching options, which can be specified and changed with the set_module_matching_options command, default to an empty prefix and a "(_[0-9]+)?" suffix, which means that the tool tries to match the core name against the module name by adding an underscore followed by a digit to it. The name matching mechanism fails to match all of the modules and an error is issued.

SETUP> report_core_descriptions

```
Name                  Mode Name       Type
------------------    -----------     -----------
* my_design           retargeting     retargeting
  m123                internal        internal
  my_module_name      internal        internal
  my_module_name_     internal        internal
  my_module_name_2    internal        internal

Note: * denotes current design
```

The corresponding design modules are:

SETUP> get_modules {m123? my_module_name??}

```
{m1231 my_module_name_1 my_module_name_2 my_module_name_3}
```

Table 3-7 shows the mapping between the core descriptions, modules, and instances:

Table 3-7. Mapping of Core Descriptions With Modules and Instances

Core	Module	Instance
m123	m1231	inst_4
my_module_name	my_module_name_1	inst_1
my_module_name_	my_module_name_3	inst_3

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-7. Mapping of Core Descriptions With Modules and Instances (cont.)

Core	Module	Instance
my_module_name_2	my_module_name_2	inst_2

Attempting to add all core instances in one step produces an error as shown below:

SETUP> add_core_instances -instance inst_*

```
// Error: No core description found to match module 'm1231' of instance
// 'inst_4'. Please use -core to specify the core description to use.
```

Example 6

In this example, there are four core descriptions in memory that can be used to add core instances:

SETUP> report_core_descriptions

```
Name                  Mode Name    Type
-------------------   -----------  -----------
* my_design           retargeting  retargeting
  m1231               internal     internal
  my_module_name_1    internal     internal
  my_module_name_2    internal     internal
  my_module_name_3    internal     internal
```

Note that * denotes current design

The corresponding design modules have the same name as the core descriptions such that the mapping between the core descriptions and the design instances is as follows:

```
Core                  Instance
-------------------   -----------
m1231                 inst_4
my_module_name_1      inst_1
my_module_name_3      inst_3
my_module_name_2      inst_2
```

The design instance cores are added:

SETUP> add_core_instances -instance inst_*

```
//  Added core instance 'inst_4'.
//  Added core instance 'inst_1'.
//  Added core instance 'inst_2'.
//  Added core instance 'inst_3'.
```

Related Topics

delete_core_instances

read_core_descriptions

report_core_descriptions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_core_instance_parameters

report_core_instances

report_core_parameters

set_core_instance_parameters

write_core_description

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_design_partition

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Defines a partition of a design.

Usage

add_design_partition ***partition_name***
{ ***gate_or_instance_set*** | **-clock_domain *clk_domain*** }
[-append]

Description

Creates a named partition of a design for use in other operations such as set_power_control. You specify the logic in a partition by using either the <*gate_or_instance_set* > argument or the "-clock_domain <*clk_domain*>" switch. If you want to define a partition containing a clock domain and additional logic outside that domain, you can run the add_design_partition command repeatedly with the -append switch.

Arguments

- ***partition_name***

  A required string that specifies the name of a partition containing a set of gates.

- ***gate_or_instance_set***

  A required set of strings that specifies one or more gate IDs, gate pin pathnames, instance names, or object specs to include in the partition. An object spec is a repeatable string that specifies design objects to include in the partition.

- **-clock_domain *clk_domain***

  A required switch and string pair that specifies all gates that can be affected by the outputs of the sequential elements after launching the specified clock.

- -append

  An optional switch that adds the set of gates specified by the other arguments to an existing partition. If the partition specified by ***partition_name*** does not already exist, the tool creates a new partition.

Examples

Example 1

This example shows how to create a partition named hot_spot by listing a gate ID and a module:

```
add_design_partition hot_spot 123456 /top/and/OUT /top/ctl_module
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

This example shows how to add the logic in the top/block1/clk1 clock domain to an existing partition named hot_spot:

```
add_design_partition hot_spot -clock_domain /top/block1/clk1 -append
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_dft_clock_enables

Context: dft (with no sub context)

Mode: setup

Prerequisites: The current design must be set with the set_current_design command.

Defines a pin or port of the current design or a port of a sub-module as a clock enable signal that, when activated, disables the clock gating on a clock branch.

Usage

add_dft_clock_enables ***pin_port_net_spec*** **-usage** {**func_en** | **test_en**} [-active_polarity 0 | 1]

Description

These clock enable signals are used to suppress clock gating during the functional mode of the operation, and are reused during memory BIST to suppress clock gaters fanning out to memories. On a clock gater cell, the clock enable signal with -usage func_en corresponds to the pin typically named "en"; it does not correspond to the pin named "te", which is used to disable clock gating during the scan shift mode.

Note

The FE/TE pins of the RTL clock gater are automatically added to the add_dft_clock_enables list. If you want to prevent these connections, use the dont_touch_enables_of_clock_gater attribute.

For example, if you have two or more clocks converge on a gate in your design that you handle for mission mode, Tessent Shell may try to sensitize them in test mode. This results in a clock DRC violation. To avoid this, use the dont_touch_enables_of_clock_gater attribute on the convergent gate. See the "Built-In Attributes" section of "Instance" on page 3668 for details.

If the clock gater is seen to fan out to the reference of a generated clock, the clock gater is disabled by all_test such that the generated clock gets its reference clock in all test modes. The following warning is generated during DRC when this happens:

```
//  Note: The clock gated pin 'cgand1/rtlc1_5/Z' fans out to the reference
//        of a generated clock.
//        Disabling it with the DFT signal '~all_test' as its enable is
//        active low.
```

You can use add_dft_control_points *pin* -dft_signal_source_name all_test explicitly on a clock enable pin as long as the use of the -inverse_dft_signal_source is specified consistently with the active polarity of the clock enable pin. You can use any DFT signal to point to the clock enable pin as long as its default_value_in_all_test option is set to 1.

If you have clock gating cells in your library, you do not need to reference their "en" pins with this command because they are automatically learned. The report_dft_clock_enables command

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reports those automatically if the cell is used at least once in your design. Note that you can also introspect the learned clock gating cells in your design by using the following command:

get_modules -filter "cell_type == clock_gating_and || cell_type == clock_gating_or"

To see the functional enable port, use:

get_ports -of_module $mod -filter "function==func_enable || function==func_enable_inv"

To see the test enable port, use:

get_ports -of_module $mod -filter "function==test_enable || function==test_enable_inv"

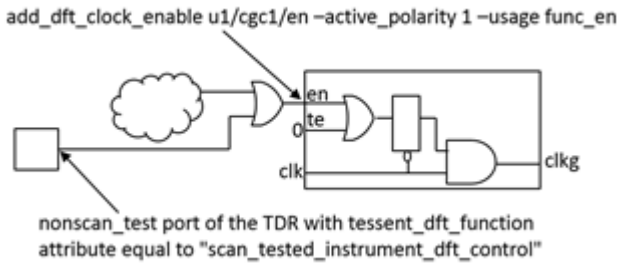
You only need to use the add_dft_clock_enables command if you have clock gating implemented with logic that does not use clock gating cells such as when you are performing memory BIST insertion in RTL mode. The examples in Table 3-8 illustrate how to define clock enable signals for different scenarios.

The clock enable signals you specify are used during DRC when you issue the check_design_rules command in dft context, as long as the "get_dft_specification_requirements -memory_bist" command returns "auto" at the time the check_design_rules command is issued. Clock enable signals that affect the clocks of memories are automatically identified in DRC and disabled when you run the process_dft_specification command in analysis mode.

Table 3-8 illustrates the logic inserted for a clock enable signal with and without a functional source and with both polarities. The source for these examples is either the nonscan or the nonscan_inv port of a special TDR register. A single TDR is used even if you do more than one insertion pass. Clock gaters fanning out to memories tested by memory BIST controllers inserted in a different pass share the special TDR inserted in the first pass.

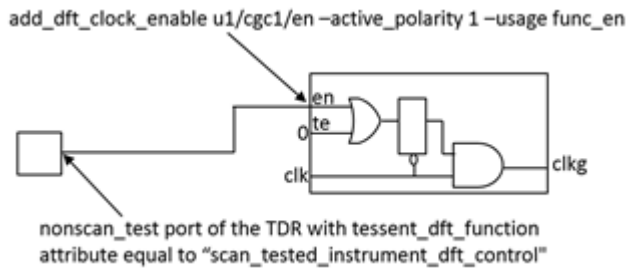
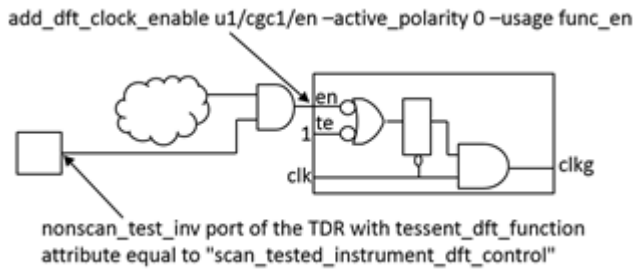
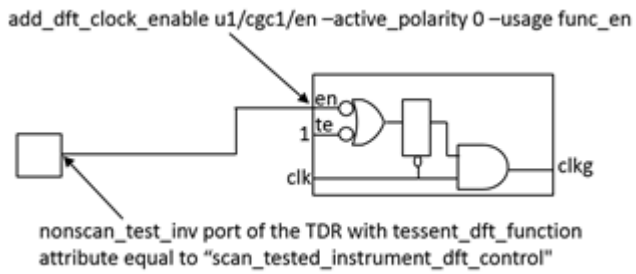
Refer to the Patterns/DftControlSettings property to understand how it is automatically controlled during the execution of the process_patterns_specification command.

Table 3-8. DFT Logic Inserted for Different add_dft_clock_enables Locations

Specification	Resulting DFT
A clock enable with a polarity of 1 and having a functional source prior to the DFT insertion pass	

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-8. DFT Logic Inserted for Different add_dft_clock_enables Locations

Specification	Resulting DFT
A clock enable with a polarity of 1 and having no functional source (it is tied off or floating) prior to the DFT insertion pass	
A clock enable with a polarity of 0 and having a functional source prior to the DFT insertion pass	
A clock enable with a polarity of 0 and having no functional source (it is tied off or floating) prior to the DFT insertion pass	

Arguments

- ***pin_port_net_spec***

  A required string that specifies a list of one or more names of a pin, port, or net object, or a collection containing one or more pin, port, or net objects. If you have an RTL clock gating module, you can specify it on the port of the clock gating module using the commands shown here:

```
set mods [get_modules my_clock_gate*]
add_dft_clock_enables [get_ports en -of_modules $mods] \
  -usage func_en
```

  As previously described, the DRC process automatically identifies the instances that need to be disabled during memory BIST and leaves all other instances untouched. The clock gating cells loaded with the read_cell_library command are automatically learned and do not need to be referenced by this command.

  If the object is a port on the current design and the port connects to the pad_io function of a pad cell, the DFT connection is inserted on the core side of the pad buffer. If the boundary scan cell is already inserted, it is connected on the core side of the boundary scan cell. If

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

there is no boundary scan cell associated to the port, it connects to the from_sji_mux function when it exists, otherwise it connects to the from_pad function.

- **-usage func_en | test_en**

  A required switch and value pair that specifies the usage of the clock enable pins and ports. The func_en value is used for pre-DFT DRC of memory BIST. The test_en value is used for the pre-DFT DRC for logic test.

- -active_polarity 0 | 1

  An optional switch and value pair that specifies the active polarity of the clock enable signal. When unspecified, the polarity defaults to 1. A polarity of 1 means the clock gating cell is not gating the clock when the clock enable signal is high.

Examples

The following example uses the add_dft_clock_enables command to identify the en ports of clock gating modules whose module name starts with "my_clock_gate", and to define the pin on an instance as a clock enable signal with polarity 0. The source of the clock enable signal is a special TDR that is scan tested with the rest of the functional logic. The add_dft_control_points command is used to control the select of an existing clock multiplexer; the add_dft_clock_enables command is not used for that signal because the clock multiplexer must be controlled in scan test mode too.

The report_dft_clock_enables command is used in analysis mode with and without the -post_drc option to show the specified clock enables and those extracted during DRC. The specified ones can exist on pins or ports of the current design or on ports of any sub-module. During DRC, the clock enables specified on ports of sub-modules are mapped to pins. Only the clock enable signals that affect the memory clocking are kept.

In analysis mode, a read-only wrapper in the tcd partition called "Core(<*design_name*>)/DftInfo(<*design_id*>)" contains the ClockEnable signals extracted by DRC. These pins have DFT added to them to disable the clock gating during memory BIST.

```
set_context dft -rtl
# read design command here

set_current_design my_design
set_design_level physical_block
set_dft_specification_requirements -memory_test on

add_clocks clka -period 6.7ns

set mods [get_modules my_clock_gate*]
add_dft_clock_enables [get_ports en -of_modules $mods] -usage func_en
add_dft_clock_enables clock_ctrl/disable -usage func_en -active_polarity 0
add_dft_control_point clock_mux/select

check_design_rules

report_dft_clock_enables
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Dft clock enable for usage 'func_en'
// ====================================
// --------------------------------  --------
//               Node                polarity
// --------------------------------  --------
// Port 'en' of 'my_clock_gate_and'     1
// Port 'en' of 'my_clock_gate_or'      1
// Pin  '/clock_ctrl/disable'           0
```

report_dft_clock_enables -post_drc

```
// Dft clock enable for usage 'func_en'
// ====================================
// ---------------------------------  --------
//               Node                 polarity
// ---------------------------------  --------
// '/clock_ctl/master_clock_gate/en'     1
// '/uart/mem_clk_gate1/en'              1
// '/clock_ctrl/disable'                 0
```

report_config_data Core(my_design)/DfTInfo(rtl)/ClockEnables -partition tcd

```
ClockEnables {
  ClockEnable(/clock_ctl/master_clock_gate/en) {
    active_polarity : 1;
  }
  ClockEnable(/uart/mem_clk_gate1/en) {
    active_polarity : 1;
  }
  ClockEnable(/clock_ctrl/disable) {
    active_polarity : 0;
  }
}
```

Related Topics

add_dft_clock_mux

add_dft_control_points

delete_dft_clock_enables

read_cell_library

report_dft_clock_enables

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_dft_clock_mux

Context: dft (with no sub context)

Mode: setup

Prerequisites: The current design must be set.

Instructs the tool to assume the presence of a clock multiplexer during pre-DFT DRC and to insert the multiplexer during DFT insertion as part of the process_dft_specification command.

Usage

add_dft_clock_mux ***pin_port_net_spec*** **-test_clock_source** ***test_clock_source_spec*** [-dft_signal_source_name *dft_signal_source_name*] [-inverse_dft_signal_source] [-auto_uniquify]

Description

Instructs the tool to assume the presence of a clock multiplexer during pre-DFT DRC and to insert the multiplexer during DFT insertion as part of the process_dft_specification command.

The specified pin, port, or net is intercepted with the 0 input of the multiplexer and the 1 input is connected to the specified test_clock_source_spec. The select of the multiplexer is connected to the specified *dft_signal_source_name* with an optional inversion based on the presence of the -inverse_dft_signal_source switch.

The default *dft_signal_source_name* value is “all_test” which is a DFT signal that is automatically asserted high during all test modes: those modes based on scan and those that are not. You can also define your own custom *dft_signal_source_name*s by using the register_static_dft_signal_names command.

If you need to insert a custom multiplexer where the select is controlled by an arbitrary source, use the intercept_connection command with the -dft_cell_function option specified as “mux”. Those, however, need to be inserted prior to running DRC.

If you are familiar with the LogicVision Memory Bist flow, a clock multiplexer was specified in ETChecker using the lv.InternalClockSource -test_clock_source option. The select was always connected to the ETClockEnable signal. In the Tessent Shell flow, the name of the ETClockEnable signal is “all_test” because it is a test signal that is activated during all test modes, whether they are based on scan or not.

The tool does not return an error if you run the same add_dft_clock_mux twice. You only get an error if the second specification has one or more options that are different from the first specification.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The leaf instance name of the inserted clock mux is <*persistent_clock_cell_prefix*>_<*dft_signal_name*>_clkmux2 (or <*persistent_clock_cell_prefix*>_<*dft_signal_name*>_inv_clkmux2 if -inverse_dft_signal_source is used).

- Specify the <*persistent_clock_cell_prefix*> value using the DftSpecification/ persistent_clock_cell_prefix property. The default value is tessent_persistent_cell.
- Specify the <*dft_signal_name*> value with the -dft_signal_source_name *dft_signal_source_name* switch.

Arguments

- ***pin_port_net_spec***

  A required string argument that specifies the name of a pin, port, or net of the current design or a collection containing one port, pin, or net object of the current design. The insertion of the multiplexer is permitted on both input and output ports and input and output pins. The select is asserted to the default_value_in_all_test value of the *dft_signal_source_name* during pre-DFT DRC, which is 1 when using all_test as the DFT signal source name. All combinations of clock path are verified during extract_icl. The default_value_in_all_test value of the *dft_signal_source_name* is specified with the register_static_dft_signal_names command.

- **-test_clock_source** ***test_clock_source_spec***

  A required switch-value pair that is uses to specify the name of a port, pin, or net of the current design or a collection containing one port, pin, or net object. The specified *test_clock_source_spec* is used to source the 1 input of the multiplexer. The 0 input of the multiplexer remains the original source of the *pin_or_port_spec*.

  If the object is a port on the current design and the port connects to the pad_io function of a pad cell, the DFT connection is inserted on the core side of the pad buffer. If the boundary scan cell is already inserted, it is connected on the core side of the boundary scan cell. If there is no boundary scan cell associated to the port, it connects to the from_sji_mux function when it exists. Otherwise, it connects to the from_pad function.

- -dft_signal_source_name *dft_signal_source_name*

  An optional switch-value pair that is used to specify a string that defines the name of a DFT signal that sources the select for the multiplexer.

  The default *dft_signal_source_name* value is “all_test” which is a DFT signal that is automatically asserted high during all test modes: those modes based on scan and those that are not. You can also define your own custom *dft_signal_source_name* using the register_static_dft_signal_names command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -inverse_dft_signal_source

  An optional Boolean option that specifies an inversion between the source DFT signal and the select port of the multiplexer. The clock multiplexer always needs to reset selecting the 0 input of the multiplexer otherwise the insertion of the multiplexer would affect the functional mode and fail the pre-DFT versus post-DFT formal verification. To satisfy the requirement, you can only refer to a *dft_signal_source_name* that resets to 0 when the -inverse_dft_signal_source option is not specified and you can only refer to a *dft_signal_source_name* that resets to 1 when the -inverse_dft_signal_source is specified.

- -auto_uniquify

  An optional Boolean option that specifies the parent module of the *pin_port_net_spec* is to be auto-uniquified if seen to be inside an instance of a repeated module. If you do not specify this option, you get an error when running the check_design_rules command and you did not specify an add_dft_clock_mux command for all instances of the repeated module. The *test_clock_source_spec* and the *dft_signal_source_name* are permitted to be different for each instance. The error message lists the missing instances making it easy to add them and avoid unification.

Examples

The following example shows the use of three add_dft_clock_mux commands. One is to enable the sharing of a common clock source during memory BIST, a technique often used to minimize the number of clock sources needed during manufacturing test. The second one is to enable bypassing of the PLL based on a user-defined DFT signal source name called pll_bypass. The third one is used to provide an alternative source for the reference of the PLL based on a user-defined DFT signal source name called alt_ref. Notice that during insertion the interception for all multiplexers is done first and then the connections for the test clock sources are performed. This makes sure the test clock source of the green multiplexer is not intercepted by the red interception. Notice that it is not permitted to insert a clock multiplexer in the middle of a balanced clock tree. This is why this example has two declared branch clocks to denote the base of the balanced clock tree feeding the memories. The inserted clock multiplexers can only fanout to the reference of generated clocks and to branch clocks but not to memories or scannable flip-flops directly. If you want to insert a multiplexer without declaring a branch clock, you have to insert it in your design (either manually or with a custom insertion script using the intercept_connection command). You leave the select tied off and then point to it using the add_dft_control_points to make it controllable by IJTAG.

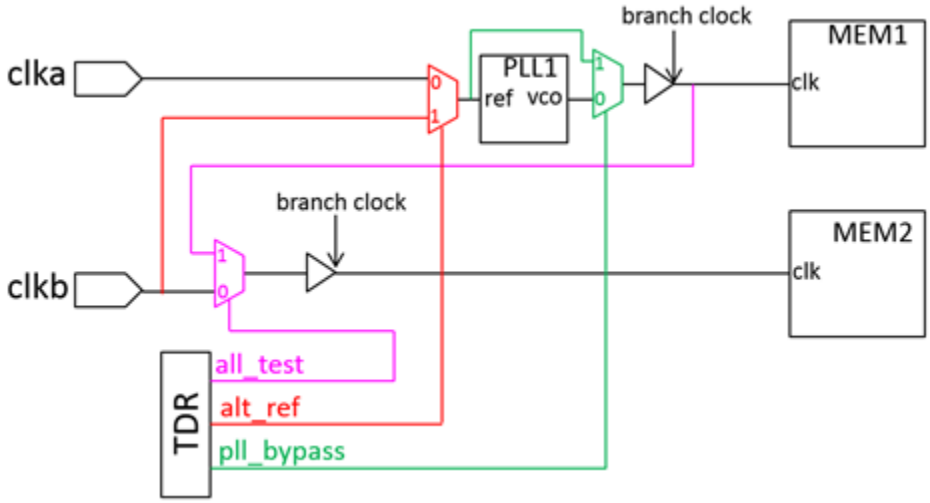
Note how the DftSpecification contains a Tdr wrapper having an Attribute tessent_dft_function set to scan_resource_instrument_dft_control. You cannot delete or modify this Tdr wrapper when you use the add_dft_clock_mux or the add_dft_control_points commands.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
> set_context dft -rtl
> #read design command here
> set_current_design my_design
> set_design_level physical_block
> set_dft_specification_requirements -memory_test on
> add_clocks clka -period 6.7ns
> add_clocks vco_buf/Y -branch
> add_clocks clkb_buf/y -branch
> register_static_dft_signal_names alt_ref pll_bypass
> add_dft_clock_mux clkb -test_clock_source mem1/clk \
    -dft_signal_source_name all_test
> add_dft_clock_mux pll1/vco -test_clock_source pll1/ref \
    -dft_signal_source_name pll_bypass
> add_dft_clock_mux pll1/ref -test_clock_source clkb \
    -dft_signal_source_name alt_ref
> report_dft_clock_mux
// Dft clock muxes// ===============
// ----------  ----------------- -------------------
// Node Test   clock source      Control source name
// ----------  ----------------- -------------------
// 'clkb'      mem1/clk          all_test
// 'pll1/vco'  pll1/ref          pll_bypass
// 'pll1/ref'  clkb              alt_ref


> check_design_rules
> set spec [create_dft_specification]
> report_config_data $spec

DftSpecification(mydesign,rtl) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Sib(sri) {
        Attributes {
          tessent_dft_function : scan_resource_instrument_host;
        }
        Sib(sri_ctrl) {
          Tdr(sri_ctrl) {
            Attributes {
              tessent_dft_function : scan_resource_instrument_dft_control;
            }
          }
        }
      }
    }
  }
}

> process_dft_specification
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 3-4. Example Usage of add_dft_clock_mux



Related Topics

add_dft_clock_enables

delete_dft_clock_muxes

report_dft_clock_muxes

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_dft_control_points

Context: dft (with no sub context)

Mode: setup

Prerequisites: The current design must be set.

Instructs the tool to assume the presence of the DFT control logic during pre-DFT design rule checking, and to insert the controlling logic during DFT insertion as part of the process_dft_specification command.

Usage

add_dft_control_points ***pin_port_net_spec*** [-type *type*] [-inverse_dft_signal_source]
[-dft_signal_source_name *dft_signal_source_name*
[-parent_instance_of_dynamic_dft_signal *parent_instance*]] [-ignore_existing_sources]
[-auto_uniquify] [-allow_editing_below_ijtag_instances]

Description

Instructs the tool to assume the presence of the DFT control logic during pre-DFT design rule checking, and to insert the controlling logic during DFT insertion as part of the process_dft_specification command.

For DFT control points of type “static_dft_control”, when the specified object is a port or an output pin, and when the specified object is an input pin or a net with a functional source (that is, its has_functional_source Pin or Net attribute is true), it is intercepted with the 0 input of a multiplexer and the 1 input gets connected to the static DFT signal source location. The source is inverted when the -inverse_dft_signal_source option is specified. The select of the multiplexer is connected to the “all_test” DFT signal. The logic is a simple OR gate or AND gate when the *dft_signal_source_name* is specified as “all_test”. (To obtain this simple OR/AND logic for a signal name other than “all_test,” use the register_static_dft_signal_names command to define the signal to be of type “logic_test_control”.) When the specified pin or port has no functional source (that is, its has_functional_source attribute is false), it gets directly connected to the DFT signal source with an optional inversion. Refer to Table 3-9 for illustrations of the inserted logic and connections. If the specified node is tied but the value does not match the reset value of the DFT signal, you get an error unless you use the -ignore_existing_sources option.

The default *dft_signal_source_name* value is “all_test”, which is a DFT signal that is automatically asserted high during all test modes: modes based on scan and those that are not. Use report_dft_signal_names to find the built-in static DFT signals. You can also define your own custom DFT signal names using the register_static_dft_signal_names command. For more information about customizing *dft_signal_source_name* and its usage, refer to the register_static_dft_signal_names command section.

If you need to insert a custom gate where the second data input and the select input is controlled by arbitrary sources, use the intercept_connection command with the -dft_cell_function option specified as “mux”, “and” or “or”. Those, however, needs to be inserted prior to running DRC.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If you are familiar with the LogicVision MemoryBIST flow, a control point was specified in ETChecker using the ***lv.ETClockEnable*** and the ***lv.InjectControl -type UserBit*** properties. In the Tessent Shell flow, the ETClockEnable signal is now called “all_test” because it is a test signal that is activated during all test modes, whether they are based on scan or not. A UserBit is now a user-registered DFT signal name.

For DFT control point of type “dynamic_dft_control”, the specified signal is always directly connected to the specified node. You get an error if the specified node has a functional source (that is, its has_functional_source attribute is true) or if it is tied to a 1 unless you use the -ignore_existing_sources switch. In such a case, the existing source is simply replaced by the dynamic DFT signal source.

You no longer get an error if you issue the exact same add_dft_control_point twice. You only get an error if the second specification has one or more options that are different from the first specification.

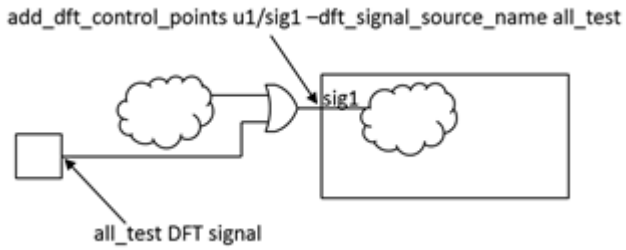
In a hierarchical flow, if you add any dynamic DFT signals in the current design and the current design instantiates some lower designs in which you had added dynamic DFT signals sourced by ports, an “add_dft_control_point -type dynamic_dft_signal” command is inferred to connect them if their "has_functional_source" attribute returns false. When this happens, the tool displays a note similar to the following:

check_design_rules

```
//  Note: Inferred 5 'add_dft_control_points <pin>
//        -type dynamic_dft_control' commands to connect the dynamic DFT
//        signals pins on the child block instances. Use
//        report_dft_control_points to see them.
```

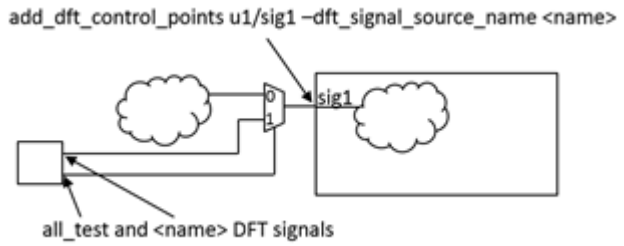
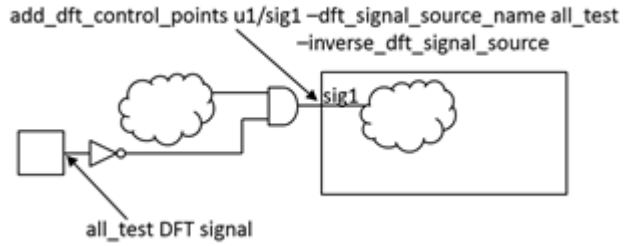
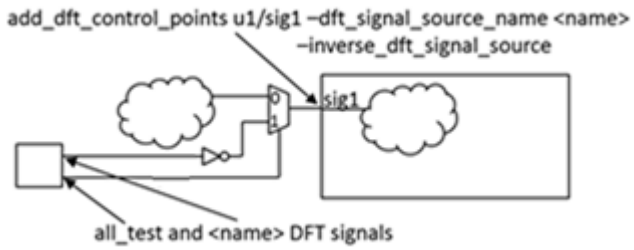
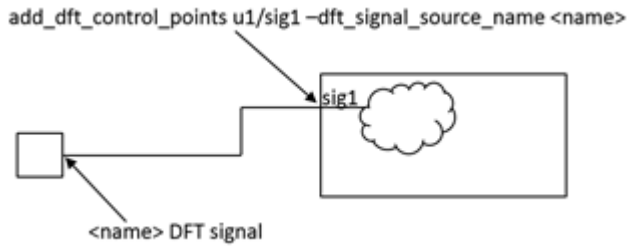
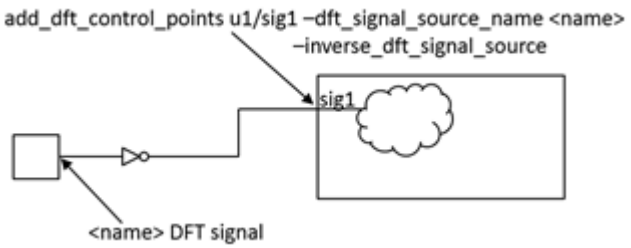
If the scan_en dynamic DFT signal was not added yet and the scan_en DFT signals are seen to be floating at the input of a sub designs, they are tied low as part of running the process_dft_specification command such that the functional and the non-scan test modes work. The scan_en is later connected to the DFT signal “scan_en” once you add it in the current design.

Table 3-9. DFT Logic for add_dft_control_points Nodes

Specification	Resulting DFT
A dft control point with DFT signal source equal to "all_test" or another DFT signal source that is of usage "Logic Test Control". The object has a functional source prior to the DFT insertion.	

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-9. DFT Logic for add_dft_control_points Nodes (cont.)

Specification	Resulting DFT
A dft control point with control source equal to a name other than "all_test" that is also of usage "Global DFT Control". The object has a functional source prior to the DFT insertion.	
A dft control point with an inverted control source equal to “all_test”. The object has a functional source prior to the DFT insertion.	
A dft control point with an inverted control source equal to a name other than “all_test”. The object has a functional source prior to the DFT insertion.	
A dft control point with any DFT signal source name. The object has no functional source (it is tied or floating) prior to the DFT insertion.	
A dft control point with any inverted DFT signal source name. The object has no functional source (it is tied or floating) prior to the DFT insertion.	

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***pin_port_net_spec***

  A required string that specifies a list of one or more names of pin, port, or net object, or a collection containing one or more pin, port, or net objects. The control point is permitted to be specified on both input and output ports and input and output pins. The DFT signal source name "all_test" is asserted active during pre-DFT Memory BIST DRC. The other DFT signal source names are asserted to their respective "default_value_in_all_test" specified when registering them. Refer to the register_static_dft_signal_names command for more information about the behavior of the DFT signals.

  If the object is a port on the current design and the port connects to the pad_io function of a pad cell, the DFT connection is inserted on the core side of the pad buffer. If the boundary scan cell is already inserted, it is connected on the core side of the boundary scan cell. If there is no boundary scan cell associated to the port, it connects to the from_sji_mux function when it exists. Otherwise, it connects to the from_pad function.

- -type static_dft_control | async_set_reset | dynamic_dft_control

  An optional switch-value pair that specifies the kind of DFT control to add. For DFT control point of type static_dft_control, the specified node is connected or intercepted with a multiplexer by the specified DFT signal as shown in Table 3-9. For DFT control of type dynamic_dft_control, the specified node is connected to the specified DFT signal.

  When you set type to async_set_reset, DFT_C9 checks the specified node. The common off value is determined if seen to source asynchronous set or reset pins of one or more scannable flip-flops. An OR gate or an AND gate is inserted to disable the asynchronous set and reset pins during shift mode. An OR gate is used when the active polarity is determined to be 0 and an AND gate is used when the active polarity is determined to be 1. The other input of the GATE is connected to the DFT signal "async_set_reset_dynamic_disable". Refer to Table 3-13 on page 219 in the add_dft_signals command description for more information about this DFT signal. By default, DFT_C9 infers the async_set_reset DFT control points on the nodes found to be controlling sources of asynchronous set or reset pins. Refer to "set_drc_handling dft_c9 -auto_fix" command description to know how to disable the auto fixes.

  If you have issued the add_dft_control_point -type async_set_reset command and set_dft_specification_requirements -logic_test is off, then issuing the check_design_rules command produces the following error:

```
//  Error: You can't have 'add_dft_signals -type async_set_reset'
//  when 'set_dft_specification_requirements -logic_test' is 'off'.
//    Unless DFT_C9 runs, the active polarity of the set or reset
//    nodes cannot be determined.
//    Either add '-logic_test on' to your
//    set_dft_specification_requirements settings or issue
//    delete_dft_control_points {{rst} {ram1/AR[1]}}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -inverse_dft_signal_source

  An optional Boolean option that specifies an inversion between the DFT signal specified by the -dft_signal_source_name option and the ***pin_port_net_spec***.

- -dft_signal_source_name *dft_signal_source_name*

  An optional switch and string pair that defines the name of the DFT signal that is to source the ***pin_port_net_spec*** during test modes.

  The default *dft_signal_source_name* value is "all_test", which is a DFT signal that is automatically asserted high during all test modes: modes based on scan and those that are not. You can also define your own custom *dft_signal_source_names* using the register_static_dft_signal_names command.

- -parent_instance_of_dynamic_dft_signal *parent_instance*

  An optional switch and string pair that specifies the parent_instance of the dynamic DFT signal to use for sourcing the specified DFT control. Use this switch when you define several copies of the same dynamic DFT signal using the "add_dft_signals -parent_instance" command. When you do not specify this switch, the tool uses the version of the DFT signal specified or inferred in the same parent instance as the ScanHost node. If none exist at that location, or when you are not using SSN, the first one added is used. Refer to Example 4 in "ScanHost" on page 4484 for an example where multiple copies of the shift_capture_clock DFT signal are added. The DftSignal(shift_capture_clock) connection value for the OCC slow_clock port automatically selects the closest DFT signals. The same is true for the DftSignal(edt_clock) connection value for the EDT edt_clock port.

- -ignore_existing_sources

  An optional Boolean option that is used to suppress the error that is normally generated in those two situations. For DFT signal of type "static_dft_control", you get an error if the specified node is tied and the tie value is not consistent with the reset value of the DFT signal taking into account if the source is inverted or not. Specifying this switch suppresses the error and enables replacing the tie value with the connection from the DFT signal. Realize that suppressing this error causes the node to have a different value in functional mode as compared to before the connection was made.

  For DFT signal of type "dynamic_dft_control", you get an error if the specified node is tied and the tie value is not consistent with the reset value of the DFT signal taking into account if the source is inverted or not. You also get an error if the node already has a functional source. Specifying this switch suppresses the error and enables replacing the tie value or the functional source with the connection from the DFT signal. Realize that suppressing this error causes the node to have a different value in functional mode as compared to before the connection was made.

- -auto_uniquify

  An optional Boolean option that specifies the parent module of the ***pin_port_net_spec*** is to be auto-uniquified if seen to be inside an instance of a repeated module. If you do not specify this option, you get an error when running the check_design_rules command and you did not specify an add_dft_control_points command for all instances of the repeated

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

module. The *dft_signal_source_name* is permitted to be different for each instance. The error message lists the missing instances making it easy to add them and avoid unification.

- -allow_editing_below_ijtag_instances

  An optional Boolean switch that enables you to point to a pin that is below a module that has a matched ICL module. By default, you are not permitted to point to a pin that is inside an instance with its is_hard_module or created_by_synthesis attribute set to true. When -type is set to static_dft_control, you are also not permitted to point to a pin that is below a module that has a matched ICL module because those DFT inject points can affect the ICL view of the module. If you are sure that the control point does not affect the ICL view, use this switch to relax the check. Otherwise, go back to a step prior to when the ICL was created to insert the DFT control point.

Examples

This example uses the add_dft_control_points to inject a control multiplexer on the select of an existing clock multiplexer. The inserted logic is illustrated in Figure 3-5. The output port "all_test" on the TDR is automatically asserted high at the beginning of the patterns created using process_patterns_specification. You can decide to activate the PLL bypass mode by setting the pll_bypass port active using the Patterns/DftControlSettings wrapper. If you have a programmable PLL with a built-in bypass_en port and ports to control the multiplication factor, you can use this command to control those ports. A more automated way is to create an ICL model for the PLL describing the DataInPorts of the PLL. The create_dft_specification command automatically hooks them up to the ijtag network. You can then use a ProcedureStep to program your PLL. You must use the add_dft_control_points when the injected values are needed by the pre-DFT DRC.

Note how the DftSpecification contains a Tdr wrapper having an Attribute tessent_dft_function set to scan_resource_instrument_dft_control. You cannot delete or modify this Tdr wrapper when you use the add_dft_clock_mux or the add_dft_control_points commands.

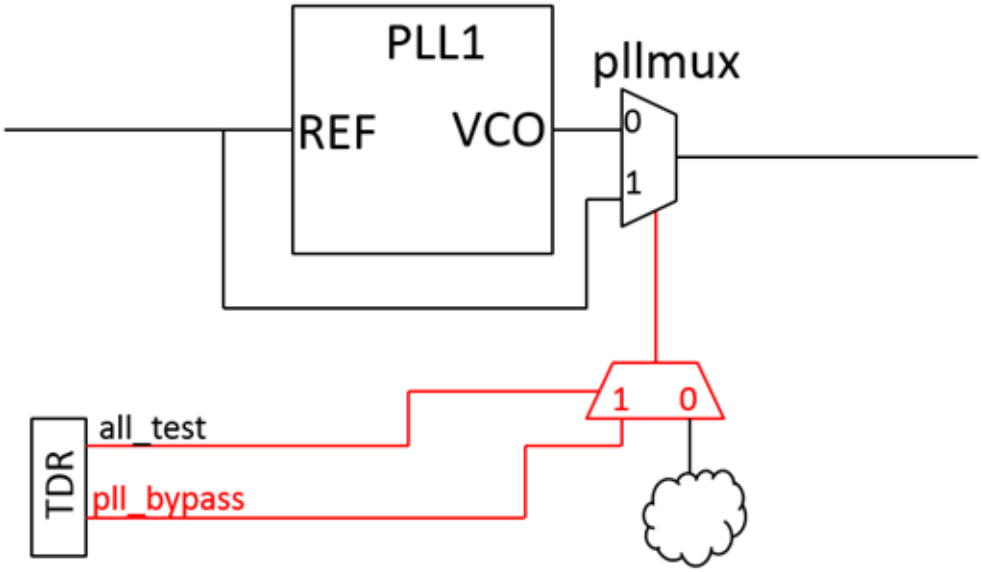
```
set_context dft -rtl

#read design command here

set_current_design my_design
set_design_level physical_block
set_dft_specification_requirements -memory_test on
add_clocks clka -period 6.7ns
add_clocks pllmux/y -reference PLL1/REF -freq_multiplier 10
register_static_dft_signal_names pll_bypass
add_dft_control_points pllmux/s -dft_signal_source_name pll_bypass

check_design_rules

set spec [create_dft_specification]
report_config_data $spec
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(my_design,rtl) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Sib(sri) {
        Attributes {
          tessent_dft_function : scan_resource_instrument_host;
        }
        Sib(sri_ctrl) {
          Tdr(sri_ctrl) {
            Attributes {
              tessent_dft_function : scan_resource_instrument_dft_control;
            }
          }
        }
      }
    }
  }
}
```

process_dft_specification

Figure 3-5. Example Usage of add_dft_control_points Command



Related Topics

add_dft_clock_enables

add_dft_clock_mux

delete_dft_control_points

process_dft_specification

report_dft_control_points

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_dft_modal_connections

Context: dft

Mode: setup

A command used to insert modal connections from specified output data source nodes or to input data destination nodes typically used to connect the EDT channel input and output pins from multiple cores to and from a limited set of input and output ports.

Usage

add_dft_modal_connections
{**-ports** ***port_spec*** | **-auxiliary_data_pins** ***pin_spec*** [-auxiliary_enable_pins *pin_spec*] }
{**-output_data_source_nodes** ***port_pin_spec***
| **-input_data_destination_nodes** ***port_pin_spec***}
{**-enable_dft_signal** ***dft_signal_name*** | **-enable_source_node** ***port_pin_spec***}
[-inverse_enable]
[-pipeline_stages *pipeline_stages* | -nonscan]
[-output_retiming {on | off}]

Description

Use this command to insert modal connections from specified output data source nodes or to input data destination nodes. Once the tool adds all modal connections, it analyzes them and inserts appropriate multiplexing and pipelining logic to implement the specification. You can use this command to connect the EDT channel input and output pins from multiple cores to and from a limited set of input and output ports. Use the report_dft_modal_connections command to find the cumulative effect of your specified add_dft_modal_connections commands and understand the inserted multiplexing logic.

Typically, you have a limited number of top-level ports that you can use to provide data to the EDT channel inputs and observe the EDT channel outputs. This prevents you from running all EDT cores in parallel. Instead, you need to use sets of cores to create multiple top-level access modes so that you do not exceed the top-level resources available in your chip.

The add_dft_modal_connections command is very flexible and enables you to insert the necessary multiplexing and pipelining logic automatically. Figure 3-6 shows that you require multiplexing logic when you specify multiple modal connections on the same output port. The blue logic represents the auxiliary output logic present on the output port. The auxiliary output logic must permit multiplexing between the auxiliary data out values and the functional values and enable the pad output buffers. You can use the BoundaryScan wrapper of the DftSpecification to equip the top-level ports with auxiliary input and output logic. The actual logic inserted is more complicated than the blue logic shown in Figure 3-6 because Boundary Scan also needs control of the ports, and you do not want to cascade two multiplexers along the functional data path. Refer to Figure 13-46 on page 4138 in the AuxiliaryInputOutputPorts section of the BoundaryScan wrapper section for a precise description of this logic.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The circuitry described next is included in modules handling one bit at a time. The tool instantiates these modules in the parent instance of the auxiliary data pin they handle. If you used the BoundaryScan AuxiliaryInputOutputPorts wrapper to insert your auxiliary data logic, the auxiliary data logic is located in the logical grouping module associated with a given port. You have the flexibility to place the logical grouping modules where you want. The modal connection modules follow the logical grouping module and are instantiated in the same parent instance module. If you use the -auxiliary_data_pins option, the modules are instantiated in the parent instance of those pins. For the modules sometimes created at the destination of auxiliary input connections and illustrated in Figure 3-8, the parent instance of the "input_data_destination_nodes" is where they are instantiated.

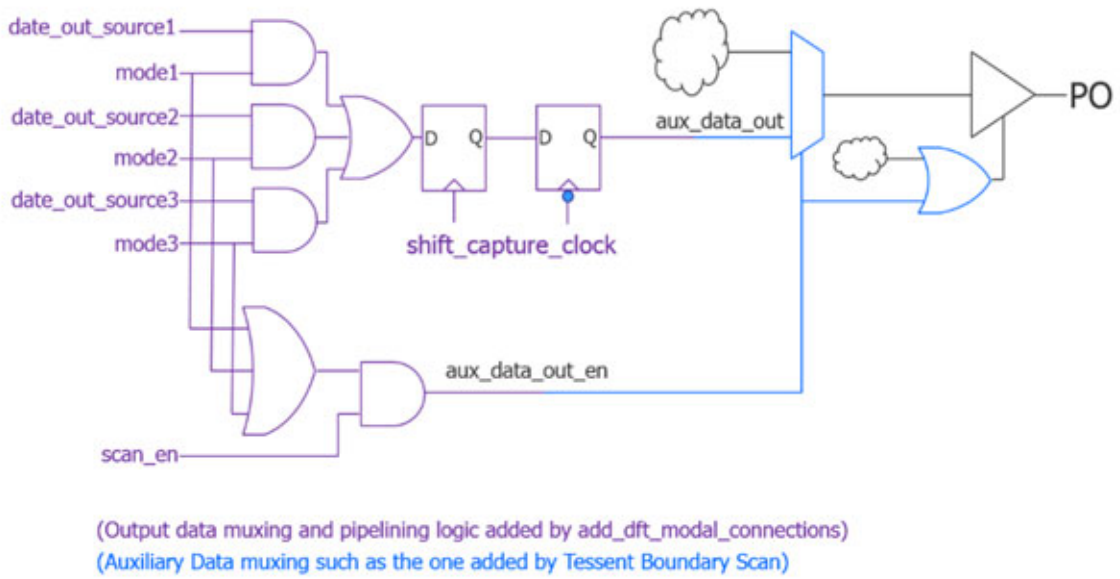
Note

The tool creates modal connection modules to handle one bit at a time. The same module is instantiated multiple times if multiple connections have the same muxing and pipelining requirement unless the reuse_modules_when_possible property found in the DftSpecification wrapper defaults is off or is set to off, in which case the tool creates one module per bit even if the content is identical to others.

The multiplexing is achieved by ANDing each data source with its associated enable source and ORing the AND gates together. This kind of multiplexing is very efficient for routing as there is no select priority logic, and the layout tool can split the OR gate to minimize the congestion. The enable sources are also ORed together to activate the auxiliary output data path. The auxiliary data output enable signal is gated by scan_en. This permits the functional output logic to be observable during the capture cycles either by the boundary scan cell if you are using the boundary scan during scan test (refer to the max_segment_length_for_logictest property in the BoundaryScan wrapper to enable that) or by the PO itself.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 3-6. Multiplexing and Pipelining Logic for Output Connections



Multiplexing logic is not needed at the source for the auxiliary input data connections. As shown in Figure 3-7, the enable sources still need to be ORed together to drive the auxiliary input enable pin of the port. The auxiliary input enable is used to gate off the auxiliary data so that it does not follow the port value during functional mode. When the port is an inout port, the enable is also used to turn off the output driver so that the port can be used to supply input data. Finally, if pipelining was requested, the auxiliary data is pipelined and the re-timed pipelined sources are used to make the connections to the specified destinations.

Figure 3-8 shows that multiplexing logic may be required at the destination nodes. For example, if an ELT core is re-used in multiple configurations, the edt_channel_in ports may have different sources in different modes. The circuit shown in Figure 3-8 is used to implement the necessary multiplexing. When such multiplexing is required, the report_dft_modal_connections command includes an “Input destinations” section showing the multiplexing on the destination pins.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 3-7. Enable Logic and Pipelining for Input Connections

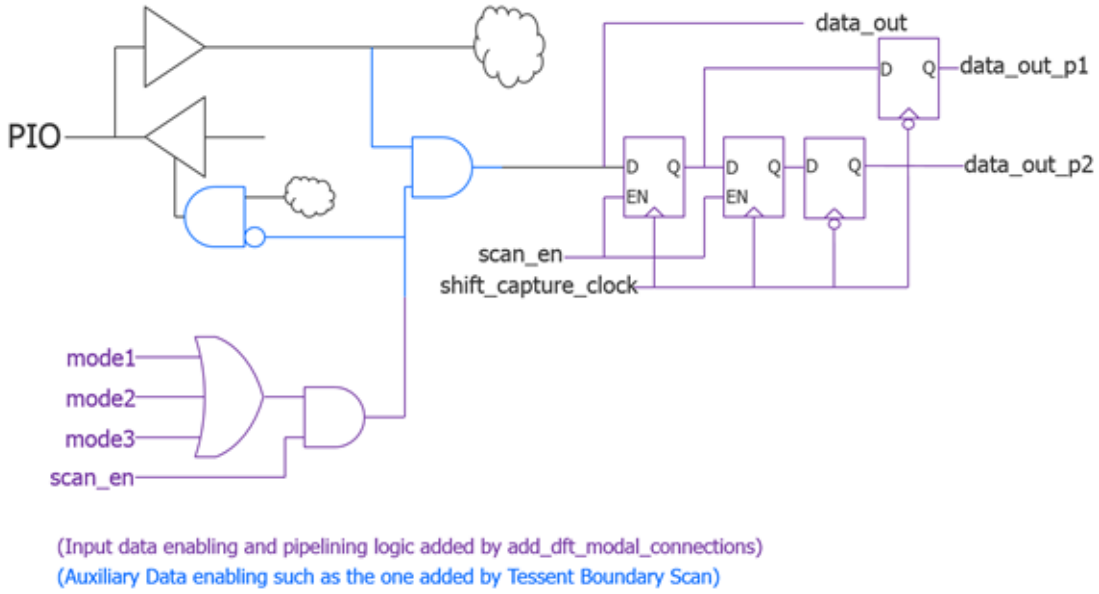
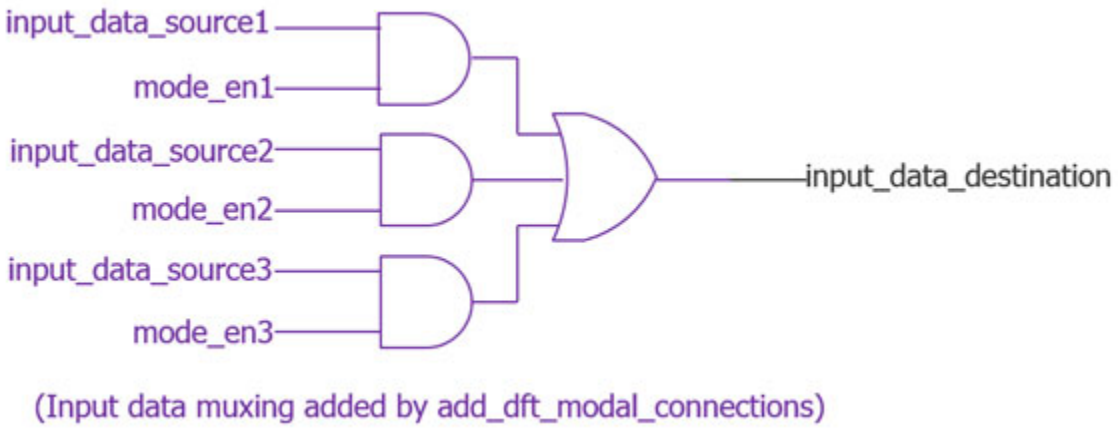


Figure 3-8. Multiplexing for Destinations of Modal Connections



The pipelining flip-flops are clocked using the shift_capture_clock that you define the source using the "add_dft_signals shift_capture_clock" command. If you are using the add_dft_modal_connections command to provide source or observation of data coming from non-scan test modes, you specify the -nonscan switch, and you do not specify the -pipeline_stages switch.

Output connections sharing the same destination port share the same output retiming setting. When you turn off output retiming for at least one connection, it is turned off for all in the group. The tool reports a warning in this case.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

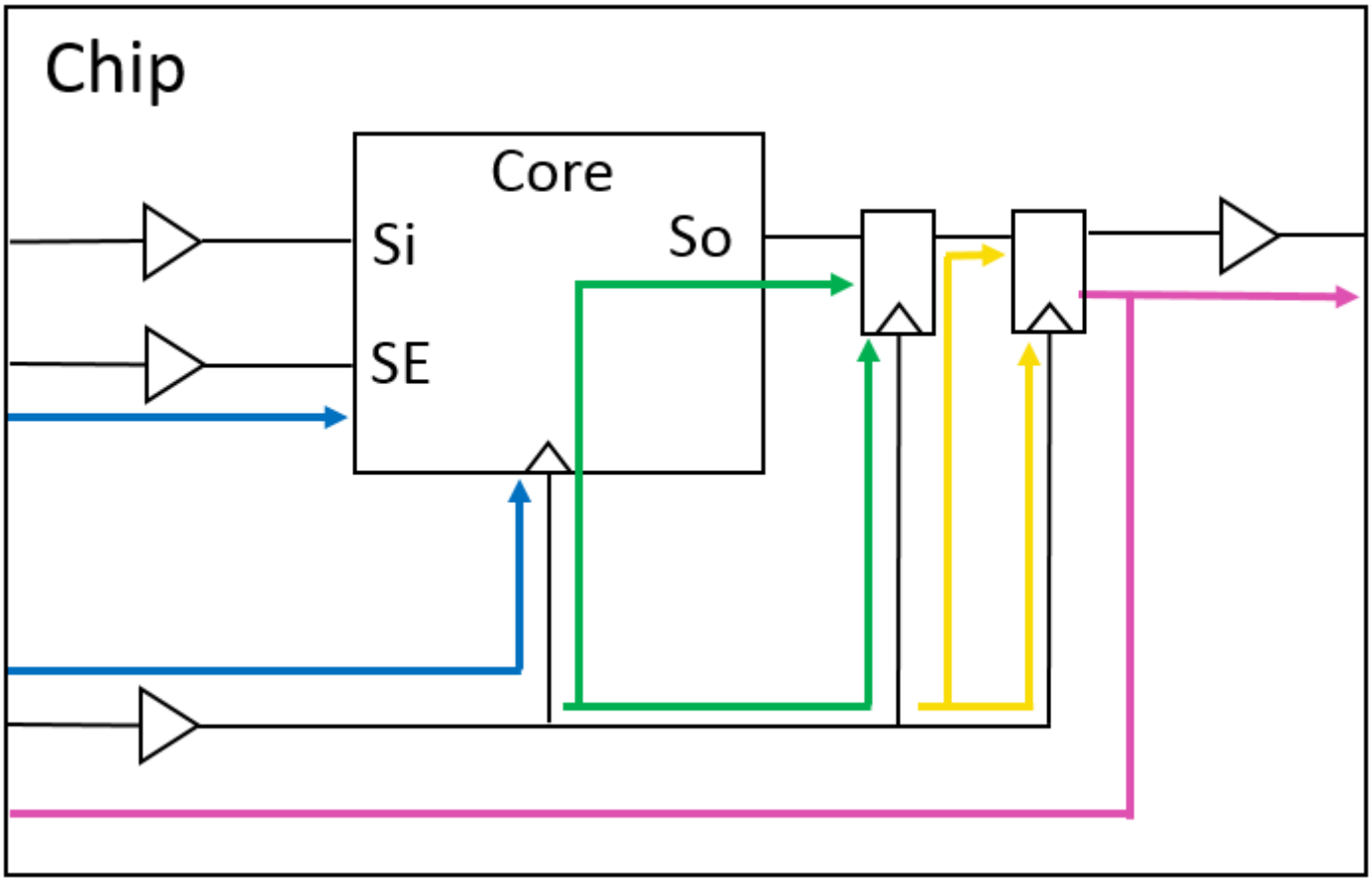
The report_dft_modal_connections command reports “(not retimed - forced)” for modes with an inferred output retiming setting.

Maximizing the Shift Rate of the logictest Mode

The shift clock rate of the scan-based test modes is often limited by the loop timing arc (illustrated by the magenta arrow in Figure 3-9) starting from the shift clock input, propagating through the clock tree to reach the last flop on the chain and exiting the chip through an output buffer. The majority of the delay is spent in the clock tree. To reduce this timing arc, you can use a few pipelining flip-flops in series for the scan-out path and control the clock tree synthesis tool to place the last flop as close as possible to the base of the clock tree. If the clock tree delay is larger than one clock period, use a clock point that is mid-point along the clock tree for the middle pipelining flop. You then have a full cycle minus the clock delay difference between the first and the middle pipelining flip-flop (illustrated by the green arcs), and again a full cycle minus the clock delay difference between the middle and the last pipelining flip-flop (illustrated by the yellow arcs). The add_dft_modal_connections command enables you to insert up to five pipelining flops in series. You then need to provide instruction to your Clock tree synthesis to not balance the clock inputs to the last stages in order to enable stepping down toward the output port. Pipelining is typically not needed on the input direction because the clock path delay largely compensates for the delay in the data paths (illustrated by the blue arcs).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 3-9. Timing Arcs for Scan In and Out Paths



Arguments

- **-ports *port_spec***

  A switch used to specify a list of one or more port names or a collection of one or more port objects that are to be used as data source or destination. The direction of the data connection is controlled by the use of one of the mandatory and mutually exclusive -output_data_source_nodes and -input_data_destination_nodes switches. When the -output_data_source_nodes switch is used, the specified ports must either be existing or to-be-created output ports or existing output or inout ports equipped with auxiliary data logic. When design level, as specified by the set_design_level command is “chip”, ports cannot be created and must already be connected to a pad equipped with auxiliary data output logic. At the physical or sub-block level, you may specify to-be-created ports or existing ports having no existing functional sources. Refer to the AuxiliaryInputOutputPorts section of the BoundaryScan or EmbeddedBoundaryScan sections to learn how to automatically equip ports with auxiliary input or output data logic.

  When you use Tessent Boundary Scan to insert the auxiliary input and output data logic, the location of the auxiliary data and enable pins is described in the Instrument Dictionary created during boundary scan insertion. You can then use the -ports switch to point to the ports and the mapping to the auxiliary data and enable pins is done automatically. If you

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

implemented the auxiliary data logic manually or with a third party tool, you have two options. You can either describe the mapping between the top level ports and the auxiliary data pins using a Tcl dictionary called "::auxiliary_data_dict" or by pointing explicitly to the auxiliary data and enable pins using the -auxiliary_data_pins/-auxiliary_enable_pins switches for each invocation of the add_dft_modal_connections commands.

The format of the "::auxiliary_data_dict" dictionary is the following:

```
<port_name> {
  auxiliary_output_pin         <pin_name>
  auxiliary_output_enable_pin  <pin_name>
  auxiliary_input_pin          <pin_name>
  auxiliary_input_enable_pin   <pin_name>
}
... #repeat for all ports
```

You can use the Tcl "dict" command to populate this dictionary once so that you can then make your DFT modal connection using the -ports switch instead of having to explicitly point to the auxiliary data and enable pins explicitly each time. Ex:

```
dict set ::auxiliary_data_dict gpio[1] auxiliary_input_pin u1/aux_in
```

The enable pin entries in the dictionary are optional. If you do not list them, the add_dft_modal_connections command only performs the data multiplexing and assumes you have already handled the enabling of the auxiliary data path.

- **-auxiliary_data_pins *pin_spec***

  A switch used to specify the location of the auxiliary input or output data pins. The *pin_spec* value is a list of one or more names of pin objects or a collection containing one or more pin objects. It refers to auxiliary input data pins when the -input_data_destination_nodes switch is used. It refers to auxiliary output data pins when the -output_data_source_nodes switch is used. If you have created the auxiliary data logic with Tessent Boundary Scan using the AuxiliaryInputOutputPorts wrapper of the BoundaryScan or the EmbeddedBoundaryScan wrappers, or if you have described your auxiliary data and enable pins in the Tcl dictionary "::auxiliary_data_dict" described above in the -ports switch section, you do not need to use the -auxiliary_data_pins switch. Instead., you use the -ports switch and the mapping to the auxiliary data and enable pins are extracted automatically.

  If you have the already connected the auxiliary enable pins and you only want the add_dft_modal_connections command to handle the data muxing, point to the auxiliary data pin using the -auxiliary_data_pins switch and omit specifying the -auxiliary_enable_pins switch.

- -auxiliary_enable_pins *pin_spec*

  A switch used to specify the location of the auxiliary input or output enable pins. The *pin_spec* value is a list of one or more names of pin objects or a collection containing one or more pin objects. It refers to auxiliary input enable pins when the -input_data_destination_nodes switch is used. It refers to auxiliary output enable pins when the -output_data_source_nodes switch is used. Refer to the description of the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

-auxiliary_data_pins switch above to know when you need to use it and how the location of the auxiliary enable pins are inferred automatically when using the -ports switch.

- **-output_data_source_nodes *port_pin_spec***

  A switch that is mutually exclusive with the -input_data_destination_nodes switch but required when the -input_data_destination_nodes switch is not specified. You use this switch to specify a list of one or more names of port or pin objects or to specify a collection containing one or more port or pin objects that sources the modal connection. The number of pins or ports specified in the *port_pin_spec* must match the number of elements specified with the -ports or -auxiliary_data_pins switches. If the port or the auxiliary output pin is already connected to the source needed in the given mode, use the symbol "*" as the *port_pin_spec* value, and the current source is preserved for that mode. Refer to Example 1 for an example of its usage.

  Note

  When you specify the -output_data_source_nodes switch and the -output_retiming switch is set to off, the tool removes the negative-edge retiming register in all modal multiplexer instances connected to the specified source nodes.

- **-input_data_destination_nodes *port_pin_spec***

  A switch that is mutually exclusive with the -output_data_source_nodes switch but required when the -output_data_source_nodes switch is not specified. It is used to specify a list of one or more names of port or pin objects or to specify a collection containing one or more port or pin objects that are the destination of the modal connections. The number of pins or ports specified in the *port_pin_spec* must match the number of elements specified with the -ports or -auxiliary_data_pins switches. If the port or the auxiliary input pin is already connected to the needed destination in the given mode, use the symbol "*" as the *port_pin_spec* value and the current destination is preserved for that mode even if pipelining is inserted. Refer to Example 1 for an example of its usage.

  Note

  When you specify the -input_data_destination_nodes switch and the -output_retiming switch is set to off, the tool connects the specified destination node to the data pin driven by a non-retimed pipeline located in a Modal Data In instance.

- **-enable_dft_signal *dft_signal_name***

  A switch that is mutually exclusive with the -enable_source_node switch but required when the -enable_source_node switch is not specified. It specifies the name of a DFT signal that is to serve as the source to enable the modal connection. When the -nonscan switch is not specified, the DFT signal must be of type "scan_mode". When the -noscan switch is used, the DFT signal must be of type "global_dft_control". Refer to the add_dft_signals command for more information about DFT signal types. If the specified *dft_signal_name* is not already present or added, it is added automatically if it is a registered DFT signal name. Use report_dft_signals to find the list of registered DFT signal names. Refer to the register_static_dft_signal_names command description to register your own.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-enable_source_node *port_pin_spec***

  A switch that is mutually exclusive with the -enable_dft_signal switch but required when the -enable_dft_signal switch is not specified. It specifies the name of a pin or port object or a collection containing a port or pin object. If you do not use a DFT signal to control the modal connections, you are responsible to provide the appropriate test_setup proc when using the modal connections during scan or scan_retargeting DRC.

  If you used a DFT signal instead, enabling the mode during the DRC is as easy as calling the set_static_dft_signal_values command.

- -inverse_enable

  An optional switch that is used to specify that the modal connection is to be enabled when the specified enable DFT signal or source node is low instead of high. You cannot use this option when making a connection involving an auxiliary_enable_pin because all auxiliary enable pins must be driven low during functional mode; otherwise, the functional mode will be broken. Also, you can only use this switch when there is only one other modal connection made to the port and this other connection is enabled by the same enable source but without the inversion.

- -pipeline_stages *pipeline_stages*

  An optional switch used to specify an integer to request between 1 to 5 levels of pipelining stages along the modal DFT connection. If you broadcast one input port to multiple destinations in a given mode, the number of pipelining stages must be equal in all broadcasted connections.

  Note

  When using the -pipeline_stages switch, the tool automatically sets the attribute is_non_scannable to true for the dft_modal cells with a pipeline flop that it inserts. It also sets the attribute is_non_scannable_reason to imported_from_icl for those cells.

- -nonscan

  An optional switch used to specify that the modal connection is for a non scan-based test mode. The DFT signal scan_en is not used to gate off the auxiliary enable pin when those modes are used.

- -output_retiming <u>on</u> | off

  An optional switch to specify if the generated modal connection logic has a negative-edge flip-flop at the end of its pipeline. The default is on.

  Note

  The -output_retiming switch has no effect when -pipeline_stages is not specified or set to 0. The tool reports a warning when you use the -output_retiming switch without -pipeline_stages.

  ```
  //  Warning: -output_retiming switch has no effect when
  -pipeline_stages is not specified or set to 0.
  ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example has two identical instances of a module called “corea”. The gpio[3:0] were equipped with auxiliary input and output logic during Boundary Scan implementation using this syntax:

```
set spec [create_dft_spec]
read_config_data -in ${spec}/BoundaryScan -from_string {
  AuxiliaryInputOutputPorts {
    auxiliary_input_ports  : gpio[1:0];
    auxiliary_output_ports  : gpio[3:2];
  }
}
```

When it is time to insert the top level EDT controller, the channels are connected normally to the auxiliary input and output pins of the ports. The get_auxiliary_pins command is used to get the auxiliary data pins created by the Boundary Scan step. The use of the “*” symbol in the edt mode commands specifies to preserve the connections that were done when the top level EDT controller was instantiated. The retargeting1 and retargeting2 modes each retarget the edt mode corea_i1 and corea_i2 through the gpio[2:0] ports. The retargeting3 mode broadcasts the gpio[1:0] auxiliary data inputs to both instances and uses gpio[2] for the EDT output of corea_i1 and gpio[3] for corea_i2.

```
set mode edt_mode
add_dft_modal_connections -ports gpio[1:0] \
   -input_data_destination_nodes * \
   -enable_dft_signal $mode
add_dft_modal_connections -ports gpio[2] \
   -output_data_source_nodes     * \
   -enable_dft_signal $mode

set mode retargeting1_mode
add_dft_modal_connections -ports gpio[1:0] \
   -input_data_destination_nodes corea_i1/edt_channels_in[1:0] \
   -enable_dft_signal $mode
add_dft_modal_connections -ports gpio[2] \
   -output_data_source_nodes corea_i1/edt_channels_out[0] \
   -enable_dft_signal $mode

set mode retargeting2_mode
add_dft_modal_connections -ports gpio[1:0] \
   -input_data_destination_nodes corea_i2/edt_channels_in[1:0] \
   -enable_dft_signal  $mode
add_dft_modal_connections -ports gpio[2] \
   -output_data_source_nodes     corea_i2/edt_channels_out[0] \
   -enable_dft_signal $mode
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set mode retargeting3_mode
add_dft_modal_connections -ports gpio[1:0] \
   -input_data_destination_nodes corea_i1/edt_channels_in[1:0] \
   -enable_dft_signal $mode
add_dft_modal_connections -ports gpio[2] \
  -output_data_source_nodes     corea_i1/edt_channels_out[0] \
  -enable_dft_signal $mode
add_dft_modal_connections -ports gpio[1:0] \
  -input_data_destination_nodes corea_i2/edt_channels_in[1:0] \
  -enable_dft_signal $mode
add_dft_modal_connections -ports gpio[3] \
   -output_data_source_nodes     corea_i2/edt_channels_out[0] \
   -enable_dft_signal $mode

check_design_rules
set spec [create_dft_spec -sri_sib_list {occ edt}]
read_config_data -in_wrapper $spec -from_string {
 Occ {
   ijtag_host_interface : Sib(occ);
   Controller(clka) {
     clock_intercept_node : clka_buf/A;
   }
 }
 Edt {
   ijtag_host_interface : Sib(edt);
   Controller(c1) {
     longest_chain_range                : 10, 50;
     scan_chain_count                   : 40;
     input_channel_count                : 2;
     output_channel_count               : 1;
     connect_bscan_segments_to_lsb_chains : on;
     Connections {
       EdtChannelsIn(1) {
       }
       EdtChannelsIn(2) {
       }
       EdtChannelsOut(1) {
         Pipelinestage {
         }
       }
     }
   }
 }
}

set_config_value port_pin_name \
 -in $spec/Edt/Controller(c1)/Connections/EdtChannelsIn(1) \
 [get_single_name [get_auxiliary_pins gpio[0] -direction input]]

set_config_value port_pin_name \
 -in $spec/Edt/Controller(c1)/Connections/EdtChannelsIn(2) \
 [get_single_name [get_auxiliary_pins gpio[1] -direction input]]

set_config_value port_pin_name \
 -in $spec/Edt/Controller(c1)/Connections/EdtChannelsOut(1) \
 [get_single_name [get_auxiliary_pins gpio[2] -direction output]]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Following is the output created by the report_dft_modal_connections when running on the above example:

```
DFT Modal Connections
=====================

Modes
-----
edt_mode          : DFT signal with usage scan_mode(unwrapped)
  Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
  Output connection (1):
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
retargeting1_mode : DFT signal with usage scan_mode(retargeting)
  Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
  Output connection (1):
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
retargeting2_mode : DFT signal with usage scan_mode(retargeting)
  Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
  Output connection (1):
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
retargeting3_mode : DFT signal with usage scan_mode(retargeting)
  Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
  Output connections (2):
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
    Port 'gpio[3]' through auxiliary output pin 'DEF_inst/gpio_3_AuxOut'

Input connections
-----------------
1) Port 'gpio[1]' through auxiliary input pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_1_AuxIn'
   edt_mode          :<Existing destination>
   retargeting1_mode :    corea_i1/edt_channels_in[1]
   retargeting2_mode :    corea_i2/edt_channels_in[1]
   retargeting3_mode :    corea_i1/edt_channels_in[1]
                     :    corea_i2/edt_channels_in[1]
2) Port 'gpio[0]' through auxiliary input pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_0_AuxIn'
   edt_mode          :    <Existing destination>
   retargeting1_mode :    corea_i1/edt_channels_in[0]
   retargeting2_mode :    corea_i2/edt_channels_in[0]
   retargeting3_mode :    corea_i1/edt_channels_in[0]
                     :    corea_i2/edt_channels_in[0]

Output connections
------------------
1) Port 'gpio[2]' through auxiliary output pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_2_AuxOut'
   edt_mode          :    <Existing source>
   retargeting1_mode :    corea_i1/edt_channels_out[0]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
   retargeting2_mode :    corea_i2/edt_channels_out[0]
   retargeting3_mode :    corea_i1/edt_channels_out[0]
2) Port 'gpio[3]' through auxiliary output pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_3_AuxOut'
   retargeting3_mode :    corea_i2/edt_channels_out[0]
```

Example 2

The following example shows three input modal connections connected to the same input; one has the output retiming disabled.

```
//  command: add_dft_modal_connections -ports gpio[7]\
               -input_data_destination_nodes sub1/ib1[0] \
  -enable_dft_signal retargeting1_mode \
               -pipeline_stages 2
//  command: add_dft_modal_connections -ports gpio[7] \
               -input_data_destination_nodes sub1/ib1[1] \
               -enable_dft_signal retargeting2_mode \
               -pipeline_stages 2  -output_retiming off
//  command: add_dft_modal_connections -ports gpio[7] \
               -input_data_destination_nodes sub1/ib1[2] \
               -enable_dft_signal retargeting3_mode \
               -pipeline_stages 3
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_dft_signals

Context: dft

Mode: setup

Adds static and dynamic DFT signals used to control various aspect of the DFT logic.

Usage

add_dft_signals {***name*** ...}
[-source_nodes *pin_port_spec* [-make_ijtag_port] [-associated_ports *port_spec*] |
-create_from_other_signals [-parent_instance *parent_instance*]
[-leaf_instance_name *leaf_instance_name*] |
-create_with_tdr [-disable_for_functional_safety {on | off | auto}]]
[-add_hold_scan_cell {on | off | auto} [-hold_scan_cell_disable *dft_signal*]]

Description

A command used to request the addition of static and dynamic DFT signals used to control various aspect of the DFT logic. There are several pre-registered DFT signals that you can use to control the circuit during its multiple test modes.

- Table 3-10, Table 3-11, and Table 3-12 list the pre-registered static DFT signals.
- Table 3-13 on page 219 lists the pre-registered dynamic DFT signals.

Use the report_dft_signal_names command to report the list of registered DFT signals. Use the register_static_dft_signal_names command to register additional signals.

The static DFT signals are separated into the following groups:

- **global_dft_control** — Contains signals used to control global resources such as clocking and power management circuitry.
- **logic_test_control** — Defines the signals used to configure various aspect of the circuit during logic test modes.
- **scan_mode** — Defines the signals used to configure the scan chains into the various scan configurations during logic test modes. When using scan_mode signals (edt_mode, multi_mode, and so on), only one mode is active at a time.

The logic_test_control signals include ltest_en, which is the signal used to place the circuit in logic test mode. This signal is systematically set to 1 when running any logic test mode. The int_ltest_en and the ext_ltest_en signals are used to place the circuit in internal or external mode. A scan mode that includes both the wrapper and internal scan chains of a core can be operated in internal mode by keeping int_ltest_en to 1. The same scan chain configuration can be turned into an unwrapped mode by forcing int_ltest_en to 0. Refer to the set_static_dft_signal_values command for more information about how to set them.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The memory_bypass_en signal is another example of a logic_test_control signal. When set to 1, the tool runs ATPG on a given scan mode with the memory bypass logic enabled. This configuration is used to cover the majority of the faults. When set to a 0, the tool keeps the memory active during scan test to cover the memory port faults using multiple load patterns.

The following tables describe each signal and how to use them.

DFT Signals and IJTAG

By default, the static DFT signals are implemented with a special IJTAG TDR. When you specify a static DFT signal, the tool automatically creates the signal on a TDR. If the DFT signal was added in a previous insertion pass, it is automatically reused.

When the Sib(sti) is already present and its input that are to be connected to a DFT signal still have the has_functional_source attribute = 0, then the tool makes the port connections as the signals are added using this command. Refer to Figure 13-18 on page 3983 and Figure 13-19 on page 3984 for more information about the Sib(sti) and its interaction with logic test. Refer to "How the IjtagNetwork Specification is Created" on page 551 for further information on Sib(sti).

If you have your own IJTAG node to source these signals, you can add attributes in the DataOutPort wrapper of its ICL description to identify them; the signals are automatically recognized as existing DFT signals. Refer to the tessent_dft_signal_* attributes in "icl_port" on page 3784 for a description of the ICL Port Attributes that you must use to identify DFT signals on your IJTAG node. "Example 2" on page 231 shows an example of such an ICL module.

DFT Signal Creation

The DFT signals are created when running the process_dft_specification command. You use create_dft_specification to create the DftSpecification.

Creating DFT Signals With -create_with_tdr

When creating static DFT signals with -create_with_tdr, the IjtagNetwork wrapper is present and contains a Tdr(sri_ctrl) wrapper with the tessent_dft_function attribute set to "scan_resource_instrument_dft_control". Do not add or delete this wrapper. The wrapper's content is auto adjusted by the tool to match the list of static DFT controls you specified.

Note

The create_dft_specification command adds the Tdr(sri_ctrl) wrapper under a SIB wrapper Sib(sri_ctrl) so the tool does not need to scan the TDR every time a lower block is accessed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The add_dft_signals command can only be called after you have run the set_design_level command, unless the design level is already imported by the read_design command. The design_level affects the default behavior of the command as described in the note below:

Note

When the design_level is set to sub_block using the set_design_level command, the static DFT signals default to be created as input ports by the name of the DFT signal. Sub blocks are by definition processed independently at the RTL level but synthesized as part of their parent physical block. The DFT signals created at the boundary of the sub-block are connected to the DFT signal by the same name added in the parent physical block. Typically, you do not add the int_ltest_en and ext_ltest_en DFT signals inside a sub-block, as only physical blocks can be wrapped cores. If for some reason you need to add the int_ltest_en or ext_ltest_en in a sub-block, by default, they are created as ports on the sub-block and connected to the respective DFT signal at the parent level. This means that you are not able to assert int_ltest_en on in the parent and off inside the child sub_block. If you need to do that, use the -create_with_tdr switch to provide an independent source inside the sub-block module.

If you are only adding dynamic DFT signals, then you must run the process_dft_specification command even if the DftSpecification contains no IjtagNetwork wrapper. The tool creates the missing ports and exports the DFT signals into the *.tcd* file located in the dft_inserted_designs directory such that the signals are automatically imported in a subsequent Tessent Shell invocation for the future DFT insertion passes and for ATPG setup.

The DFT signals you add in one insertion pass are automatically imported in the subsequent DFT insertion passes. If the location of the DFT signal is no longer present, the tool issues a warning telling you that the DFT signal could not be imported. For example, if you added edt_clock with -create_from_other_signals in pass1, a clock gater is added to create the DFT signal. If you synthesized the RTL and forgot to preserve the cells with the special persistent_cell_prefix value, the clock gater may have changed names, and it is no longer possible to find the signal.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-10. Pre-Registered Static Global DFT Control Signal Names

Static DFT Signal Name	Usage and Default Implementation - Global DFT Control
all_test	Usage: Global DFT Control reset_value : 0 default_value_in_all_test : 1 value_in_pre_scan_drc: 1 A global DFT control signal that is set high during all tests including all logic test and non-logic test modes such as MemoryBIST. This signal is used by default when using the add_dft_control_points command and the -dft_signal_source_name switch is not specified. The signal is also used to control the select of the multiplexers inserted when the add_dft_control_points command refers to a pin that already has a functional source. Refer to Table 3-9 on page 179 in the add_dft_control_points command description for more details. The all_test signal is special in that the signal is automatically activated during all test modes. The create_patterns_specification command automatically sets it to on in all blocks where it exists. The set_static_dft_signal_values command automatically infers it high whenever another DFT signal is set in a given block. Because the all_test signal is used to force open clock gaters in the fan-in of the reference of generated clocks, it is forced high during ICL extraction such that clocks can be traced through those clock gaters. The ICL ends up not describing the presence of the clock gater along the clock path, but because it is opened by all_test, it can be assumed that the all_test signal is guaranteed to be active during all test modes because of the automation mentioned previously. If you want to control a clock gater with a different DFT signal that you want to control manually, the tool cannot force it high during ICL extraction as the ICL model becomes invalid when the signal is not asserted high. If you want to use a different DFT signal anyway, the clock gater must be modeled in the ICL file. Example 3 of the extract_icl reference page describes how to do that.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-10. Pre-Registered Static Global DFT Control Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation - Global DFT Control
bscan_clamp_enable	Usage: Global DFT Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: X A global DFT control signal used to force the Boundary Scan wire called select_jtag_output to 1. This enables pre-loading the boundary scan cells with a set of values and forcing those values on the output during any test. The connection between the Boundary Scan interface module and this signal is automatic as soon as the command adds the DFT signal.
bscan_input_isolation_enable	Usage: Global DFT Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: X A global DFT control signal used to force the select_jtag_input Boundary Scan wire to 1. This enables using the boundary scan cells on input and inout ports as dedicated wrapper cells and isolates the ATPG mode from the primary inputs. The BoundaryScan patterns cover the testing of the input and inout ports. The bscan_input_isolation_enable DFT signal is automatically set to 1 when a DFT signal of usage scan_mode and type internal is set to 1 with the set_static_dft_signal_values command. See Example 1 in the EmbeddedBoundaryScan/HostBScanInterface reference section for an example where you would use this DFT signal.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-10. Pre-Registered Static Global DFT Control Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation - Global DFT Control
output_pad_disable	Usage: Global DFT Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: X A global DFT control signal used to force the output pad buffers to high-z using the force_disable wire of the Boundary Scan. This enables forcing the output pads to high-z during any test, and saves on power. The connection between the Boundary Scan interface module and this signal is automatic when the command adds the DFT signal. **Note:** If you add this signal and you have inserted BoundaryScan or EmbeddedBoundaryScan with the max_segment_length_for_logictest property not equal to off, the tool automatically connects the output_pad_disable DFT signal to the boundary scan interface module, and forces to_force_disable high when asserted. Use the "set_static_dft_signal_values output_pad_disable 1" command to tri-state the output pad buffers during logictest modes. The tool automatically infers output_pad_disable to 1 when you use set_static_dft_signal_values to set a DFT signal with usage scan_mode and its type is internal. You can choose to force it on explicitly, even in a scan mode of type external, if you declared the mode type as internal using the set_current_mode command.
dft_signal_disable	Usage: Global DFT Control reset_value : 1 default_value_in_all_test : 0 value_in_pre_scan_drc: 0 A global DFT control signal that is set low during all tests, including all logic test and non-logic test modes such as MemoryBIST. This signal provides functional-mode fault tolerance in the TDR register hosting static DFT signals. The tool creates it in the Tessent IP when you request that static DFT signals interacting with functional logic be disabled during the functional mode of design operation. Make this request with the set_dft_signals_options or add_dft_signals commands. The tool adds the dft_signal_disable signal automatically when you implement the functional safety solutions as described in the Tessent Shell User's Manual. Refer to "set_dft_signals_options" on page 2464 for further information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-10. Pre-Registered Static Global DFT Control Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation - Global DFT Control
occ_kill_clock_en	Usage: Global DFT Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: 0 A DFT signal used to drive the kill_clock_en port on the OCC module. Refer to the kill_clock_mode property description in the OCC wrapper.
ssn_en	Usage: Global DFT Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: X A global DFT control signal that is used to activate the SSN data path. The typical usage of this DFT signal is to activate the Auxiliary data path when the SSN data paths share functional ports at the top level. You can use this DFT signal to sensitize any logic you may have along the SSN data path such as power isolation cells. This DFT signal is automatically asserted high during ICL extraction and the DFT signal is automatically written to 1 when ever the SSN network is used. You don't need to add this signal explicitly. It will be inferred with -create_from_tdr during process_dft_specification if it is needed.
tck_occ_en	Usage: Global DFT Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: 0 A global DFT control signal that is used to enable the mini-OCC present inside the Sib(sti) node. The signal is automatically set to 1 when you invoke the add_core_instances command on the Sib(sti) instances.
tck_select	Usage: Global DFT Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: 0 A global DFT control signal that is used to control the multiplexer used to inject TCK at the base of the clock trees. This signal is not needed when using the Tessent OCC as it is already equipped with a TCK multiplexer controlled by a local TDR bit.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-10. Pre-Registered Static Global DFT Control Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation - Global DFT Control
tristate_enable_nc_test	Usage: Global DFT Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: 0 A global DFT control signal that switches the Boundary Scan interface module to support the use of the update and force disable control signals in support of the tristate_enable_non_contacted test (refer to "RunTest" on page 4586). This input is available if the tristate_non_contacted_test_support property in the BoundaryScan wrapper is set to "on."
tristate_enable_nc_test_type	Usage: Global DFT Control reset_value : 0 local : 0 global: 1 A global DFT control signal that switches the behavior selected by tristate_enable_nc_test between the two NonContactedPinOptions/TristateEnableOptions/test_path values (local and global). This input is available if the tristate_non_contacted_test_support property in the BoundaryScan wrapper is set to "on."

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-11. Pre-Registered Static Logic Test Control Signal Names

Static DFT Signal Name	Usage and Default Implementation - Logic Test Control
async_set_reset_static_disable	Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: 0 A logic test control signal that is used to completely disable all asynchronous set and reset signals during logic test modes. This signal is combined with scan_en to create the dynamic DFT signal called async_set_reset_dynamic_signal as described in Table 3-13 on page 219. Normally, the asynchronous set and reset signals are only disabled when scan_en is high. The state of the functional circuit determines if the reset and set pins are active or not during capture cycles. Setting the async_set_reset_static_disable signal to high keeps the set and reset pins gated off even during the capture cycle. It is normally set low but can be set high using the set_static_dft_signal_values command.
capture_per_cycle_static_en	Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: X A logic test control signal that enables observation scan observe points in the hybrid TK/LBIST flow. Observation scan uses observe test points that are monitored during every shift cycle as opposed to traditional observe points that are monitored only in capture. The capture_per_cycle_static_en signal controls these test points.
control_test_point_en	Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: X A logic test control signal that is used to enable the control test point.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-11. Pre-Registered Static Logic Test Control Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation - Logic Test Control
ext_ltest_en	Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: X A logic test control signal that is used to isolate the output ports of a core from the internal logic. In dft -scan context, it is used with the dynamic DFT signal called scan_en to control the isolation in external modes.
ext_lbist_en	Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: X A logic test control signal that enables you to reuse the internal OCC or a hybrid EDT from the core for the LBIST of its parent physical region. This signal is required for internal OCC control only if a clock of a child wrapped core is sourced by an embedded PLL.
int_ltest_en	Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: X A logic test control signal that is used to isolate the input ports of a core from the internal logic. In dft -scan context, it is used with the dynamic DFT signal called scan_en to control the isolation in external modes. **Note:** If you add this signal at the top level and you have inserted BoundaryScan with the max_segment_length_for_logictest property not equal to off, the int_ltest_en DFT signal is automatically connected to the boundary scan interface module and force to_select_jtag_input high when asserted. Then, you must use the set_static_dft_signal_values int_ltest_en 1 command to use the boundary scan chain and isolate the core from the primary input ports.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-11. Pre-Registered Static Logic Test Control Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation - Logic Test Control
lbist_en	Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: 0 A logic test control signal that is used to enable the LogicBIST controller. When not present, the LogicBIST controller contains an internal TDR to enable the LogicBIST mode. This DFT signal is mandatory when you specify the dynamic DFT signal lbist_async_set_reset_dynamic_enable. Refer to the description in Table 3-13.
lbist_low_power_shift_en	Usage: Logic Test Control reset_value : 0 default_value_in_all_test: 0 value_in_pre_scan_drc: 0 A logic test control that activates low-power shift.
lbist_self_test_en	Usage: Logic Test Control reset_value : 0 default_value_in_all_test: 0 value_in_pre_scan_drc: 0 A logic test control that activates Hybrid EDT/LBIST self-test mode.
ltest_en	Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: 1 A logic test control signal that is used to enable the logic test mode. This signal is force high during all logic test modes.
mcp_bounding_en	Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: X A logic test control signal that is used to enable the X-bounding of false paths and multicycle paths.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-11. Pre-Registered Static Logic Test Control Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation - Logic Test Control
memory_bypass_en	Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: 1 A logic test control signal that is used to enable the isolation around the memories. This signal is set to 1 by default during logic test but can be set low using the set_static_dft_signal_values command.
observe_test_point_en	Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: X A logic test control signal that is used to enable the observation test points.
se_pipeline_en	Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: 0 A logic test control signal that enables the pipelining of the scan enable signal to perform launch off shift (LOS) at-speed test. If you specify this signal, scan insertion automatically pipelines the scan enable signal. However if you subsequently run the "set_scan_signals -se_pipeline_enable *pathname*" command, you must explicitly run the set_scan_insertion_options command with the "-se_pipeline_count 1" switch to implement scan enable pipelining.
wrapper_toggle_en	Usage: Logic Test Control reset_value : 0 default_value_in_all_test : 0 value_in_pre_scan_drc: X A logic test control signal that is used to toggle the type of feedback (direct or inverting) in holding wrapper cells created with the "programmable_hold" capture window behavior. This DFT signal is OR-ed by default with a holding scan cell, giving ATPG the ability to request that wrapper cells hold or toggle.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-11. Pre-Registered Static Logic Test Control Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation - Logic Test Control
x_bounding_en	Usage: Logic Test Control reset_value: 0 default_value_in_all_test: 0 value_in_pre_scan_drc: X A logic test control signal that is used to enable the X-bounding circuitry caused by X sources.

Table 3-12. Pre-Registered Static Scan Mode Signal Names

Static DFT Signal Name	Usage and Default Implementation - Scan Mode
controller_chain_mode	Usage: Scan Mode reset_value : 0 scan_mode_type: unwrapped value_in_pre_scan_drc: X A scan mode signal that is used to enable the controller chain mode scan configuration. You can use this scan mode to control the controller chain mode when not using wrapper chains because the scan_mode_type of this signal is unwrapped. Refer to the description of the set_static_dft_signal_values command to understand the importance of using scan mode DFT signals with the accurate scan_mode_type values. In dft -scan context, this signal is automatically associated with the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together. Refer to "Controller Chain Mode" in the *Hybrid TK/LBIST Flow User's Manual* for details.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-12. Pre-Registered Static Scan Mode Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation - Scan Mode
edt_mode	Usage: Scan Mode reset_value : 0 scan_mode_type: unwrapped value_in_pre_scan_drc: X A scan mode signal that is used to enable the EDT scan configuration. You can use this scan mode to control the edt mode when not using wrapper chains because the scan_mode_type of this signal is unwrapped. When using wrapper chains, you use int_edt_mode or int_mode instead. Refer to the description of the set_static_dft_signal_values command to understand the importance of using scan mode DFT signals with the accurate scan_mode_type values. In dft -scan context, this signal is automatically associated with the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-12. Pre-Registered Static Scan Mode Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation - Scan Mode
ext_edt_mode	Usage: Scan Mode reset_value : 0 scan_mode_type: external value_in_pre_scan_drc: X A scan mode signal that is used to enable the external EDT scan mode when using wrapper chains. This DFT signal has scan_mode_type equal to external and is used to configure the external EDT scan configuration of a wrapped core. Refer to the description of the set_static_dft_signal_values command to understand the importance of using scan mode DFT signals with the accurate scan_mode_type values. Typically, the wrapper chains are not driven by an EDT controller but it is an option that can be used. If you are using the multi- and single chain bypass mode built in the EDT controller, you do not need this DFT signal. Instead you use the ext_mode DFT signal. You only use this signal if you have an EDT to drive your wrapper chain and you are implementing the external single and multi-chain bypass mode at the scan insertion stage. In dft -scan context, this signal is automatically associated with the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.
ext_mode	Usage: Scan Mode reset_value : 0 scan_mode_type: external value_in_pre_scan_drc: X A scan mode signal that is used to enable the external scan mode when using wrapper chains. This DFT signal has scan_mode_type equal to external and is used to configure the external scan configuration of a wrapped core. Refer to the description of the set_static_dft_signal_values command to understand the importance of using scan mode DFT signals with the accurate scan_mode_type values. In dft -scan context, this signal is automatically associated with the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-12. Pre-Registered Static Scan Mode Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation - Scan Mode
ext_multi_mode	Usage: Scan Mode reset_value : 0 scan_mode_type: external value_in_pre_scan_drc: X A scan mode signal that is used to enable the external multi-chain scan mode when using wrapper chains. Typically, the wrapper chain are not driven by an EDT controller, but it is an option that can be used. If you are using the multi- and single chain bypass mode built in the EDT controller, you do not need this DFT signal. Instead you use the ext_mode DFT signal. You only use this signal if you have an EDT controller to drive your wrapper chain and you are implementing the external single and multi-chain bypass mode at the scan insertion stage. In dft -scan context, this signal is automatically associated with the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.
ext_single_mode	Usage: Scan Mode reset_value : 0 scan_mode_type: external value_in_pre_scan_drc: X A scan mode signal that is used to enable the external single chain scan mode when using wrapper chains. Typically, the wrapper chain are not driven by an EDT controller, but it is an option you can use. If you are using the multi- and single chain bypass mode built in the EDT controller, you do not need this DFT signal. Instead you use the ext_mode DFT signal. You only use this signal if you have an EDT to drive your wrapper chains, and you are implementing the single and multi-chain bypass mode at the scan insertion stage. In dft -scan context, this signal is automatically associated with the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-12. Pre-Registered Static Scan Mode Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation - Scan Mode
int_edt_mode	Usage: Scan Mode reset_value : 0 scan_mode_type: internal value_in_pre_scan_drc: X A scan mode signal that is used to enable the EDT scan mode when using wrapper chains because the scan_mode_type of this signal is internal. Refer to the description of the set_static_dft_signal_values command to understand the importance of using scan mode DFT signals with the accurate scan_mode_type values. If you are using the multi- and single chain bypass mode built in the EDT controller, you do not need this DFT signal. Instead you use the int_mode DFT signal. You only use this signal if you are implementing the single and multi-chain bypass mode at the scan insertion stage. In dft -scan context, this signal is automatically associated with the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.
int_mode	Usage: Scan Mode reset_value : 0 scan_mode_type: internal value_in_pre_scan_drc: X A scan mode signal that is used to enable the internal mode when using wrapper chains and you are using the multi and single chain bypass mode built in the EDT controller. This DFT signal has scan_mode_type equal to internal and is used to configure the internal scan configuration of a wrapped core. Refer to the description of the set_static_dft_signal_values command to understand the importance of using scan mode DFT signals with the accurate scan_mode_type values. If you are implementing the single and multi chain bypass mode at the scan insertion stage, use the int_edt_mode, int_multi_mode, and int_single_mode signals. In dft -scan context, this signal is automatically associated with the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-12. Pre-Registered Static Scan Mode Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation - Scan Mode
int_multi_mode	Usage: Scan Mode reset_value : 0 scan_mode_type: internal value_in_pre_scan_drc: X A scan mode signal that is used to enable the internal multi-chain scan mode when using wrapper chains. If you are using the multi- and single chain bypass mode built in the EDT controller, you do not need this DFT signal. Instead you use the int_mode DFT signal. This DFT signal has scan_mode_type equal to internal and is thus to be used to configure the internal multi-chain scan configuration of a wrapped core. Refer to the description of the set_static_dft_signal_values command to understand the importance of using scan mode DFT signals with the accurate scan_mode_type values. You only use this signal if you are implementing the single and multi-chain bypass mode at the scan insertion stage. In dft -scan context, this signal is automatically associated with the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-12. Pre-Registered Static Scan Mode Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation - Scan Mode
int_single_mode	Usage: Scan Mode reset_value : 0 scan_mode_type: internal value_in_pre_scan_drc: X A scan mode signal that is used to enable the internal single chain scan mode when using wrapper chains. This DFT signal has scan_mode_type equal to internal and is used to configure the internal single chain scan configuration of a wrapped core. Refer to the description of the set_static_dft_signal_values command to understand the importance of using scan mode DFT signals with the accurate scan_mode_type values. If you are using the multi- and single chain bypass mode built in the EDT controller, you do not need this DFT signal. Instead you use the int_mode DFT signal. You only use this signal if you are implementing the single and multi-chain bypass mode at the scan insertion stage. In dft -scan context, this signal is automatically associated with the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.
multi_mode	Usage: Scan Mode reset_value : 0 scan_mode_type: unwrapped value_in_pre_scan_drc: X A scan mode signal that is used to enable the multi-chain bypass scan configuration. You use this scan mode when not using wrapper chains because the scan_mode_type of this signal is unwrapped. When using wrapper chains, you use int_multi_mode instead. Refer to the description of the set_static_dft_signal_values command to understand the importance of using scan mode DFT signals with the accurate scan_mode_type values. If you are using the bypass mode built in the EDT controller, you do not need this DFT signal. You only use this signal if you are implementing the multi-chain bypass mode at the scan insertion stage. In dft -scan context, this signal is automatically associated with the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-12. Pre-Registered Static Scan Mode Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation - Scan Mode
promoted_cells_mode	Usage: Scan Mode reset_value : 0 scan_mode_type: unwrapped value_in_pre_scan_drc: X A scan mode signal that is used to enable the promoted cells scan mode. You use this mode when you have OCCs driven by embedded PLLs inside a wrapped core and this core is instantiated below another wrapped core. In that case, you simply add this DFT signal in the lower level wrapped core during OCC insertion and use "add_scan_mode promoted_cells_mode" during scan insertion. Chains belonging to OCCs driven by embedded PLLs are automatically included in this mode. They are also automatically included in all scan modes of type external. At the next level up, when doing scan insertion in the parent core, the chain associated with the promoted_cells_mode in the child cores is automatically promoted to belong to the wrapper chains of the parent core so that the OCC is accessible from the top level using the wrapper chains of its child wrapped cores.
retargeting#_mode = 1, 2, ...,8	Usage: Scan Mode reset_value : 0 scan_mode_type: external value_in_pre_scan_drc: X A set of scan mode signals used to configure the retargeting modes that make use of the internal scan modes of child blocks. For example: retargeting1_mode can be used to provide scan channel data to the internal scan mode of core_a, core_b, and core_c, and retargeting2_mode can be used to provide scan channel data to the internal scan mode of core_d, and core_e. If you have more than 8 retargeting combinations at a given level, you simply need to register more scan mode signals of type retargeting using the register_static_dft_signal_names command. In dft -scan context, this signal is automatically associated with the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-12. Pre-Registered Static Scan Mode Signal Names (cont.)

Static DFT Signal Name	Usage and Default Implementation - Scan Mode
single_mode	Usage: Scan Mode reset_value : 0 scan_mode_type: unwrapped value_in_pre_scan_drc: X A scan mode signal that is used to enable the single chain bypass scan mode. You can use this scan mode when not using wrapper chains because the scan_mode_type of this signal is unwrapped. Refer to the description of the set_static_dft_signal_values command to understand the importance of using scan mode DFT signals with the accurate scan_mode_type values. If you are using the single chain bypass mode built in the EDT controller, you do not need this DFT signal. You only use this signal if you are implementing the single chain bypass mode at the scan insertion stage. In dft -scan context, this signal is automatically associated with the added scan mode when the name of the scan mode matches the name of the DFT signal. When you name the scan mode differently from the DFT signal name, use the -enable_dft_signal <dft_signal_name> switch to associate them together.

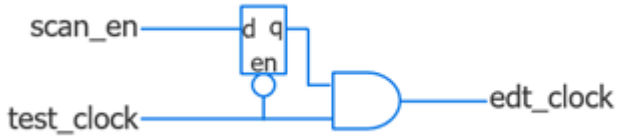
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-13. Pre-Registered Dynamic DFT Signal Names

Dynamic DFT Signal Name	Usage and Default Implementation
async_set_reset_dynamic_disable	A DFT signal that is used to disable the asynchronous set and reset pins of scannable flip-flops during shift mode. This signal is high during the load_unload cycles and is permitted to return to low in the capture cycles. When using the -create_from_other_signals switch, it is created by ORing the scan_en and the async_set_reset_static_disable DFT signals. The DFT signals scan_en and async_set_reset_static_disable must be added before the async_set_reset_dynamic_disable DFT signal can be added using the -create_from_other_signals switch. If you want to create this DFT signal without creating the async_set_reset_static_disable DFT signal, use the -source_node switch and map the async_set_reset_dynamic_disable DFT signal on the same node you have defined the scan_en DFT signal. By default, the async_set_reset_static_disable signal is set to 0 such that the async_set_reset_dynamic_disable signal follows scan_en. However, the set and reset pins can be disabled systematically by setting the async_set_reset_static_disable signal to 1 using the set_static_dft_signal_values command. You can choose to set the async_set_reset_static_disable signal to avoid D1 violations or to reduce the amount of care bits needed by ATPG to disable the set and reset signals to observe the fan-in of the scannable flip-flops. You must, however, generate some patterns with the async_set_reset_static_disable signal set to 0 to gain coverage of the reset circuitry.
capture_per_cycle_dynamic_en	A DFT signal that enables observation scan observe points in the LBIST-ready blocks. This signal is sourced from the LBIST controller's output at the level where the controller is placed.

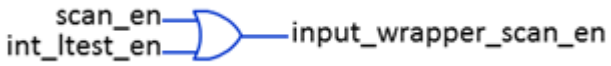
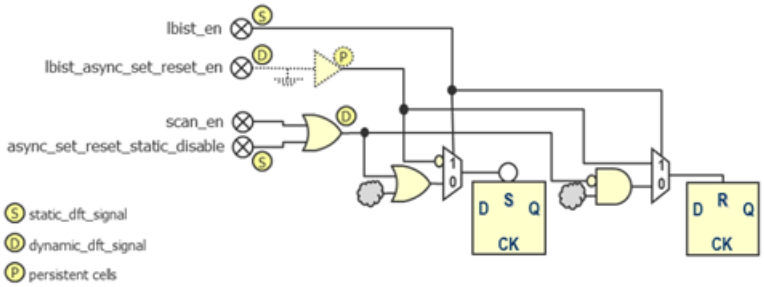
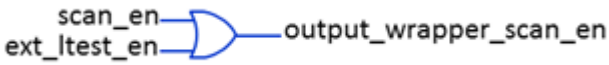
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-13. Pre-Registered Dynamic DFT Signal Names (cont.)

Dynamic DFT Signal Name	Usage and Default Implementation
edt_clock	A DFT signal that is used to supply a clock to the EDT controller. This can be an independent primary input, but it can also be synthesized using a clock gater driven by test_clock and having its enable sourced by scan_en. This is the recommended method as it permits sharing a common clock source for both the edt_clock and the shift_capture_clock. The DFT signal scan_en and test_clock must be added before the edt_clock can be added with the -create_from_other_signals switch, unless you intend to use SSN. In this case, the ScanHost node sources those signals. Refer to Figure 3-10 on page 224 for an illustration of its waveforms.  When hybrid LogicBIST is present, this gater is created within the LogicBIST controller. If this gater was inserted in a previous insertion pass, then it is removed when inserting the hybrid controller and the existing connections are moved to the LogicBIST output port. If you are using SSN with a ScanHost node, the edt_clock DFT signal can be inferred on the edt_clock output port of the ScanHost node. If you do not specify this signal and you set “edt_clock_present off” on the ScanHost, the signal is not created even if it is needed by EDT, OCC, or similar functionality. If you do explicitly add this signal, it will be redefined during the execution of the process_dft_specification command.
edt_update	A DFT signal that is used to supply the update enable port of the EDT controller. This signal is also used to create the shift_capture_clock. Refer to Figure 3-10 on page 224 for an illustration of its waveforms. If you are using SSN with a ScanHost node, the edt_update DFT signal can be inferred on the edt_update output port of the ScanHost node. You do not need to add it explicitly. If you do, the tool will redefine it during the execution of the process_dft_specification command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-13. Pre-Registered Dynamic DFT Signal Names (cont.)

Dynamic DFT Signal Name	Usage and Default Implementation
input_wrapper_scan_en	A DFT signal that is used (only for third-party scan insertion) to supply the scan_en signal to the input wrapper chains. When using the -create_from_other_signals switch, it is created by ORing scan_en with int_ltest_en. 
lbist_async_set_reset_dynamic_enable	A DFT signal that is used to enable the asynchronous set and reset pins of scannable flip-flops during the LogicBIST capture procedure. This signal is low during the LogicBIST shift cycles. This signal is multiplexed with the dynamic DFT signal async_set_reset_dynamic_disable and uses the static DFT signal lbist_en as the mux select.  If the Hybrid LogicBIST controller is not inserted at the same time as the DFT signal, then the tool uses a tied-off persistent buffer to anchor the DFT signal. When the tool inserts the LogicBIST controller, it deletes the buffer and moves the buffer fanout to a LogicBIST controller output port.
lbist_misr_accumulate_en	A DFT signal that indicates when the MISR accumulates. This is ON during shift cycles from the 2nd pattern onwards.
lbist_prpg_en	A DFT signal from lbist controller that indicates when the PRPG is active. This is ON during shift cycles and one cycle in post-capture when the low-power registers are updated.
lbist_reset	A DFT signal that resets the EDT logic asynchronously as part of test_setup.
output_wrapper_scan_en	A DFT signal that is used (only for third-party scan insertion) to supply the scan_en signal to the output wrapper chains. When using the -create_from_other_signals switch, it is created by ORing scan_en with ext_ltest_en. 

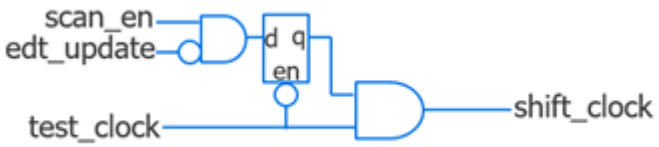
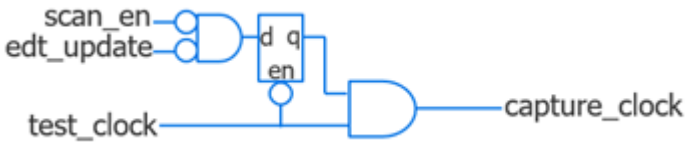
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-13. Pre-Registered Dynamic DFT Signal Names (cont.)

Dynamic DFT Signal Name	Usage and Default Implementation
scan_en	A DFT signal that is used to define the source of scan_en. When using an OCC, this signal is constrained off during the capture cycle. When the async_set_reset_static_disable signal is not set to 1, you must make sure the scan_en source is high during all cycles of the load_unload procedure to avoid D1 violations. Refer to Figure 3-10 on page 224 for an illustration of its waveforms. If you are using SSN with a ScanHost node, the scan_en DFT signal can be inferred on the scan_en output port of the ScanHost node. You do not need to add it explicitly. If you do, the tool will redefine it during the execution of the process_dft_specification command.
shift_capture_clock	A DFT signal that is used to supply the slow clock to the OCC controller. This can be an independent primary input, but it can also be synthesized using a clock gater driven by test_clock and having its enables sourced by edt_update. This is the recommended method as it permits sharing a common clock source for both the edt_clock and the shift_capture_clock. The DFT signal edt_update and test_clock must be added before the shift_capture_clock can be added with the -create_from_other_signal switch, unless you intend to use SSN. In that case, the ScanHost node sources those signals. Refer to Figure 3-10 on page 224 for an illustration of its waveforms. When hybrid LogicBIST is present, this gater is created within the LogicBIST controller. If this gater was inserted in a previous insertion pass, it is removed when inserting the hybrid controller and the existing connections are moved to the LogicBIST output port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-13. Pre-Registered Dynamic DFT Signal Names (cont.)

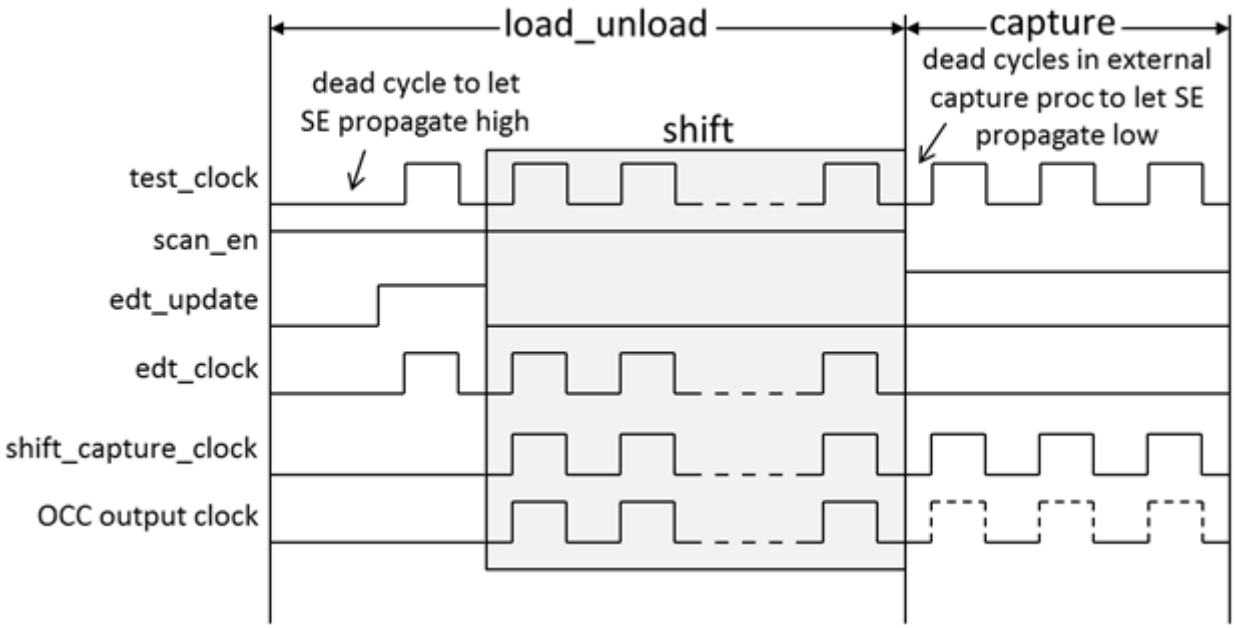
Dynamic DFT Signal Name	Usage and Default Implementation
shift_clock	A DFT signal that is used to create a shift_clock. This can be an independent primary input, but it can also be synthesized using a clock gater driven by test_clock and having its enables sourced by "!edt_update & scan_en". This is the recommended method, because it permits sharing a common clock source for both the edt_clock and the shift_clock. You must add the DFT signals edt_update, scan_en, and test_clock before the shift_clock can be added with the -create_from_other_signal switch, unless you intend to use SSN. In that case, the ScanHost node sources those signals. Refer to Figure 3-10 on page 224 for an illustration of its waveforms. The shift_clock includes only the pulses of the shift_capture_clock when scan_en is high. 
capture_clock	A DFT signal that is used to create a capture_clock. This can be an independent primary input, but it can also be synthesized using a clock gater driven by test_clock and having its enables sourced by "!edt_update & !scan_en". This is the recommended method, because it permits sharing a common clock source for both the edt_clock and the capture_clock. You must add the DFT signals edt_update, scan_en, and test_clock before the shift_clock can be added with the -create_from_other_signal switch, unless you intend to use SSN. In that case, the ScanHost node sources those signals. Refer to Figure 3-10 on page 224 for an illustration of its waveforms. The capture clock only includes the pulses of the shift_capture_clock when scan_en is low. 

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-13. Pre-Registered Dynamic DFT Signal Names (cont.)

Dynamic DFT Signal Name	Usage and Default Implementation
test_clock	A DFT signal that defines a clock input as the test_clock, which is to be used as the common source to synthesize the edt_clock and the shift_capture_clock. You do not need this signal if the edt_clock and the shift_capture_clock are sourced by independent sources. Refer to Figure 3-10 on page 224 for an illustration of its waveforms. **Note:** You cannot share or reuse the test_clock with an already-existing functional clock that feeds sequential elements intended to be tested with ATPG. The test_clock needs to be a separate independent clock because the test_clock must pulse an extra amount of time to initialize the EDT without disturbing the functional clock domains. Specify a top level port as the source for test_clock when you are not using SSN or when you are using SSN with scan_signal_bypass set to "on". You can share the same port for test_clock and ssn_bus_clock. Refer to the ScanHost property use_ssn_bus_clock_as_test_clock_bypass for more information about this functionality. The existing test_clock connections are not moved automatically to the ScanHost sourcing test_clock signal. If you require the test_clock to drive something, use the following command: **add_dft_control_points pin_port_net_spec -type \ dynamic_dft_control -dft_signal_source_name test_clock -ignore_existing_sources**

Figure 3-10. Waveforms of Selected Dynamic DFT Signals



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Knowing Which Signals to Add for Logic Test

Follow these instructions to know which signals to add for logic test:

- When you have memory BIST:

```
add_dft_signals memory_bypass_en
```

- When you have at least one Scan Tested Instrument like memory BIST or boundary scan (when boundary scan segments are used for logic test):

```
add_dft_signals tck_occ_en
```

- When you are using logic test:

```
add_dft_signals ltest_en async_set_reset_static_disable
```

- When you are using logic test without SSN:

```
add_dft_signal scan_en -source_nodes my_scan_en
```

- When you are creating a wrapped core:

```
add_dft_signals int_ltest_en ext_ltest_en
add_dft_signals int_mode ext_mode
```

- When using EDT without SSN, create the shift_capture_clock and edt_clock from a common test_clock:

```
add_dft_signals test_clock edt_update -source_nodes \
  {my_test_clock my_edt_update}
add_dft_signals shift_capture_clock edt_clock \
  -create_from_other_signals
```

- When not using EDT nor SSN:

```
add_dft_signals shift_capture_clock -source_nodes my_test_clock
```

- When reusing boundary scan for logic test at the top level, and you want to be able to perform logic test without contacting inputs that have boundary scan cells:

```
add_dft_signals int_ltest_en output_pad_disable
```

Arguments

- ***name*** ...

  A required repeatable string value used to specify one of more names of DFT signals to add. You can specify several names in one command line invocation. You can also supply several names in a single well formatted Tcl list. The name values must corresponds to built in DFT signal names or user registered ones. Refer to the register_static_dft_signal_names command description for instructions on how to register your own.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -source_nodes *pin_port_spec*

  An optional switch that is used to associate ports or pins to the DFT signal. This option is mandatory when adding the following dynamic DFT signals: scan_en, test_clock, and edt_update. Specified pin names must refer to existing objects. Port names are permitted to refer to ports that are to be created as long as they are scalar names and the design level set using the set_design_level command is not "chip". When the design level is set to "chip", specified port names must refer to existing ports and the ports must be connected to a properly described pad buffer cell. The connection to the DFT signal automatically maps the specified top-level port to the output pin of its associated input pad buffer.

  If you have added auxiliary input logic on the input pad used to source the dynamic DFT signals such as scan_en, test_clock and edt_clock, the DFT signal is automatically mapped on the auxiliary input data pin and an "add_dft_control_point -dft_signal_source_name ltest_en" command is inferred to control the auxiliary input enable pin as shown in Example 5. Refer to the documentation of the get_auxiliary_pins and the add_dft_modal_connections commands for a detailed description of the auxiliary logic and how to describe third party implementation using a Tcl dictionary called "::auxiliary_data_dict".

  If you do not specify any of the following switches:

  - -create_from_other_signals
  - -create_with_tdr
  - -source_node

  the -create_with_tdr switch is assumed when the design_level is not a sub_block; otherwise the -source_node switch with a port name equal to the DFT signal name is inferred by the tool. This is the recommended implementation for all static DFT signals. Refer to the set_static_dft_signal_values command to find how the setting of the static DFT signals is automated when sourced by IJTAG.

- -make_ijtag_port

  An optional switch that can be used when adding a static DFT signal with the -source_nodes switch referring to ports. The ports are identified as DataInPorts when running extract_icl. When running ICL extract at the next level, those ports are verified to be properly sourced by IJTAG data ports. The set_static_dft_signal_values command remains usable for setting those static DFT signals specified on a port when you also specify the -make_ijtag_port switch. The switch is inferred automatically when the static DFT signal is defined on a port and the design_level was specified as "chip" using the set_design_level command. It is also inferred when the -source_node switch is inferred to be equal to the DFT signal name when the design_level was specified as "sub_block" using the set_design_level command. Refer to Example 4.

- -associated_ports *port_spec*

  An optional switch and value pair that enables you to associate a port with a dynamic DFT signal when you specify an internal pin with the -source_nodes switch. You can use this

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

switch to identify the association with only the following dynamic DFT signals: scan_en, test_clock, and edt_update.

Port names can refer to ports that are not yet created if they are scalar names and the design level is not "chip" (set using the set_design_level command). When the design level is "chip", specified port names must refer to existing ports, and the ports must connect to a properly described pad buffer cell. The connection to the dynamic DFT signal automatically maps the specified top-level port to the output pin of its associated input pad buffer. See Example 8.

- -create_from_other_signals

  An optional switch that specifies to create the DFT signal from other signals. The list of the DFT signals that can be created from other signals are edt_clock, shift_capture_clock, async_set_reset_dynamic_disable, input_wrapper_scan_en, and output_wrapper_scan_enable. It is actually recommended that you use the -create_from_other_signals switch when adding them as it make the timing closure easier during synthesis and layout.

  Refer to Table 3-13 on page 219 for a description of those signals and how they are synthesized from other signals. Figure 3-10 on page 224 shows their respective waveforms.

- -parent_instance *parent_instance*

  An optional switch-value pair that specifies an instance in which to instantiate the clock gating cell used to synthesize the edt_clock and the shift_capture_clock signals. When left unspecified, the parent instance defaults to the current design. You can create several copies of the clock gating cells as long as you specify a different parent instance for each one of them. The other three dynamic signals that can be created from other signals only have one copy. The OR gate is instantiated in the parent instance of the static DFT signal that feeds one input of the OR gate.

- -leaf_instance_name *leaf_instance_name*

  An optional switch-value pair that specifies the leaf name of the logic gate used to synthesize the logic signal. By default, the leaf instance name is the name of the DFT signal prefixed by the value specified with the persistent_cell_prefix property inside the DftSpecification wrapper. The default is "tessent_persistent_cell_".

- -create_with_tdr

  An optional switch that specifies that the DFT signal is to be created from a TDR. This switch is assumed when the -source_node and the -create_from_other_signals switches are not specified. This switch is only permitted when adding static DFT signals.

  This switch is assumed when the -source_node and the -create_from_other_signals switches are not specified or inferred. Refer to the -source_node switch description to know when it is inferred automatically.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -disable_for_functional_safety {on | off | auto}

  An optional switch that specifies that a signal created from a TDR is created with redundancy gating. This switch is only valid when the -create_with_tdr switch is specified or inferred.

  on — Gating logic is added.

  off — Gating logic is not added.

  auto — This is the option default. The tool uses the value set with the set_dft_signals_options command.

- -add_hold_scan_cell {on | off | auto}

  An optional switch and literal pair to indicate whether a DFT signal should be created with or without additional control logic for ATPG.

  Note

  The insertion of the holding scan cell requires that the shift_capture_clock and ltest_en DFT signals are already defined. If they do not exist, they will be automatically created.

  If the tool automatically creates the shift_capture_clock signal, the edt_update and test_clock signals are also created because they are required for the shift_capture_clock clock gater. The tool reports the following notes when automatically creating these signals:

```
Note: Adding Dft signal 'edt_update' with option '-source_nodes
edt_update' because it is needed for holding scan cell

Note: Adding Dft signal 'test_clock' with option '-source_nodes
test_clock' because it is needed for holding scan cell

Note: Adding Dft signal 'shift_capture_clock' with option
'-create_from_other_signals' because it is needed for holding scan
cell

Note: Adding Dft signal 'ltest_en' because it is needed for holding
scan cell
```

  Restriction

  Automatic creation of the shift_capture_clock signal is not supported at the chip level.

  The DFT signals that support this option are:

  - async_set_reset_dynamic_disable (dynamic)
  - x_bounding_en (static)
  - mcp_bounding_en (static)
  - control_test_point_en (static)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- observe_test_point_en (static)
- wrapper_toggle_en

on — The supported DFT signal is created with additional control logic for ATPG.

off — The supported DFT signal is created without additional control logic for ATPG.

auto — This is the option default. The StaticDftSignals and DynamicDftSignals wrapper data for this option is inherited if modified before using this switch. By default, unless modified before using this switch, the DefaultsSpecification configuration syntax for inserting a holding scan cell for ATPG control defaults to "on" for the wrapper_toggle_en signal and "off" for all other valid DFT signals.

- -hold_scan_cell_disable *dft_signal*

  An optional switch and value pair that specifies the DFT signal used to disable the holding scan cell.

  dft_signal should be a pre-registered DFT signal. If the signal does not exist, the tool automatically adds it.

Examples

Example 1

The following example defines the DFT signals needed for each test block in the my_core1 design. The design is implementing memory BIST and logic tests using EDT and OCC. The first dofile adds DFT signals while inserting memory BIST and IJTAG into the RTL during RTL Pass1. The second dofile adds and automatically connects the Sib(sti) and DFT signals in the second pass, as it inserts the EDT and OCC controllers during RTL Pass2. Finally, the third dofile uses the DFT signals int_mode and ext_mode from RTL Pass1 during the Scan Insertion Pass with the add_scan_modes command.

RTL Pass1

```
set_context dft -rtl
read_cell_library ...
read_verilog ...
set_current_design my_core1
set_design_level physical_block
set_dft_specification_requirements -memory_test on



add_dft_signals ltest_en int_ltest_en ext_ltest_en \
                memory_bypass_en async_set_reset_static_disable \
                int_mode ext_mode
add_dft_signals scan_en test_clock edt_update \
               -source_nodes {my_scan_en my_test_clock my_edt_update}
add_dft_signals edt_clock shift_capture_clock \
                input_wrapper_scan_en output_wrapper_scan_en \
                async_set_reset_dynamic_disable \
                -create_from_other_signals

...
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

RTL Pass2

```
set_context dft -rtl2
read_cell_library ...
read_design my_core1 -design_id rtl
set_current_design my_core1
add_dft_signals ltest_en int_ltest_en ext_ltest_en \
                memory_bypass_en async_set_reset_static_disable \
                int_mode ext_mode
add_dft_signals scan_en test_clock edt_update \
               -source_nodes {my_scan_en my_test_clock my_edt_update}
add_dft_signals input_wrapper_scan_en output_wrapper_scan_en \
                async_set_reset_dynamic_disable \
                -create_from_other_signals
check_design_rules
set spec [create_dft_specification -sri_sib_list {occ edt}]
read_config_data -in $spec -from_string {
  OCC {
      ijtag_host_interface : Sib(occ);
  }
  EDT {
      ijtag_host_interface : sib(edt) ;
  }
}
set edt1 [add_config_element EDT/Controller(1)      -in $spec]
set_config_value output_channel_count 1             -in $edt1
set_config_value longest_chain_range {100 200}     -in $edt1
set_config_value scan_chain_count 500               -in $edt1
set_config_value input_channel_count 8              -in $edt1
set_config_value input_channel_count 3              -in $edt1

set_config_value EDT/Connections/edt_clock -in $spec \
                 DftSignal(edt_clock)
set_config_value EDT/Connections/edt_update -in $spec\
                 DftSignal(edt_update)

...

foreach clock [get_name_list [get_clocks -type async_source]] {
  set id [get_clock_option $clock -label]
  set occ [add_config_element OCC/Controller($id) -in $spec]
  set_config_value clock_intercept_node -in $occ \
      [get_single_name $clock]
  set_config_value Connections/slow_clock -in $occ \
      DftSignal(shift_capture_clock)
  set_config_value Connections/scan_en -in $occ \
      DftSignal(scan_en)
}
...
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Scan Insertion Pass

```
set_context dft -scan
read_cell_library ...
read_verilog ...
read_design my_core1 -design_id rtl2 -no_hdl
set_current_design my_core1

set edt_icl_module [get_icl_module \
                -filter tessent_instrument_type==mentor::edt]
set edt_instance [lindex \
              [get_attribute_value_list \
                [get_icl_instance -of_module $edt_icl_module] \
                -name hierarchical_design_instance] 0]

analyze_wrapper_cells
add_scan_mode int_mode \
          -type internal \
          -chain_length 200 \
          -edt_instance $edt_instance \
          -enable_dft_signal int_mode
add_scan_mode ext_mode \
          -type external \
          -chain_length 200
```

Example 2

This example shows the DFT signals pre-existing on a third-party TDR. The first part shows the DataOutPort attributed in the ICL module. The tessent_dft_function attribute identifies the ICL module as having attributed DFT signals and is mandatory for the attributed DFT signals to be recognized. For the tessent_dft_signal_* attributes, refer to the icl_port data model section. The second part calls the report_dft_signals command once the design has been elaborated and the ICL module matched to its Verilog counterpart to show how the DFT signals were auto detected and added as existing signals.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Section of MyIJTAGNode.icl

```
Module MyIJTAGNode {
  ...
  Attribute tessent_dft_function = "scan_resource_instrument_dft_control";
  DataOutPort XXX_all_test                  {
    Source TDR[0];
    Attribute connection_rule_option = "allowed_no_destination";
    Attribute tessent_dft_signal_name = "all_test";
    Attribute tessent_dft_signal_usage = "global_dft_control";
    Attribute tessent_dft_signal_default_value_in_all_test = 1;
    Attribute tessent_dft_signal_reset_value = 0;
  }
  DataOutPort XXX_ltest_en                  {
    Source TDR[3];
    Attribute connection_rule_option = "allowed_no_destination";
    Attribute tessent_dft_signal_name = "ltest_en";
    Attribute tessent_dft_signal_usage = "logic_test_control";
    Attribute tessent_dft_signal_value_in_pre_scan_drc = "1";
    Attribute tessent_dft_signal_reset_value = 0;
  }
  DataOutPort XXX_int_mode {
    Source TDR[6];
    Attribute connection_rule_option = "allowed_no_destination";
    Attribute tessent_dft_signal_name = "int_mode";
    Attribute tessent_dft_signal_usage = "scan_mode";
    Attribute tessent_dft_signal_scan_mode_type = "internal";
    Attribute tessent_dft_signal_reset_value = 0;
  }
  DataOutPort XXX_ext_mode {
    Source TDR[7];
    Attribute connection_rule_option = "allowed_no_destination";
    Attribute tessent_dft_signal_name = "ext_mode";
    Attribute tessent_dft_signal_usage = "scan_mode";
    Attribute tessent_dft_signal_scan_mode_type = "retargeting";
    Attribute tessent_dft_signal_reset_value = 0;
  }
  ScanRegister TDR[7:0] {
    ScanInSource si;
    ResetValue 8'b0;
  }
  ...
}
```

Dofile Section

set_context dft -rtl
read_cell_library ...
read_verilog ...
set_current_design my_core1
report_dft_signals

```
// --------- --------  --------------  --------------  ----------------
// Name      Status    Origin          Type            Location
// --------- --------  --------------  --------------  ----------------
// all_test  Existing  ICL attributes  IJTAG register  tdr/XXX_all_test
// ltest_en  Existing  ICL attributes  IJTAG register  tdr/XXX_ltest_en
// int_mode  Existing  ICL attributes  IJTAG register  tdr/XXX_int_mode
// ext_mode  Existing  ICL attributes  IJTAG register  tdr/XXX_ext_mode
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

This example shows you which DFT signal you need to add and when:

```
# When you have memory BIST
add_dft_signals memory_bypass_en

# When you have at least one Scan Tested Instrument like Memory BIST
add_dft_signals tck_occ_en

# When you are using logictest
add_dft_signals ltest_en
add_dft_signals scan_en -source_node my_scan_en

# When you are building a wrapped core
add_dft_signals int_ltest_en ext_ltest_en int_mode ext_mode

# When using EDT

add_dft_signals test_clock edt_update \
   -source_node {my_test_clock my_edt_update}
add_dft_signals edt_clock shift_capture_clock \
   -create_from_other_signals

# When not using EDT
add_dft_signals shift_capture_clock -source_node {my_test_clock}
report_dft_signals
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Existing and to be created DFT signals
// ======================================
// -------------------  --------  -------  ------------ -----------------
// Name                 Status    Origin   Type         Location
// -------------------  --------  -------  ------------ -----------------
// edt_clock            To be      User    ClockGateAnd
//                      created    added
//                                                 <persistent_cell_prefix>
//                                                  edt_clock
// edt_update           To be      User    Port         my_edt_update
//                      created    added
// ext_ltest_en         To be      User    IJTAG        register
//                      created    added
// ext_mode             To be      User    IJTAG        register
//                      created    added
// int_ltest_en         To be      User    IJTAG        register
//                      created    added
// int_mode             To be      User    IJTAG        register
//                      created    added
// ltest_en             To be      User    IJTAG        register
//                      created    added
// memory_bypass_en     To be      User    IJTAG        register
//                      created    added
// scan_en              To be      User    Port         my_scan_en
//                      created    added
// shift_capture_clock  To be      User    ClockGateAnd
//                      created    added
//                                                 <persistent_cell_prefix>
//                                                  shift_capture_clock
// tck_occ_en           To be      User    IJTAG        register
//                      created    added
// test_clock           To be      User    Port         my_test_clock
//                      created    added
//
```

Example 4

The following example illustrates how the -make_ijtag_port switch is automatically inferred for the static DFT signals defined on ports when the *design_level* is chip. In this example, the node gpio[136] (in red) is the inferred ijtag_port.

```
add_dft_signals test_clock edt_update ltest_en \
   -source_node {gpio[135] gpio[134] gpio[136]}
report_dft_signals
```

```
// Existing and to be created DFT signals
// ======================================
// ----------  --------  ----------  ----------  ----------------------
// Name        Status    Origin      Type        Location
// ----------  --------  ----------  ----------  ----------------------
// edt_update  Existing  User added  Port        gpio[134] -> gpio134/C
// ltest_en    Existing  User added  IJTAG port  gpio[136] -> gpio136/C
// test_clock  Existing  User added  Port        gpio[135] -> gpio135/C
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 5

The following example uses add_dft_signal to declare the source of scan_en, test_clock and edt_update DFT signals. The pads are equipped with auxiliary data logic using the AuxiliaryInputOutputPorts wrapper during boundary scan insertion, so they are automatically mapped to the auxiliary input pin and the auxiliary input enable pins are inferred to be driven by the “ltest_en” DFT signal.

add_dft_signals scan_en test_clock edt_update -source_node {gpio[32:30]}

```
//  Note: Adding DFT signal 'ltest_en' with the -create_with_tdr option.

//  Note: Inferred 'add_dft_control_points
//  top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_32_AuxInEn
//  -dft_signal_source_name ltest_en' because the DFT signal 'scan_en' is
//  sourced by an auxiliary data input pin.

//  Note: Inferred 'add_dft_control_points
//  top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_31_AuxInEn
//  -dft_signal_source_name ltest_en' because the DFT signal 'test_clock'
//  is sourced by an auxiliary data input pin.

//  Note: Inferred 'add_dft_control_points
//  top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_30_AuxInEn
//  -dft_signal_source_name ltest_en' because the DFT signal 'edt_update'
//  is sourced by an auxiliary data input pin.
```

report_dft_signals// Existing and to be created DFT signals

```
// ======================================
// ------------  ------------  ----------  --------------  -------------------------------
// Name         Status         Origin      Type            Location
// ------------  ------------  ----------  --------------  -------------------------------
// edt_update   Existing      User added  Port            gpio[30] -> DEF_inst/gpio_30_AuxIn
// ext_ltest_en Existing      User added  IJTAG port      gpio[33] -> gpio33/C
// ltest_en     To be created User added  IJTAG register
// scan_en      Existing      User added  Port            gpio[32] -> DEF_inst/gpio_32_AuxIn
// test_clock   Existing      User added  Port            gpio[31] -> DEF_inst/gpio_31_AuxIn
```

Example 6

The following example creates one static_dft_en with ATPG control and one without ATPG control.

add_dft_signals x_bounding_en -add_hold_scan_cell on
add_dft_signals mcp_bounding_en -source_node {mcp_bounding_en}

Note

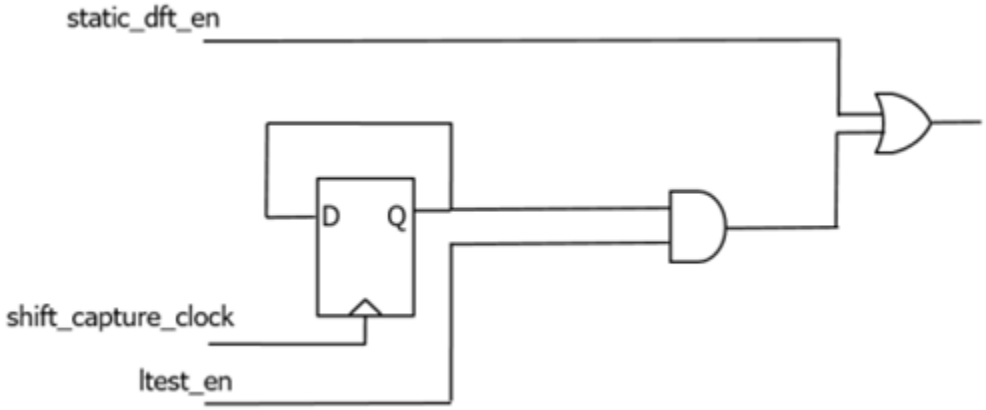
The “static_dft_en” generic term is used to represent any of the supported DFT signals in the list specified in “-add_hold_scan_cell {on | off | auto}”.

Neither of the static_dft_en signals is directly controlled by a TDR. The mcp_bounding_en signal was created from a source node. The node_name of x_bounding_en indicates that it is controlled by the inserted logic circuit. The original source node location and its type are

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reported under the unmapped_node_name and unmapped_type properties respectively. The "add_hold_scan_cell" field appears under Core(*module_name*)/DesignInfo/StaticDftSignals/StaticDftSignal(*static_dft_e*).

Figure 3-11. Holding Scan Cell Without Disable

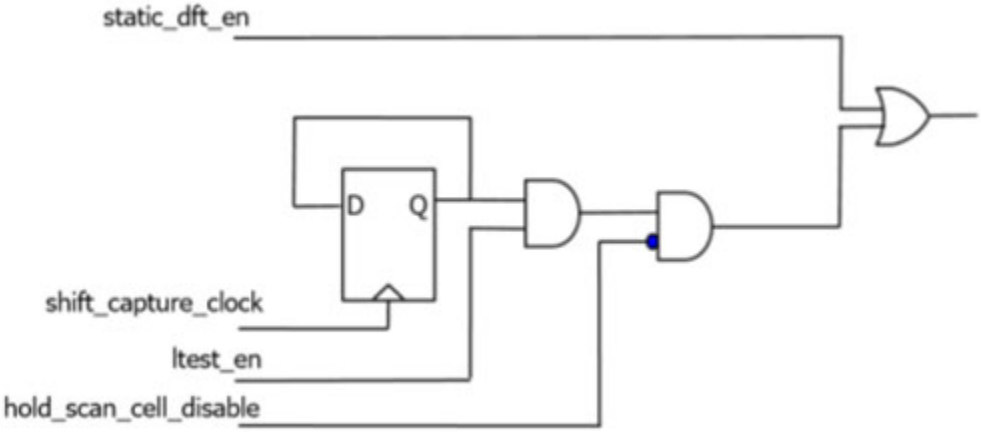


Example 7

This example shows a set of commands that register a user-defined static DFT signal, which can be used as a holding scan cell disabling signal. Then it adds a DFT signal with a holding scan cell.

```
register_static_dft_signal_names xbe_hold_scan_cell_disable \
-default_value 0 -reset_value 0
add_dft_signals x_bounding_en -add_hold_scan_cell on \
-hold_scan_cell_disable xbe_hold_scan_cell_disable
```

Figure 3-12. Holding Scan Cell With Disable

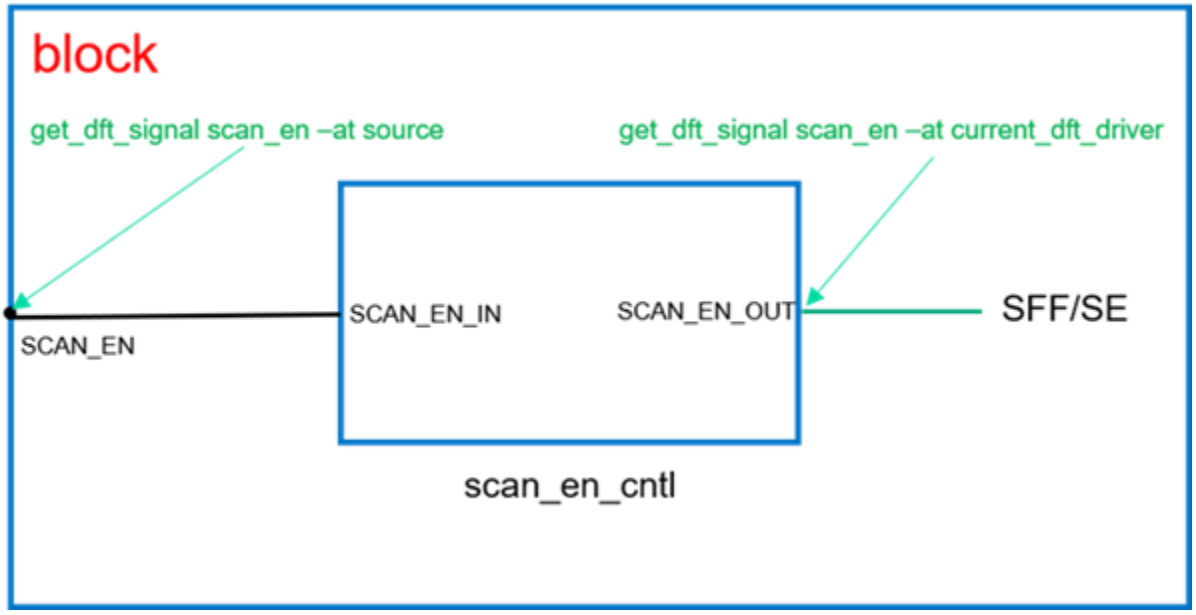


Example 8

The following example shows a design with an external module intercepting the scan_en signal coming from the port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 3-13. Block-Level Port Association



Define the scan_en dynamic DFT signal source port (to automate the signal connection at the parent level and constrain it) and the DFT connection point on the output of the scan_en_cntl block. The output of the scan_en_cntl should be saved as the current_dft_driver and used during scan insertion as the final point connected to scan flops.

add_dft_signals scan_en -source_node scan_en_cntl/SCAN_EN_OUT -associated_ports SCAN_EN

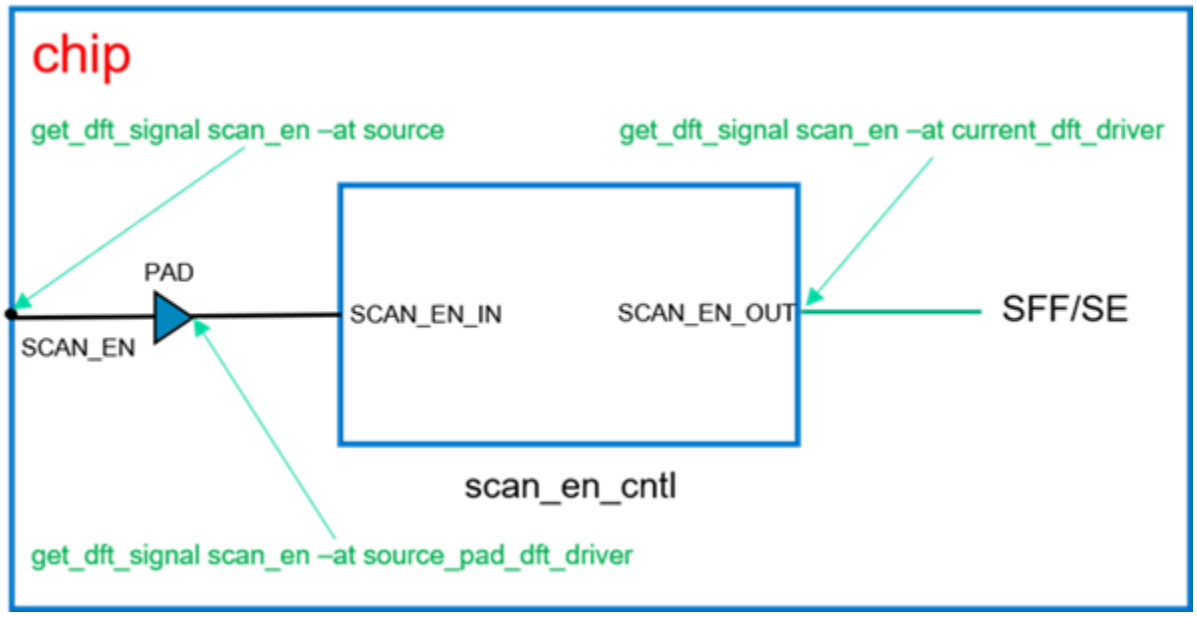
After block processing, the TCD file of this block has corresponding entries for the scan_en dynamic DFT signal to reflect all defined locations of the scan_en DFT signal.

```
Core(gps_baseband) {
  DesignInfo {
    DynamicDftSignals {
      DynamicDftSignal(scan_en) {
        Instance {
          type : pin;
          add_hold_scan_cell : off;
          node_name : scan_en_cntl/SCAN_EN_OUT;
          unmapped_node_name : SCAN_EN;
          unmapped_type : port;
        }
      }
    }
  }
}
```

If you do the signals declaration at the chip level, dynamic DFT signal mapping must also keep the location of the PAD output. The following figure shows the scan_en signal mapping and the corresponding TCD file snippet for chip level.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 3-14. Chip-Level Port Association




```
Core(gps_baseband) {
  DesignInfo {
    DynamicDftSignals {
      DynamicDftSignal(scan_en) {
        Instance {
          type : pin;
          add_hold_scan_cell : off;
          node_name : scan_en_cntl/SCAN_EN_OUT;
          unmapped_node_name : SCAN_EN;
          unmapped_type : port;
          At {
            source_pad_dft_driver_node_name : PAD/OUT;
          }
        }
      }
    }
  }
}
```

Related Topics

add_dft_control_points

add_dft_clock_mux

delete_dft_signals

get_dft_signal

process_dft_specification

register_static_dft_signal_names

report_dft_signals

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_dft_signal_names

set_static_dft_signal_values

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_drc_violation

Context: dft, patterns

Mode: analysis

Adds a DRC violation to the list of DRC violations for a given DRC rule.

Usage

add_drc_violation ***itcl_object_instance_name*** [-of_type *drc_name*]

Description

This command is used to create a new DRC violation. The main argument of the command, ***itcl_object_instance_name***, is the name of an object of an ITCL class containing a set of prescribed methods. The prescribed and required list of methods are display, report, get_location, and set_simulation_context. The Tessent Shell tool comes with a built-in DrcViolation ITCL class that provides these methods. The body of the class and the method are provided below. The likely hood that you ever need to write your own class is very low as this class is very flexible and complete. For example, if you wanted your display method to create a configuration tree instead of displaying logic in the schematic, then you would derive a new class from the DrcViolation and replace the "display" method with the method you create.

The -of_type argument is typically not used either as it defaults to the *drc_name*, which invokes the proc that calls the add_drc_violation command. As the add_drc_violation command is normally used in a proc referred by the "register_drc -drc_proc" command, it automatically inherits its *drc_name* from the caller.

You import the ITCL class into your namespace using the following command:

```
namespace import ::mentor::DrcUtilities::*
```

If you wanted to derive a new class to replace the body of a portion of the method, you would use the following:

```
itcl::class MyDrcViolation {
  inherit DrcViolation

  #Replaced methods
  ...
}
```

Refer to examples in the register_drc command description section for multiple example usages of the class. As shown in those examples, you use this to create an object of the class as follows:

```
set violation [DrcViolation #auto]
```

You then use the dict_set method to populate the information as shown in the code block as follows. If you want to show two paths at the same time, you can also populate the side1 data. The $path_to_tagged_objects is a list of collections of paths. The paths are typically the ones

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

created by the "trace_flat_model -store_paths_to_tagged_objects" switch. The permitted values for the path_direction are forward and backward. If the $path_to_tagged_objects contains just one element, you do not need to specify the direction of the path. The $msg_list variable contains a Tcl list of strings that represent the message that is displayed when the analyze_drc_violation or the report_drc_rules commands are run. A new line is inserted between each element of the message_string_list:

```
$violation dict_set list_of_paths side0 [list $path_to_tagged_objects]
$violation dict_set path_direction side0 backward
$violation dict_set start_callout_string side0 $start_callout_string
$violation dict_set end_callout_string side0 $end_callout_string
$violation dict_set message_string_list $msg_list
```

Once you have finished configuring the ITCL object, you add it as a DRC violation using the following command:

```
add_drc_violation [namespace current]::$violation
```

Below is the source code defining the DrcViolation as it existed when this documentation was written. You can always see the current list of methods or the implementation of each of the methods by issuing the following commands.

```
info class methods ::mentor::DrcUtilities::DrcViolation
info class definition ::mentor::DrcUtilities::DrcViolation display
```

Notice the use of the environment variable "DrcViolation.echo_commands" and "DrcViolation.no_run_commands" in the display method. If you set the environment variable "DrcViolation.echo_commands" to 1 using the setenv command in your command shell or within the Tessent Tcl shell, the commands used to display the violation are echoed in the transcript. If you set the environment variable "DrcViolation.no_run_commands" to 1, the command does not run. This is useful to create a regression suite where you can "goldenize" the echoed display commands in the transcript and make sure your custom DRC continues to display the right information without actually having to run the GUI and look at the result.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
##### DrcViolation.tcl #####

itcl::class DrcViolation {
  private {
    variable dictionary {
      list_of_paths {
        side0 {}
        side1 {}
      }
      path_direction {
        side0 {backward}
        side1 {backward}
      }
      message_string_list {}
      start_callout_string {
        side0 {}
        side1 {}
      }
      end_callout_string {
        side0 {}
        side1 {}
      }
      simulation_context {}
    }
  }
  #
  # Interface
  #
  method dict_set {args} {}
  method dict_get {args} {}
  method report {} {}
  method set_simulation_context {} {}
  method display {{append 0}} {}
  method get_location {} {}
  private method process_cmd {cmd} {}
  private method map_to_design_module_boundary {path_list position} {}
}
#
# Implementation
#
itcl::body DrcViolation::dict_set {args} {
  if {[llength $args] < 2} {
    return_error "Method 'dict_set' of 'DrcViolation' class requires at
least one key and one value."
  } elseif {[lindex $args 0] ni [dict keys $dictionary]} {
    return_error "Method 'dict_set' of 'DrcViolation' class support only
those keys : [join [dict keys $dictionary] ,]."
  }
  if {[lindex $args 0] eq "list_of_paths"} {
    if {[lindex $args 1] ni {side0 side1}} {
       return_error "Key following list_of_paths must be side0 or side1,
 not '[lindex $args 1]'"
    }
    set list_of_paths [lindex $args 2]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
    foreach path $list_of_paths {
      if {[get_attribute_value_list [index_collection $path 0] \
                                -name object_type] eq "gate_pin"} {
        lappend modified_list_of_paths [get_design_objects \
                                                -from_objects $path]
      } else {
        lappend modified_list_of_paths $path
      }
    }
    dict set dictionary list_of_paths [lindex $args 1] \
                                    $modified_list_of_paths
  } elseif {[lindex $args 0] eq "path_direction"} {
    if {[lindex $args 1] ni {side0 side1}} {
      return_error "Key following path_direction must be side0 or side1,
not '[lindex $args 1]'"
    }
    if {[lindex $args 2] ni {backward forward}} {
      return_error "Key following path_direction [lindex $args 1] must be
 backward or forward, not '[lindex $args 2]'"
    }
    dict set dictionary {*}$args
  } else {
    dict set dictionary {*}$args
  }
}
##################################################################
itcl::body DrcViolation::dict_get {args} {
  if {[llength $args] == 0} {
    return_error "Method 'get' of 'DrcViolation' class requires at least
one key."
  } elseif {[lindex $args 0] ni [dict keys $dictionary]} {
    return_error "Method 'get' of 'DrcViolation' class support only those
keys : [join [dict keys $dictionary] ,]."
  }
  dict get $dictionary {*}$args
}
##################################################################
itcl::body DrcViolation::report {} {
  return [dict get $dictionary message_string_list]
}
##################################################################
itcl::body DrcViolation::set_simulation_context {} {
  if {[dict get $dictionary simulation_context] ne ""} {
    process_cmd "set_current_simulation_context \
                     [dict get $dictionary simulation_context]"
    process_cmd "set_gate_report simulation_context"
  } else {
    process_cmd "set_gate_report normal"
  }
}
##################################################################
itcl::body DrcViolation::display {{append 0}} {
  set echo_commands   [expr {\
          [info exists ::env(DrcViolation.echo_commands)] && \
          $::env(DrcViolation.echo_commands) eq "1"}]
  set no_run_commands [expr {\
          [info exists ::env(DrcViolation.no_run_commands)] && \
           $::env(DrcViolation.no_run_commands) eq "1"}]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
    process_cmd "open_visualizer -display hierarchical_schematic"
    if {!$append} {
      process_cmd "delete_schematic_objects -all \
                       -display hierarchical_schematic"
    }
    set_simulation_context
    foreach side {side0 side1} {
      foreach path [dict get $dictionary list_of_paths $side] {
        if {[sizeof_collection $path] == 0} {
          continue
        }
        set cmd "add_schematic_path -highlight red \
                   -display hierarchical_schematic \
                   -direction [dict get $dictionary path_direction $side] "
        if {$echo_commands} {
          set display_cmd $cmd
          append display_cmd \
            "\{\\\n  \{[join [get_name_list $path] "\} \\\n  \{"]\} \\\n\}"
          puts $display_cmd
        }
        if {!$no_run_commands} {
          append cmd $path
          eval $cmd
        }
      }
      foreach point {start end} {
        set object [map_to_design_module_boundary \
                   [dict get $dictionary list_of_paths $side] $point]
        if {[dict get $dictionary ${point}_callout_string $side] ne "" &&\
            ![info exists done(${point}_callout_string,$side)] && \
            [sizeof_collection $object] > 0} {
          if {$echo_commands} {
            set callout_string \
               [dict get $dictionary ${point}_callout_string $side]
            puts "add_achematic_callout \"[get_single_name $object ]\"
              -message  \"$callout_string\" \
              -display hierarchical_schematic"
          }
          if {!$no_run_commands} {
            eval "add_schematic_callout $object
                    -message \"$callout_string\" \
                    -display hierarchical_schematic"
          }
        }
      }
    }
  }
  ##################################################################
  itcl::body DrcViolation::get_location {} {
    # Location is the start of the path
    if {[llength [dict get $dictionary list_of_paths side0]] > 0 &&\
        [sizeof_collection [lindex [dict get $dictionary list_of_paths \
        side0] 0]] > 0} {
      set loc [index_collection [lindex [dict get $dictionary \
                  list_of_paths side0] 0] 0]
    } else {
      set loc [index_collection [lindex [dict get $dictionary \
                  list_of_paths side1] 0] 0]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
    }
    if {[get_attribute_value_list $loc -name object_type] eq \
       "pseudo_port"} {
      set loc [get_design_objects -from_objects $loc -map_pseudo_ports]
    }
    if {[get_attribute_value_list $loc -name object_type] eq "net"} {
      set loc [get_instances -of_nets $loc]
    }
    if {[sizeof_collection $loc] == 0} {
      set loc [get_current_design]
    }
    return $loc
  }
  ##################################################################
  itcl::body DrcViolation::process_cmd {cmd} {
    upvar no_run_commands no_run_commands
    upvar echo_commands   echo_commands
    if {$echo_commands} {
      puts $cmd
    }
    if {!$no_run_commands} {
      eval $cmd
    }
  }
  ##################################################################
  itcl::body DrcViolation::map_to_design_module_boundary \
                        {path_list position} {
    if {$position ni {start end}} {
      return_error "position is not start or end"
    }
    if {$position eq "end"} {
      for {set i [expr [llength $path_list] -1]} {$i >= 0} {incr i -1} {
        set path [lindex $path_list $i]
        if {[sizeof_collection $path] > 0} {
          return [index_collection $path \
                      [expr [sizeof_collection $path] -1]]
        }
      }
    } else {
      for {set i 0} {$i < [llength $path_list]} {incr i} {
        set path [lindex $path_list $i]
        if {[sizeof_collection $path] > 0} {
          return [index_collection $path 0]
        }
      }
    }
  }
```

Arguments

- ***itcl_object_instance_name***

  A required string containing the name of an object of an iTcl class. As described above, the iTcl class must have a prescribed set of required methods. Refer to the description above for a description of those methods and the description of the built-in DrcViolation class that you can use for your custom DRC violation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -of_type *drc_name*

  An optional switch-value pair used to specify the DRC name for which the violation belongs. The -of_type argument is typically not used either as it defaults to the *drc_name*, which invoke the proc that called the add_drc_violation command. As the add_drc_violation command is normally used in a proc referred by the "register_drc -drc_proc" switch, it automatically inherits its *drc_name* from the caller.

Examples

The following example usage of the add_drc_violation command is taken from one of the many examples shown in the register_drc command examples. The built-in DrcViolation class is used to create the $drc_violation object. The dict_set method is used to configure its data. After that the add_drc_violation command is issued to create the violation.

```
if {[sizeof_collection $multi_input_gates] > 0} {
  append_to_collection multi_input_cells \
    [get_instances -of_gate_pins $multi_input_gates -silent] -unique
  set multi_input_cell_count [sizeof_collection $multi_input_cells]
  set violation [DrcViolation #auto]
  $violation dict_set list_of_paths side0 \
                              [list $path_to_tagged_objects]
  $violation dict_set path_direction side0 backward
  $violation dict_set start_callout_string side0 \
        "PO with path to PI with multi input gates along the path"
  $violation dict_set end_callout_string side0 "PI in fanin of PO"
  set start_port [index_collection $path_to_tagged_objects 0]
  set end_port [index_collection $path_to_tagged_objects \
           [expr [sizeof_collection $path_to_tagged_objects] - 1]]
  lappend msg_list "The output port '[get_single_name $start_port]' has a
combinational path from the input port '[get_single_name $end_port]'"
  lappend msg_list "but the path is not a pure feed-through as the path
crosses $multi_input_cell_count multi-input cells:"
  set cnt 1
  set max 3
  foreach_in_collection multi_input_cell $multi_input_cells {
    if {$cnt > $max} {
      lappend msg_list "  ..."
      break
    }
    lappend msg_list "  [get_single_name $multi_input_cell]"
    incr cnt
  }
  $violation dict_set message_string_list $msg_list
  add_drc_violation [namespace current]::$violation
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_edt_blocks

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Creates an arbitrary identifier for an EDT block and applies certain subsequent commands only to that block.

Usage

add_edt_blocks ***block_name***

Description

This command defines a string, referred to as the block name or block tag, as the identifier for an EDT block instantiated in a design netlist and simultaneously restricts the applicability of certain subsequent commands to that block. Restricting the applicability of these commands to a particular EDT block is referred to as setting or changing the EDT context, and the EDT block on which context-sensitive commands currently operate is referred to as the current EDT block. The current EDT block remains the sole target of the context-sensitive commands until you delete its definition using the delete_edt_blocks command or change the context to a different EDT block using set_current_edt_block or another add_edt_blocks command.

You must use this command to define a unique name for an EDT block prior to defining its attributes using context-sensitive commands. The add_scan_chains command, EDT-specific commands that define attributes for a given block, and some reporting commands are presently the only context-sensitive commands. All other commands (set_system_mode, create_patterns and report_statistics for example) apply to the entire netlist.

If multiple EDT blocks are defined and none of them is specified as the current EDT block, subsequent context-sensitive commands that define attributes for a given block ("set_edt_options -channels" for example) returns an error. This can happen if you add multiple EDT blocks, then delete the one that is the current EDT block.

For more information about EDT blocks or the modular Tessent TestKompress flow, refer to the "Modular Compressed ATPG" chapter of the *Tessent TestKompress User's Manual*.

Arguments

- ***block_name***

  A required string that specifies a name to use for uniquely identifying a specific EDT block instantiated within a netlist. The name of the block must begin with an alphabetic character and contain only alphabetic characters, numeric characters, and the underscore (a-zA-Z0-9_). Consecutive underscores are not permitted. Underscores cannot appear as the first or last character of the identifier. The block_name is independent of the module and instance names used in the netlist. Escaped identifiers are not permitted.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example defines the block name, “my_core1_block” for an EDT block that contains four scan chains. The tool determines the EDT block to associate with this name from the pin pathnames specified in the add_scan_chains commands that follow the add_edt_blocks command.

```
add_edt_blocks my_core1_block

# my_core1_block is the current EDT block.

...

add_scan_chains -internal core1_chain1 grp1 /edt_top_core1/cpu1_i/edt_si1 \
        /edt_top_core1/cpu1_i/edt_so1
add_scan_chains -internal core1_chain2 grp1 /edt_top_core1/cpu1_i/edt_si2 \
        /edt_top_core1/cpu1_i/edt_so2
add_scan_chains -internal core1_chain3 grp1 /edt_top_core1/cpu1_i/edt_si3 \
        /edt_top_core1/cpu1_i/edt_so3
add_scan_chains -internal core1_chain4 grp1 /edt_top_core1/cpu1_i/edt_si4 \
        /edt_top_core1/cpu1_i/edt_so4
```

Example 2

The following example defines two more block names, “my_core2_block” and “my_core3_block”, then displays a list of the current block names. The display indicates subsequent context-sensitive commands apply only to the EDT block named “my_core3_block”.

```
add_edt_blocks my_core2_block

# my_core2_block is the current EDT block.

...
add_scan_chains -internal core2_chain1 grp1 /edt_top_core2/cpu3_i/edt_si1 \
        /edt_top_core2/cpu3_i/edt_so1
add_scan_chains -internal core2_chain2 grp1 /edt_top_core2/cpu3_i/edt_si2 \
        /edt_top_core2/cpu3_i/edt_so2
...

add_edt_blocks my_core3_block

# my_core3_block is the current EDT block.

...
add_scan_chains -internal core3_chain1 grp1 /edt_top_core3/cpu7_i/edt_si1 \
        /edt_top_core3/cpu7_i/edt_so1
...
add_scan_chains -internal core3_chain8 grp1 /edt_top_core3/cpu7_i/edt_si8 \
        /edt_top_core3/cpu7_i/edt_so8

report_edt_blocks

//  my_core1_block
//  my_core2_block
//  my_core3_block   (current block)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

The following example removes from the tool's internal database, the data for the EDT block named "my_core3_block", making it unavailable for use with the set_current_edt_block command.

delete_edt_blocks my_core3_block

```
// Block deleted. No EDT block now selected as current block.
```

report_edt_blocks

```
//  my_core1_block
//  my_core2_block
```

Related Topics

delete_edt_blocks

report_edt_blocks

set_current_edt_block

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_edt_configurations

Context: dft -edt

Mode: setup

Sets up EDT logic with two compression configurations.

Usage

add_edt_configurations ***config_name***

Description

The add_edt_configurations command creates a compression configuration and applies subsequent EDT channel specifications to the configuration. Up to two compression configurations can be defined for a design block.

In a multi-block design flow, you can define the compression configurations at the top-level integration and all subsequently added blocks inherit the defined compression configurations. The configuration parameters need to be set for each block context.

Note

When two configurations are specified and one of the configurations is missing an input/output channel parameter, it inherits the needed value from the other configuration. You can override this feature by explicitly specifying the channel configuration.

For more information on setting up two compression configurations, see the "Defining Dual Compression Configurations" section in the *Tessent TestKompress User's Manual*.

Arguments

- ***config_name***

  A required string that specifies a name for the compression configuration.

Examples

The following example shows a dofile that creates two configurations for a single block.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# edt_ip_creation.do
#
# Dofile for EDT logic creation
dofile scan_chain_setup.dofile

# Set up EDT configurations
set_edt_options -location internal
add_edt_configurations my_sys_test_config
set_edt_options -input_channels 4 -output_channels 2
add_edt_configurations my_wafer_test_config
set_edt_options -input_channels 5 -output_channels 4

# Set bypass pin
set_edt_pins bypass my_bypass_pin

# Set edt configuration pin
set_edt_pins configuration my_configuration_pin
set_system_mode analysis

# Report and write EDT logic.
report_edt_configurations -all # Report configurations for all blocks.
report_edt_pins # Report all pins including compression configuration
# specific pins.
write_edt_files created -verilog -replace # Create dofiles and
# test procedure files for both the configurations and the bypass mode

write_design -output_file created_edt_top_rtl.v -replace # Write the
# current design in Verilog format to the file created_edt_top_rtl.v.
```

Related Topics

delete_edt_configurations

report_edt_configurations

set_current_edt_configuration

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_false_paths

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Defines false paths in the design. False paths are paths that cannot be activated in the design's functional (at-speed) mode of operation. You would typically determine them as a part of static timing analysis, prior to ATPG.

Usage

add_false_paths
{[-From *source_node…* | -FROM_Clock *clock_signal_name…*]
[-TO *sink_node…* | -TO_Clock *clock_signal_name…*]
[{-THrough *through_node…*}…]} | **-CRoss_clock_domains**
[-Hold] [-Setup] [-MAX_Skew {HALf_cycle | ONE_cycle | MULtiple_cycles}]

Description

If a false path is sensitized for an at-speed test during ATPG, invalid capture responses can occur at certain scan cells and POs with resulting invalid capture responses at the end points of the false path. This can cause simulation mismatches when you verify the patterns in a timing-based simulator. By defining the false paths, you enable the tool to identify the invalid capture responses. Once identified, the tool masks these responses (modify them to X) in the resultant patterns so they are not the source of simulation mismatches.

For additional information, refer to "Pattern Failures Due to Timing Exception Paths," in the *Tessent Scan and ATPG User's Manual*.

You can use the "set_timing_exceptions_handling -allow_invalid_pin_names on" command to enable the add_false_paths command to issue warnings instead of errors when defined objects are not found.

Arguments

Tip

When using the following arguments, you can include any number of asterisks (*) in pin or instance pathnames. The command treats an asterisk as a wildcard character, enabling you to use it to match many pathnames in the design.

- -From *source_node…*

  An optional switch and repeatable string pair that specifies the starting point of a false path. A valid starting point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the output pins of the instance to be starting points. For a clock PI, the tool considers the outputs of all the non-transparent latches, flip-flops and RAMs associated with the clock PI to be starting points. The tool considers a pin pathname to be that of an internal clock pin if its fanout

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

By default, if a specified source node does not translate to an internal gate, the tool transcribes an error message, and not add the false path definition to its internal database. If you issue the "set_timing_exceptions_handling -allow_invalid_pin_name on" command, the tool reports a warning and adds the false path definition.

- -FROM_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the starting point of a false path by its associated clock pin pathname. The tool interprets any pin pathname you specify with this argument to be the source of a clock signal and considers the outputs of all non-transparent latches, flip-flops and RAMs associated with this clock signal to be starting points.

  By default, if a specified clock signal name does not translate to an internal gate, the tool transcribes an error message and not add the false path definition to its internal database. If you issue the "set_timing_exceptions_handling -allow_invalid_pin_name on" command, the tool reports a warning and adds the false path definition.

- -TO *sink_node…*

  An optional switch and repeatable string pair that specifies the end point of a false path. A valid end point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the input pins of the instance to be end points. For a clock PI, the tool considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with the clock PI to be end points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

  By default, if a specified sink node does not translate to an internal gate, the tool transcribes an error message and does not add the false path definition to its internal database.If you issue the "set_timing_exceptions_handling -allow_invalid_pin_name on" command, the tool reports a warning and adds the false path definition.

  Note

  For a 2-port latch with ports: Port1 = (D1, CLK1), Port2 = (D2, CLK2), and primary input clocks CLK1 and CLK2, the command "add_false_paths... -to CLK2" would define paths with an end point at D2.

- -TO_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the end point of a false path by its associated clock pin pathname. The tool interprets any pin pathname you specify with this argument to be the source of an internal clock signal and considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with this clock to be end points.

  By default, if a specified clock signal name does not translate to an internal gate, the tool transcribes an error message and does not add the false path definition to its internal

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

database. If you issue the "set_timing_exceptions_handling -allow_invalid_pin_name on" command, the tool reports a warning and adds the false path definition.

- {-THrough *through_node…*}…

  An optional, repeatable switch and repeatable string pair that specifies circuit node(s) through which the false path must pass. A valid through node is a pin or instance pathname. Clock PIs are not supported as through nodes and the tool makes no attempt to determine if a pin pathname is that of an internal clock pin.

  When you use multiple -Through arguments, the tool considers their order left-to-right in the command string to be the order in which a signal would pass through the nodes on the specified path. For example, the following false path specification:

  ```
  add_false_paths -from  A  -through  B  C  -through  D  E  -to  F
  ```

  specifies all paths that start from A, pass through either B or C, then pass through D or E, and end at F. If you do not use the -Through argument when specifying a path, the tool considers all paths from the specified starting points to the specified end points to be false paths.

  By default, if a through node does not translate to an internal gate, the tool transcribes an error message and does not add the false path definition to its internal database. If you issue the "set_timing_exceptions_handling -allow_invalid_pin_name on" command, the tool reports a warning and adds the false path definition.

- **-CRoss_clock_domains**

  A switch that adds hold time false paths between all clocks.

  Note

  You cannot use this switch in Setup mode.

  Use this switch instead of multiple add_false_paths command instances using the -TO_Clock and -FROM_Clock switches to specify each path and direction between the clocks. Refer to Example 7.

- -Hold

  An optional switch that evaluates the false path for hold time exceptions.

  By default, the tool considers the hold time false path only when the -from flop and the -to flop are updated at the same clock edge for most paths. This behavior matches the "same edge" interaction of the domain clock setting. The exception is when you use the "add_false_paths -cross_clock_domains" or the false paths derived from asynchronous clock groups defined in SDC files. For such cross clock domain false paths, the tool assumes that the clock skew can be multiple cycles.

  When you define a false path without the -hold and the -setup options, the default is to consider the false path with both the hold and setup time exceptions.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Setup

  An optional switch that evaluates the false path for setup time exceptions.

  For more information, refer to "Types of Timing Exception Paths" in the *Tessent Scan and ATPG User's Manual*.

- -MAX_Skew

  An optional switch that specifies the maximum clock skew for the hold time exception for this false path. This option overrides the global -max_skew specified by the set_timing_exceptions_handling command.

  HALf_cycle

  Specifies half cycle as the maximum clock skew. This is the default value for this option, unless -cross_clock_domains is specified, in which case, the max skew is multiple cycle.

  ONE_cycle

  Specifies one cycle as the maximum clock skew.

  MULtiple_cycles

  Specifies multiple cycles as the maximum clock skew.

Examples

Example 1

The following example creates three false paths definitions, then reports all current false path definitions:

```
add_false_paths -from  /my_design/a  /my_design/b  -through  /my_design/u1
add_false_paths -to  /my_design/c
add_false_paths -through  /my_design/u24  /my_design/u25
report_false_paths -all
```

```
// False Path -from /my_design/a /my_design/b -through /my_design/u1
// False Path -to /my_design/c
// False Path -through /my_design/u24 /my_design/u25
// Total reported paths = 3
```

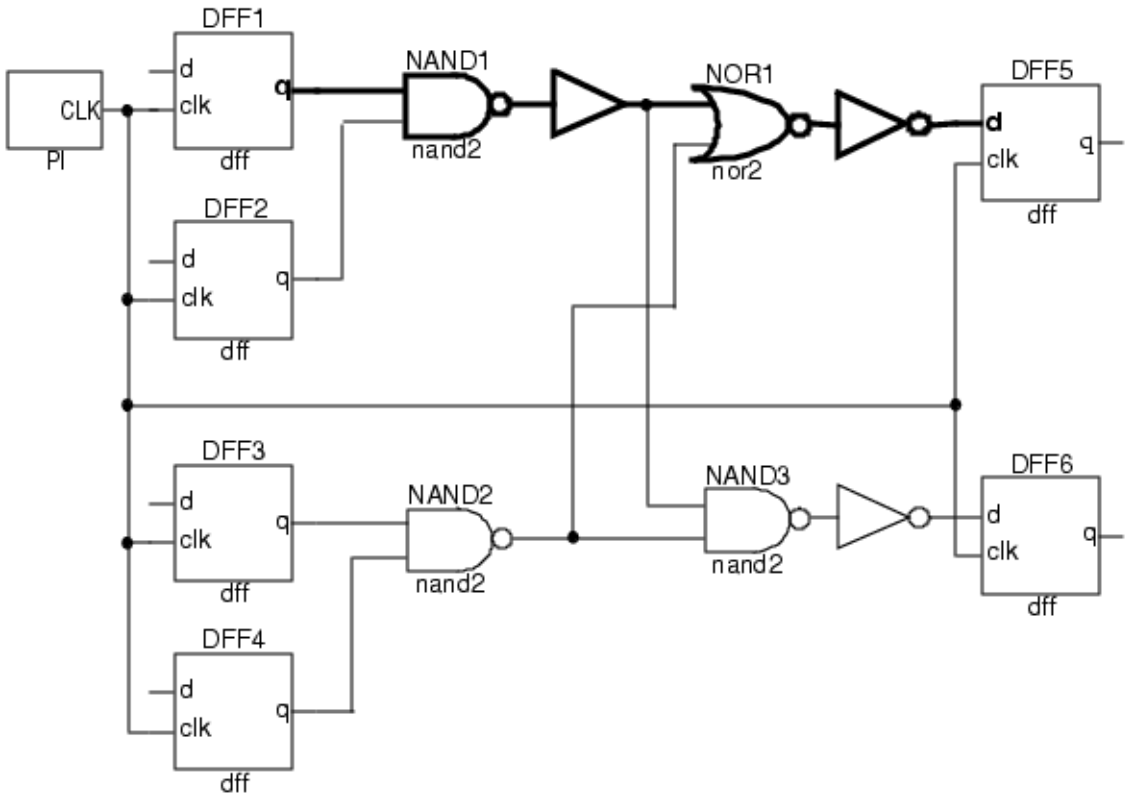
Example 2

The following example defines the path highlighted in Figure 3-15 as a false path:

```
add_false_paths -from  DFF1/q  -to  DFF5/d
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 3-15. Example of a False Path to be Defined



Here are four more ways you could define the same false path:

```
add_false_paths -from  DFF1/q  -through  NOR1  -to_clock  CLK
add_false_paths -from  DFF1/clk  -to  DFF5/d
add_false_paths -from  DFF1/q  -to  DFF5/clk
add_false_paths -from  DFF1/clk  -to  DFF5/clk
```

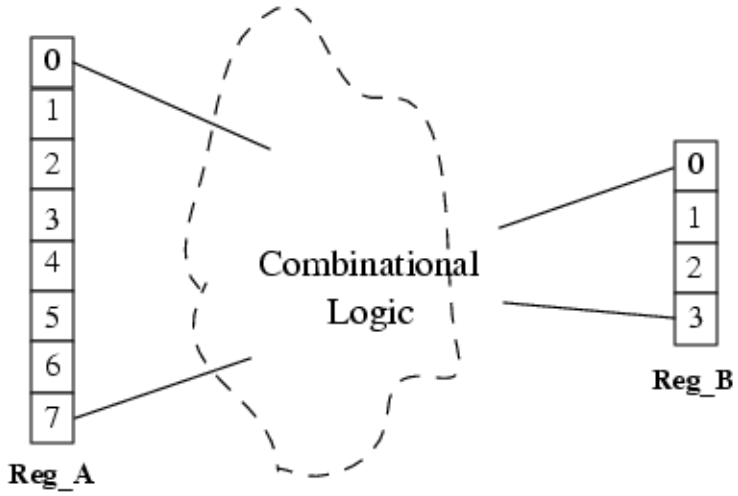
Example 3

Referring to Figure 3-16, the following example defines as false paths all the paths that start from Reg_A[0...7] and end at Reg_B[0...3]:

```
add_false_paths -from  Reg_A  -to Reg_B
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 3-16. Example Circuit for Defining False Paths



Example 4

This example defines as false paths (in Figure 3-16) all the paths that start from Reg_A[0] and end at Reg_B[0...3]:

add_false_paths -from Reg_A[0] -to Reg_B

Example 5

This example defines as false paths all the paths that start from Reg_A[0] or Reg_A[3] and end at Reg_B[2]:

add_false_paths -from Reg_A[0] Reg_A[3] -to Reg_B[2]

Example 6

The remaining six examples describe signal matches for various sample pin pathnames specified using the asterisk (*) wildcard character. The pathnames are based on the following design:

```
module top(...);
   Input ...;
   Output...;
   ...
   foo u1(...);
   ...
endmodule;module foo(...);
   Input...;
   Output...;
   ...
   staticRegister I_reg_1(.D1(...), .C1(...), .Z(...));
   staticRegister I_reg_2(.D1(...), .C1(...), .Z(...));
   staticRegister I_reg_3(.D1(...), .C1(...), .Z(...));
   staticRegister F_reg_1(.D1(...), .C1(...), .Z(...));
   staticRegister F_reg_2(.D1(...), .C1(...), .Z(...));
   ...
endmodule;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

1. The signal name /u1/*reg_1/D1 maps to signals /u1/I_reg_1/D1 and /u1/F_reg_1/D1.
2. The signal name /u1/I_reg*/D1 maps to signals /u1/I_reg_1/D1, /u1/I_reg_2/D1 and /u1/I_reg_3/D1.
3. The signal name /u1/*reg*/Z maps to signals /u1/I_reg_1/Z, /u1/I_reg_2/Z, /u1/I_reg_3/Z, /u1/F_reg_1/Z and /u1/F_reg_2/Z.
4. The signal name /u1/F_reg_2/* maps to signals /u1/F_reg_2/D1, /u1/F_reg_2/C1 and /u1/F_reg_2/Z.
5. The signal name /u1/F*/*1 maps to signals /u1/F_reg_1/D1, /u1/F_reg_1/C1, /u1/F_reg_2/D1 and /u1/F_reg_2/C1.
6. The signal name /u1/I*1/C1 maps to signal /u1/I_reg_1/C1.

Example 7

Assume the design contains two defined clocks: clk1 and clk2. The following command adds hold time false paths between clk1 and clk2:

```
add_false_paths -cross_clock_domains -hold
```

The above command is equivalent to the following two commands:

```
add_false_paths -from_clock clk1 -to_clock clk2 -hold
add_false_paths -from_clock clk2 -to_clock clk1  -hold
```

Related Topics

delete_false_paths

delete_multicycle_paths

read_sdc

report_false_paths

report_multicycle_paths

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_fault_sites

Context: dft -edt, patterns -scan

Mode: analysis

Automatically adds fault site definitions for library cells that do not have UDFM definitions.

Usage

add_fault_sites {**-ALl** | **-UNDEFINED_Cells**} [-VERBose]

Description

The add_fault_sites command automatically adds port fault site definitions for library cells that are not covered by the current UDFM definitions so that you do not have to add them manually to the UDFM file. In order to distinguish those added definitions and resulting fault sites in report functions, the added definitions are collected in the UDFM type "auto-inserted-faults."

After you've added the fault sites, the add_faults command adds static or delay faults to those locations, as specified by the "set_fault_type udfm" command.

Note

The add_fault_sites command works only for the fault type UDFM, and is available only after you read a UDFM fault file and before you create the fault list.

For more information about UDFM, refer to "About User-Defined Fault Modeling" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- **-ALl**

  A required switch that automatically adds UDFM definitions for all undefined (uncovered) port fault sites, including PI's and PO's. The fault sites affected by this switch include all those affected by the -Undefined_Cells switch plus remaining port fault site definitions for existing cells and modules, as well as additional port fault sites without this scope.

- **-UNDEFINED_Cells**

  A required switch that automatically adds UDFM definitions for cell port fault sites in undefined (uncovered) cells or modules.

- -VERBose

  An optional switch that displays additional information about each added UDFM definition.

Examples

The following example shows the effect of adding fault site definitions for all library cells not covered by UDFM definitions:

```
read_fault_sites /home/design/c090_std/c090_std.udfm
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Reading UDFM File: /home/design/c090_std/c090_std.udfm
//  UDFM definitions read: Total=3, Cells=1, Modules=2, Instances=0,
//  No Match=0
```

report_fault_sites -undefined_cells

```
//  Report of cells not defined by UDFM
//
//  Cell Name  Instances
//  ---------  ---------
//  M1                 2
//  M2                 1
//  u_mux2             1
//  ---------  ---------
```

add_fault_sites -undefined_cells

```
//  3 UDFM definition(s) for uncovered cells have been added
```

add faults -all

```
//  Number added faults=56  instances considered=7  not-considered=3
```

Related Topics

add_faults

delete_faults

read_fault_sites

report_fault_sites

set_fault_type

write_fault_sites

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_faults

Context: dft -edt, dft -test_points, patterns -scan

Mode: analysis

Adds faults to the current fault list, discards all patterns in the current test pattern set, and sets all faults to undetected.

Usage

add_faults ***fault_model_specific_options*** [*power_aware_options*] [-KEEP_Patterns]
[{> | >>} *file_pathname*]

Stuck/Transition/Toggle/Iddq Fault Model Options:
[-All | { *object_expression*…
[-PIN | -INstance | -MODule [-PIN *module_pin_pathname* ...]] }
| {-CLOCK_domains {ALL | *clock_pathname*…} [-NO_EQUivalent_clocks]
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]
[-exclude_constrained_clock_domains]}
| {-CAPture_procedures {ALL | *capture_procedure_name*…}}]
[-Stuck_at {01 | 0 | 1}] [-VERBose]
[-FRom *pin* ...][-THrough *pin* ...][-TO *pin* ...][-FROM_Clock *pin* ...][-TO_Clock *pin* ...]

UDFM Fault Model Options:
[-All | {*object_expression*... [-INstance]}
| {-CLOCK_domains {ALL | *clock_pathname*…} [-NO_EQUivalent_clocks]
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]
[-exclude_constrained_clock_domains]}]
[-UDFM_type *name*] [-CELL *name*] [-MODule *name*] [-FAULT *name*] [-VERBose]
[-FRom *pin* ...][-THrough *pin* ...][-TO *pin* ...][-FROM_Clock *pin* ...][-TO_Clock *pin* ...]

Bridge Fault Model Options:
[-All | {*bridge_name*… -NAME} | {*net_pathname*… -SINGle}
| {*instance_name*… -SINGle} |
{-CLOCK_domains {ALL | *clock_pathname*…} [-NO_EQUivalent_clocks]
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]
[-exclude_constrained_clock_domains]}
| {-CAPture_procedures {ALL | *capture_procedure_name*…}}]

Path Delay Fault Model Options:
[*name_of_delay_path*… | -All] [-Both | -Rise | -Fall]

Power-Aware Options (applicable only after you have loaded CPF/UPF power data):
[-ON_domains | -OFf_domains | {-POWer_domains {*domain_name* …}}]
[-ISolation_cells] [-LEvel_shifters] [-REtention_cells]

Description

The add_faults command adds faults to the current fault list—either all faults specific to the fault model, or faults you specify by an object expression. By default, it also discards all

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

patterns in the current test pattern set and sets all faults to undetected (specifically, to the UC category).

When you enter Setup mode, the tool deletes all faults from the current fault list. Furthermore, if you change the fault type, it also deletes all faults. The power-aware options add faults based on power domains and power features after you have loaded a CPF or UPF file.

You cannot add faults for clock domains, capture procedures, and specific objects or paths within the same command instance. Use a separate command instance for each.

When the fault mode is "uncollapsed" (the invocation default), the tool adds all possible faults to the list. If you change the fault mode to "collapsed" with the set_fault_mode command, only one instance of any given fault is added, ignoring any equivalent faults.

Arguments

- -KEEP_Patterns

  An optional switch that preserves the current test pattern set and maintains the status of the current faults while adding faults. The default behavior is to discard the current test pattern set and set all faults to undetected.

- **-All**

  A required switch that adds all faults, depending on the fault model.

  - **Bridge Faults** — Adds all previously loaded bridge entries to the internal fault list. This is the default behavior.
  - **UDFM Faults** — Adds faults for all previously loaded UDFM fault sites to the internal fault list. This is the default behavior.
  - **Non-Bridge Faults** — Adds all faults on all model, netlist primitive, and top module pins. The setting of the set_internal_fault command affects the behavior of this switch. If you have not specified "set_internal_fault on", the switch adds only faults residing on the boundaries of library cells. If you have specified "set_internal_fault on", the -All switch adds faults that reside on the inputs and outputs of gates within library cells. For more information about fault locations, refer to "Fault Locations" in the *Tessent Scan and ATPG User's Manual*.

- ***name_of_delay_path***

  A required, repeatable string that specifies the names of the delay paths whose faults you want to add to the current fault list. The name must correspond to the name of a delay path defined in the path definition file.

- -Both | -Rise | -Fall

  An optional switch that specifies which faults to add for each path already added with the read_fault_sites command. Use these switches for path delay faults only.

  -Both — Adds both the slow-to-rise and slow-to-fall faults. This is the default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

-Rise — Adds only the slow-to-rise faults.

-Fall — Adds only the slow-to-fall faults.

- ***object_expression***

  A required, repeatable string that specifies a list of instances or pins whose faults you want added to the current fault list. For UDFM, you can specify only instances. You can use any number of asterisk (*) and question mark (?) wildcard characters in the expression.

  Instance pathnames must be Tessent Cell library cell instances. Pin pathnames must be Tessent Cell library cell instance pins, also referred to as design level pins. If the object expression specifies a pin within an instance of a Tessent Cell library model, the tool ignores it. By default, the tool matches pin pathnames first. If it does not find a pin pathname match, the tool next tries to match instance pathnames. Force the tool to match only pin pathnames or only instance pathnames by including the -PIN or -INstance switch after the *object_expression*.

- -PIN

  An optional switch that specifies to use the preceding object expression to match only pin pathnames; the tool adds faults for all the pins matched.

- -INstance

  An optional switch that specifies to use the preceding object expression to match only instance pathnames; the tool adds faults for all the pins on the instances matched.

- -MODule [-PIN *module_pin_pathname* ...] (Stuck/Transition/Toggle/Iddq only)

  A switch that specifies to interpret the ***object_expression*** argument as a module pathname. All instances of the module are affected. You can use the asterisk (*) and question mark (?) wildcards for the ***object_expression*** argument, and the tool adds the fault for all matching modules or library models.

  When you follow the -Module switch with the optional -PIN switch, you can specify a list of module pin pathnames for the pin within the module. The tool automatically finds all the instances with the given object_expression and adds faults for every pin specified in module_pin_pathname list.

- ***bridge_name***

  A required, repeatable string that specifies bridge names whose bridge faults you want added to the current fault list.

- ***net_pathname***

  A required, repeatable string that specifies single net names whose bridge faults you want added to the current fault list.

- ***instance_name***

  A required, repeatable string that specifies instances names whose bridge faults you want added to the current fault list.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -NAME

  An optional switch that specifies all entered strings are bridge names.

- -SINGle

  An optional switch that specifies all the entered strings are either a single net or instance. The tool searches for a net first. If it cannot find a corresponding net, the tool searches for a corresponding instance.

  If the string specifies a net, the tool adds faults to the fault list for each bridge entry associated with the net.

  If the string specifies an instance that contains a non-dominant net specified in a bridge entry, the tool adds associated faults to the fault list.

- -CLOCK_domains {ALL | *clock_pathname…*}

  A required switch and literal or repeatable string pair that specifies a list of clocks that the tool uses to decide the faults it adds to the fault list, subject to the following requirements:

  - **Static Faults** — The tool adds a fault to the fault list if it can be captured by any clock in the specified list of clocks or any of its equivalent clocks.
  - **Transition Faults** — The tool adds a fault to the fault list if one of the following is true:
    i. The launch and capture clock are the same clock from the specified list of clocks
    ii. The launch and capture clock are synchronous clocks from the specified list of clocks
    iii. The launch or capture clock from the specified are equivalent clocks

  When you use this switch, the tool ignores the constraint values implied in the data path when adding faults in the data path. However, the tool still considers the constraint value for the clock cone tracing to determine the state element clock domain.

  The argument choices are as follows:

  ALL — A literal that specifies all the clocks in the design.

  clock_pathname — A repeatable string that specifies individual clocks.

  Note

  A fault added using this switch might also be detectable by an unspecified clock.

- -NO_EQUivalent_clocks

  An optional switch that prevents the -Clock_domains switch from adding faults in equivalent clock domains.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains

  An optional switch used with -Clock_domains to instruct the tool to exclude the inter-clock faults for synchronous clocks. When you specify this switch, the tool considers only faults within clock domains when adding or deleting faults by clock domain.

- -exclude_constrained_clock_domains

  An optional switch used with -Clock_domains to exclude the faults in constrained clock domains. By default, the command includes the faults in constrained clock domains when you run "add_faults -clock_domains all."

- **-CAPture_procedures**{**ALL** | ***capture_procedure_name…***}

  A required switch and literal or repeatable string pair that specifies a list of enabled named capture procedures and directs the tool to add faults that are potentially detectable by any of the specified procedures. The argument choices are as follows:

  ALL — A literal that specifies all enabled named capture procedures.

  capture_procedure_name — A repeatable string that specifies a particular enabled named capture procedure.

  For information about enabling named capture procedures, see the set_capture_procedures command description.

- -FRom *pin…*

  An optional switch and repeatable string pair that sets the start pin or pins of the cone of logic the tool uses to select faults to add. When specified, the tool performs forward cone tracing from the specified pin or pins until it encounters a stopping point. You can specify one or more instance names rather than pin names. When you specify an instance name with -from, the tool uses its output pins when defining the cone.

- -THrough *pin…*

  An optional switch and repeatable string pair that sets the through pin or pins of the cone of logic the tool uses to select faults to add. When specified, the tool performs forward and backward cone tracing from the specified pin or pins until it encounters a stopping point in both directions. You can specify one or more instance names rather than pin names. When you specify an instance name with -through, the tool uses its output pins when defining the cone.

- -TO *pin…*

  An optional switch and repeatable string pair that sets the end pin or pins of the cone of logic the tool uses to select faults to add. When specified, the tool performs backward cone tracing from the specified pin or pins until it encounters a stopping point. You can specify one or more instance names rather than pin names. When you specify an instance name with -to, the tool uses its input pins when defining the cone.

  Note

  When you use -from, -to, and -through with other switches, the selected faults are the intersection of the cones traced for all specified switches.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -FROM_Clock *pin* ...

  An optional switch and repeatable string pair that sets the launch clock pin or pins of the cone of logic the tool uses to select faults to add. When specified, the tool performs forward cone tracing from the output ports of the state elements launched by the specified clock pin or pins until it encounters a stopping point.

- -TO_Clock *pin* ...

  An optional switch and repeatable string pair that sets the capture clock pin or pins of the cone of logic the tool uses to select faults to add. When specified, the tool performs backward cone tracing from the data ports of the state elements captured by the specified clock pin or pins until it encounters a stopping point.

- -Stuck_at {01 | 0 | 1}

  An optional switch and literal pair that specifies which stuck-at or transition faults to add to the fault list according to the following table:

	01 (default)	**0**	**1**
Stuck-At	stuck-at-0 stuck-at-1	stuck-at-0	stuck-at-1
Transition	slow-to-rise slow-to-fall	slow-to-rise	slow-to-fall

- -Udfm_type *name*

  A switch and string pair that specifies a UDFM type. Use this switch to add faults associated with a specified UDFM type. The name string is case insensitive and may include any number of asterisk (*) and question mark (?) wildcard characters.

- -CEll *name*

  A switch and string pair that specifies a library cell. Use this switch to add faults associated with a specified library cell. The name string is case insensitive and may include any number of asterisk (*) and question mark (?) wildcard characters.

- -MODule *name (UDFM only)*

  A switch and string pair that specifies a module. Use this switch to add faults associated with a specified module. The name string is case insensitive and may include any number of asterisk (*) and question mark (?) wildcard characters.

- -FAULT *name (UDFM only)*

  A switch and string pair that specifies a UDFM fault name. The name string is case insensitive and can include any number of asterisk (*) and question mark (?) wildcard characters.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -VERbose

  By default, when you specify wildcard characters for instance (-INstance) or pin (-PIN) names, the tool outputs a summary message similar to the following:

  ```
  // Note:  Adding faults for 330 fault sites.
  ```

  When you specify the optional -VERbose switch, the tool outputs the instance or pin names instead of the summary. You can optionally redirect this output to a file using the > or >> redirection operators.

  If you use actual instance or pin names instead of wildcards, then this switch has no effect.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Power-Aware Options (applicable only after you have loaded CPF/UPF power data)

- -ON_domains

  An optional switch that adds faults on all power-on domains.

- -OFF_domains

  An optional switch that adds faults on all power-off domains.

- -Power_domains {*domain_name* …}

  An optional switch and repeatable string that adds faults on the specified power domains.

- -Isolation_cells

  An optional switch that adds faults on all isolation cells.

- -Level_shifters

  An optional switch that adds faults on all level shifters.

- -REtention_cells

  An optional switch that adds faults on all retention cells.

Examples

Example 1

The following example adds all faults to the circuit so you can run an ATPG process:

```
set_system_mode analysis
create_patterns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example uses a prefix match to add faults only to pins on instances whose pathnames begin with “/u9/u2/LCT_reg”:

add_faults /u9/u2/LCT_reg*

```
=== Found 16 design pins ===
/u9/u2/LCT_reg_0_/A0
/u9/u2/LCT_reg_0_/A1
/u9/u2/LCT_reg_0_/S0
/u9/u2/LCT_reg_0_/Y
/u9/u2/LCT_reg_1_/A0
/u9/u2/LCT_reg_1_/A1
/u9/u2/LCT_reg_1_/S0
/u9/u2/LCT_reg_1_/Y
/u9/u2/LCT_reg_2_/A0
/u9/u2/LCT_reg_2_/A1
/u9/u2/LCT_reg_2_/S0
/u9/u2/LCT_reg_2_/Y
/u9/u2/LCT_reg_3_/A0
/u9/u2/LCT_reg_3_/A1
/u9/u2/LCT_reg_3_/S0
/u9/u2/LCT_reg_3_/Y
```

Example 3

The following example uses the same prefix match as the preceding example but includes the -Instance switch:

delete_faults -all
add_faults /u9/u2/LCT_reg* -instance

```
=== Found 4 design instances ===
/u9/u2/LCT_reg_0_
/u9/u2/LCT_reg_1_
/u9/u2/LCT_reg_2_
/u9/u2/LCT_reg_3_
```

When you use the -Instance switch, the command lists the instances whose pins are faulted as a result of the wildcard character rather than listing each pin.

Example 4

The following example demonstrates how to use a wildcard in the second level of hierarchy:

add_faults /u9/*/LCT_reg_0_ -instance

```
=== Found 4 design instances ===
/u9/u4/LCT_reg_0_
/u9/u3/LCT_reg_0_
/u9/u2/LCT_reg_0_
/u9/u1/LCT_reg_0_
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 5

The following example combines a wildcard character at the second level of hierarchy with a prefix match:

add_faults /u9/*/LCT_reg* -instance

```
=== Found 32 design instances ===
/u9/u4/LCT_reg_0_
/u9/u4/LCT_reg_1_
/u9/u4/LCT_reg_2_
/u9/u4/LCT_reg_3_
/u9/u4/LCT_reg_4_
/u9/u4/LCT_reg_5_
/u9/u4/LCT_reg_6_
/u9/u4/LCT_reg_7_
/u9/u3/LCT_reg_0_
/u9/u3/LCT_reg_1_
/u9/u3/LCT_reg_2_
...
/u9/u2/LCT_reg_5_
/u9/u2/LCT_reg_6_
/u9/u2/LCT_reg_7_
/u9/u1/LCT_reg_0_
/u9/u1/LCT_reg_1_
/u9/u1/LCT_reg_2_
/u9/u1/LCT_reg_3_
/u9/u1/LCT_reg_4_
/u9/u1/LCT_reg_5_
/u9/u1/LCT_reg_6_
/u9/u1/LCT_reg_7_
```

Example 6

The following example adds only faults in the power domain named PD1:

add_faults -power_domain PD1

Example 7

The following example uses a prefix match to add faults to pins on instances whose pathnames begin with “/u9/u2/LCT_reg”, plus all level shifters in the design:

add_faults /u9/u2/LCT_reg* -level_shifters

Example 8

The following example adds intra_cell_bridges UDFM fault models associated with all library cells that start with “mux” to the current fault list.

add_faults -all -udfm_type intra_cell_bridges -cell mux*

Example 9

The following example adds intra_cell_bridges UDFM fault models to the instance my_module1_ins/my_module1_or02. When you add a UDFM fault model, you do not need the -instance switch.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
add_faults my_module1_ins/my_module1_or02 -udfm_type intra_cell_bridges
```

Example 10

The following example adds fault models to every module in the current fault list whose name begins with adder.

```
add_faults -all -module adder*
```

Example 11

The following example displays the fault value on the specified gate_pin after the add_faults command runs:

```
add_nofaults ha1/ddd/q -Pin -Stuck 1
add_nofaults ha1/ddd/clk -Pin -Stuck 0
add_nofaults ha1/ddd/d -Pin -Stuck 01
....
add_faults -all
....

foreach_in_collection i [get_gate_pins ha1/ddd/* ] {
        puts "[get_name_list $i] fault_site: [get_attribute_value_list $i -name fault_site]"

}
```

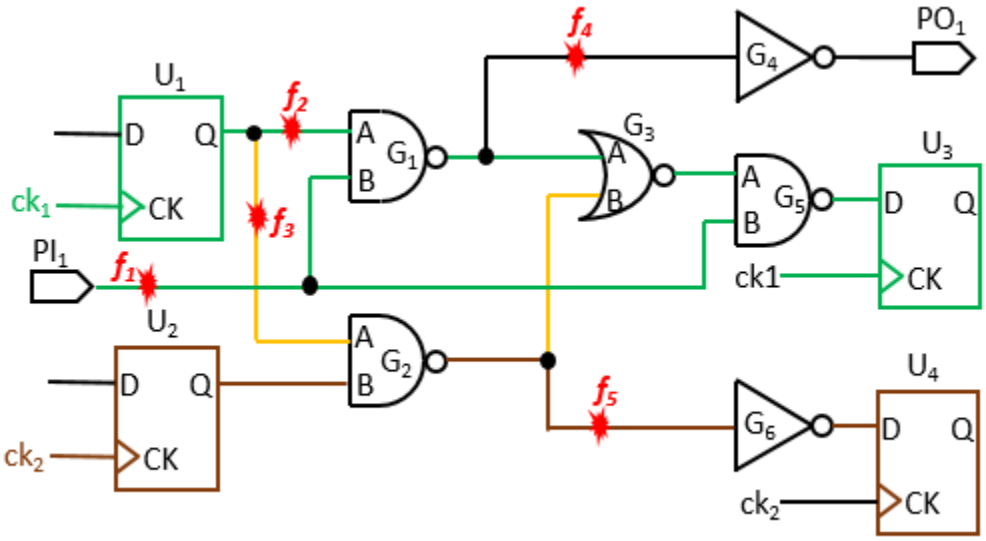
```
ha1/ddd/clk   fault_site: 1
ha1/ddd/d   fault_site: none
ha1/ddd/q   fault_site: 0
```

Example 12

The following example illustrates using the -clock_domains switch with stuck-at faults:

```
add_faults -clock_domains ck1 ck2
```

Figure 3-17. Static Fault



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

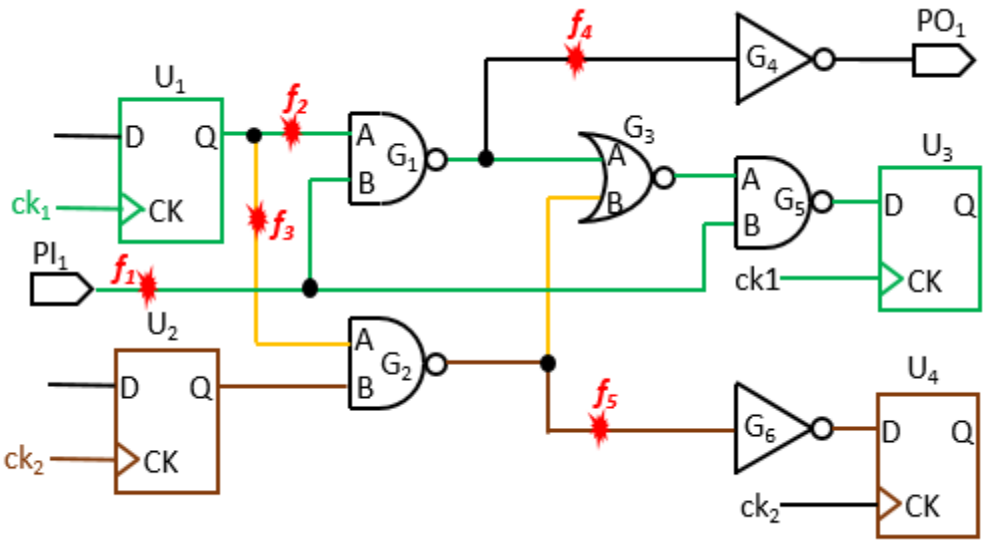
This adds the faults f_1, f_2, f_3, and f_5.

Example 13

For this example, assume that f_1, f_2, f_3, f_4, and f_5 are legitimate fault sites. Other pins are no-faulted. Use the -clock_domains switch with transition faults and without defining a synchronous clock group. In this case, ck1 and ck2 are incompatible.

```
add_faults -clock_domains ck1 ck2
```

Figure 3-18. Transition Fault - Incompatible Clocks



The added faults are f_2, and f_5. In this case, the command does not add f_3 because clk1 (launch clock) and clk2 (capture clock) are asynchronous.

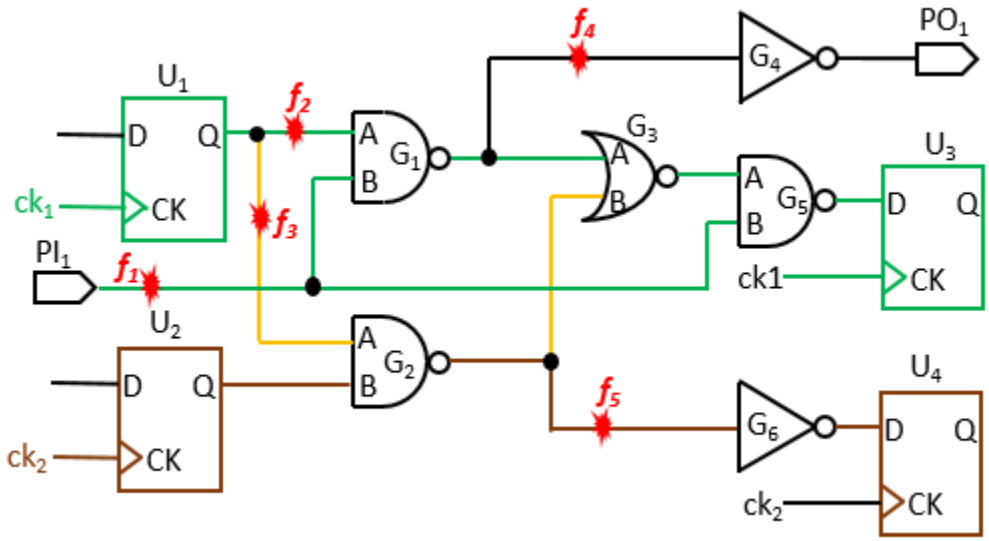
Example 14

The following example illustrates using the -clock_domains switch with transition faults, after defining a synchronous clock group. In this case, ck1 and ck2 are compatible:

```
add_synchronous_clock_group {ck1 ck2}
...
add_faults -clock_domains ck1 ck2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 3-19. Transition Fault - Compatible Clocks



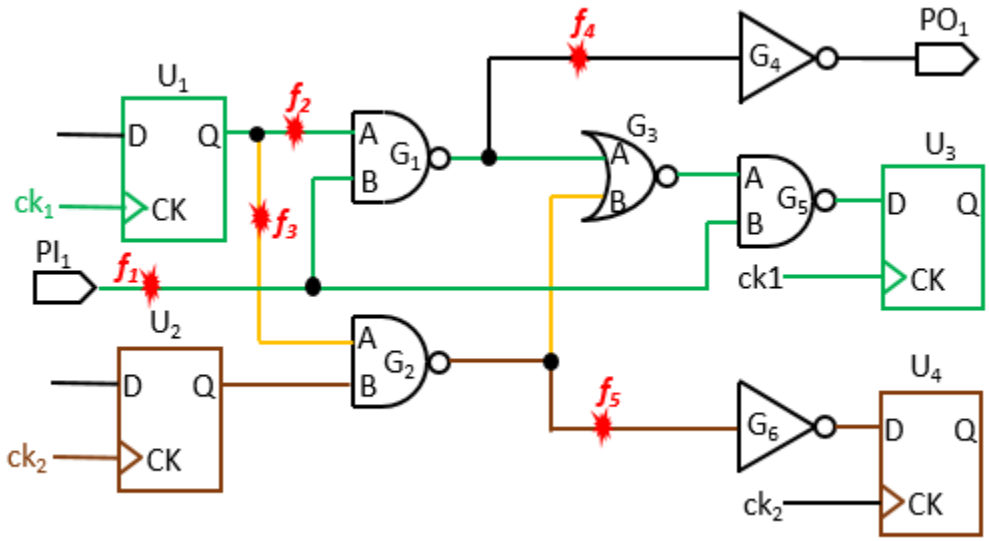
The added faults, launched and captured by ck1 and ck2, are f_2, f_3, and f_5. You can use the add_faults option -exclude_faults_between_synch_clock_domains to exclude cross-domain faults, such as f_3.

Example 15

For this example, assume that f_1, f_2, f_3, f_4, and f_5 are legitimate fault sites. Other pins are no-faulted. Use the -clock_domains switch with transition faults and without defining equivalent pins:

```
add_faults -clock_domains ck1
```

Figure 3-20. Transition Fault - No Equivalent Pins



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

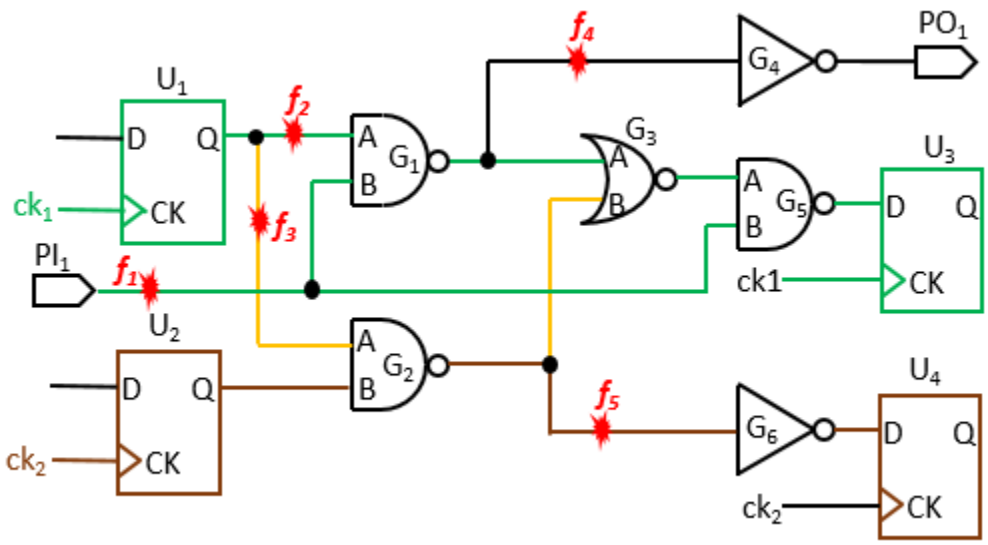
The added fault is f_2.

Example 16

For this example, assume that f_1, f_2, f_3, f_4, and f_5 are legitimate fault sites. Other pins are no-faulted. Use the -clock_domains switch with transition faults and with pin equivalents defined for ck1 and ck2:

```
add_input_constraints -equivalent ck1 ck2
...
add_faults -clock_domains ck1
```

Figure 3-21. Transition Fault - Equivalent Pins



The added faults are f_2, f_3, and f_5. You can use the add_faults option -no_equivalent_clocks to prevent the addition of faults in equivalent clock domains. In this case, the command does not add faults f_3 and f_5.

Related Topics

create_patterns

delete_faults

read_fault_sites

read_faults

read_cpf

read_upf

report_faults

report_testbench_simulation_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_fault_mode

set_fault_sampling

set_fault_type

set_internal_fault

write_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_gate_to_rtl_mapping

Context: all contexts

Mode: setup

Adds rules for post-synthesis/post-layout to RTL instance name-mapping for DEF files.

Usage

add_gate_to_rtl_mapping **-instance_name_map_list *string_pair_list***

Description

This command enables you to add extra name-mapping rules for post-synthesis/post-layout to RTL name mapping for DEF file instances. Normally, the read_def command is capable of handling the mapping of all instances from the DEF file to those in the RTL design without any user intervention. The add_gate_to_rtl_mapping, report_gate_to_rtl_mapping, and delete_gate_to_rtl_mapping commands can be used to create and manage additional mapping rules as needed in cases where the read_def command failed to map gate-level DEF file instances.

Arguments

- **-instance_name_map_list *string_pair_list***

  A required switch and string pair list that specifies the instance name mapping rules to be added. The *string_pair_list* must contain an even number of string entries, where the first string of the pair is the gate instance name in the DEF file and the second string is the RTL instance name in the design.

Examples

The DEF file A.def lists the following memory instances that did not map to RTL instances during the read_def command execution:

```
A/B[0].pool/mem1
A/B{0].pool/mem2
A/B[1].pool/mem3
A/B[1].pool/mem4
```

These DEF instances correspond to the following RTL instances in the design:

```
A/\B[0].pool/mem1
A/\B[0].pool/mem2
A/\B[1].pool/mem3
A/\B[1].pool/mem4
```

The following add_gate_to_rtl_mapping command can be used to map the DEF instances to the RTL instances in the design:

```
add_gate_to_rtl_mapping -instance_name_map_list {
                        {B[0].pool} {\B[0].pool}
                        {B[1].pool} {\B[1].pool} }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

delete_gate_to_rtl_mapping

report_gate_to_rtl_mapping

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_icl_ports

Context: patterns -ijtag, dft

Mode: setup, insertion

Specifies top design ports that are added in the ICL file generated during ICL extraction.

Usage

add_icl_ports ***port_objects*** [-type *ijtag_function*] [-function_modifier *modifier*] [-source *source_port_object* [-inverted] | -trace] [-forced_low_dft_signal_list *forced_low_dft_signals*] [-forced_high_dft_signal_list *forced_high_dft_signals*]

Description

Specifies top design ports that are added in the ICL file generated during ICL extraction. The ports can be of any ICL port type listed in the ICL Port Types table below.

Table 3-14. ICL Port Types

ICL Port Type	IJTAG Function	Direction
DataInPort	data_in	Input
DataOutPort	data_out	Output
ScanInPort	scan_in	Input
ScanOutPort	scan_out	Output
ClockPort	clock	Input
ToClockPort	to_clock	Output
ResetPort, ActivePolarity 1	reset	Input
ToResetPort, ActivePolarity 1	to_reset	Output
ResetPort, ActivePolarity 0	reset_n	Input
ToResetPort, ActivePolarity 0	to_reset_n	Output
TCKPort	tck	Input
ToTCKPort	to_tck	Output
TMSPort	tms	Input
ToTMSPort	to_tms	Output
TRSTPort	trst	Input
ToTRSTPort	to_trst	Output
CaptureEnPort	capture_en	Input

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-14. ICL Port Types (cont.)

ICL Port Type	IJTAG Function	Direction
ToCaptureEnPort	to_capture_en	Output
ShiftEnPort	shift_en	Input
ToShiftEnPort	to_shift_en	Output
UpdateEnPort	update_en	Input
ToUpdateEnPort	to_update_en	Output
SelectPort	select	Input
ToSelectPort	to_select	Output
AddressPort	address	Input
WriteEnPort	write_en	Input
ReadEnPort	read_en	Input
IRSelectPort	ir_select	Input
ToIRSelectPort	to_ir_select	Output

You must specify the current design with the set_current_design command before issuing this command. Changing the current design discards all previously specified input and output ICL ports. The tool validates that the ports specified as input ports are input or bidirectional ports, and ports specified as output ports are bidirectional or output ports.

Note

You must load the design (gate-level Verilog or RTL) before you issue the set_current_design command as the tool verifies that the specified ports exist in the portlist. You may also load the top-level design as a black-box module.

During ICL extraction, top level ports in the controlling fan-in of input ports of sub-modules are automatically inferred as input ports, and top level ports in the controlling fanout of output ports of sub-modules are automatically inferred as output ports. Use this command if you want the top level ICL file to declare input or output ports even when they do not have any input or output ports in their fan-in or fanout. Later on, when the current design is integrated into a higher module, ICL extraction traces the newly created input and output ports and ijtag retargeting can be used to retarget iWrite and iRead commands to the boundary of the new top module.

The IJTAG function “to_clock” plays a special role. Ports of the current design with this IJTAG function serve as starting points for ICL Extraction tracing, and the tool enforces proper connectivity of those ports. If there is no appropriate driver for the newly created ToClockPort of the top module (ToClockPort of another ICL instance, ClockPort of the top module, internal clock), an I2 violation is reported.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***port_objects***

  A required string that specifies a Tcl list of one or more design port names or a collection of one or more design ports. An error is generated if the specified ports are not found in the portlist of the design.

  The ports in the collection must not have been used as arguments of the -source switch of another add_icl_port command before. (A port that is already in use as a "source port", cannot be modified by means of add_icl_ports.)

- -type *ijtag_function*

  An optional switch and literal pair that specifies the function that the ports have in the generated ICL module. The currently supported IJTAG functions are listed in the ICL Port Types table. If the -type option is not specified, the ICL port type is inferred as DataInPort for input ports and DataOutPort for output ports. In case of bidirectional ports, the ICL port type is inferred as DataInPort, if the -source option is omitted, and as DataOutPort, if the -source option is specified. An error is generated if the specified type conflicts with the actual port direction.

- -function_modifier *modifier*

  An optional switch and literal pair that specifies the function modifier that the ports have in the generated ICL module. See the function_modifier icl_port attribute for more information.

- -source *source_port_object*

  An optional switch and value pair that specifies the top level port that is used as source of the **port_objects** in the ICL file generated during ICL Extraction. This switch can be used to enforce feedthroughs (direct connections between a primary input port and a primary output port), which otherwise would not be created by ICL Extraction, because ICL Extraction does not start tracing at the top level ports.

  The port specified as *source_port_object* must have been added as an ICL port by means of the add_icl_ports command before.

- -inverted

  An optional switch that specifies that the connections between the *source_port_object* and the **port_objects** shall be inverting.

- -trace

  An optional switch that specifies that ICL Extraction should trace from the specified primary input port or primary output port. It is mutually exclusive with the -source switch. This switch can be used to trigger the extraction of connections that would otherwise not be found, for example because of the usage of "set_icl_extraction_options -ignore_muxes_at_data_out_ports on" in combination with DataOutPorts that fan out to primary outputs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -forced_low_dft_signal_list *forced_low_dft_signals*

  An optional switch that triggers the creation of the associated attribute on the ICL port in the extracted ICL. The parameter forced_low_dft_signals specifies a list of signal names to assert low during tracing from this port.

- -forced_high_dft_signal_list *forced_high_dft_signals*

  An optional switch that triggers the creation of the associated attribute on the ICL port in the extracted ICL. The parameter forced_high_dft_signals specifies a list of signal names to assert low during tracing from this port.

Examples

Example 1

The following example declares an ICL DataInPort port D1, and three ICL DataOutPort ports O1 O2 and O3. The generated ICL module includes “DataInPort D1;”, “DataOutPort O1;”, “DataOutPort O2;” and “DataOutPort O3;” even if no ICL ports traced to or from them in the design.

```
set_current_design top
add_icl_ports D1 -type data_in
add_icl_ports {O1 O2 O3} -type data_out
```

Example 2

The following example declares two ICL host ScanInterfaces and an ICL client ScanInterface and enforces the control ports of the host ScanInterfaces to be driven directly by the control ports of the client ScanInterface. Parts of the SelectPorts, ToSelectPorts, ScanInPorts and ScanOutPorts are assumed to be created and connected automatically.

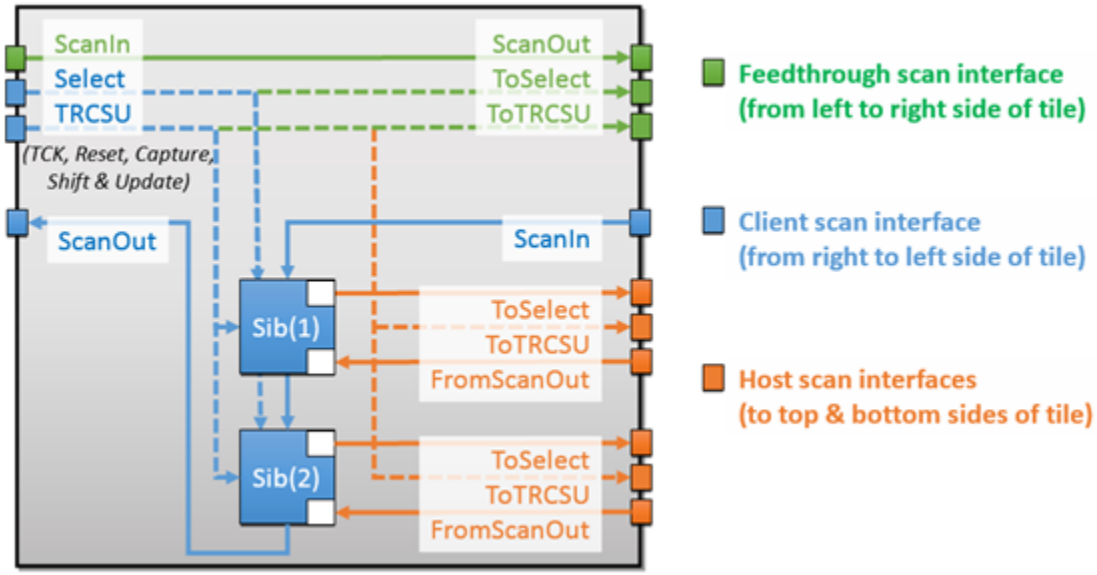
```
add_icl_scan_interfaces {host1 host2 client}
set_icl_scan_interface_ports -name host1 -ports {toCe1 toSe1 toUe1 toSel1 fromSo1 toSi1}
set_icl_scan_interface_ports -name host2 -ports {toCe2 toSe2 toUe2 toSel2 fromSo2 toSi2}
set_icl_scan_interface_ports -name client -ports {ce se ue sel si so}
add_icl_ports ce -type capture_en
add_icl_ports se -type shift_en
add_icl_ports ue -type update_en
add_icl_ports si -type scan_in
add_icl_ports {toCe1 toCe2} -type to_capture_en -source ce
add_icl_ports {toSe1 toSe2} -type to_shift_en -source se
add_icl_ports {toUe1 toUe2} -type to_update_en -source ue
add_icl_ports {toSi1 toSi2} -type scan_out -source si
```

Example 3

This example adds two SIBs inside a block and connects them into a ring between the si/so of the client interface. This same concept can be generalized to a greater number of scan interfaces if required. Two host ports are then created below each SIB to open up the path to for tiling rows.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 3-22. Example Scan Interfaces



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_context dft -rtl
read_verilog ./RTL/coreb.v
set_current_design coreb
add_icl_ports ijtag_tck -type tck
set_design_level physical_block
set_system_mode analysis
// The below procedure creates & connects ports of scan interfaces host_top<n>
proc connect_top_SIBs {root_wrapper args} {
  for {set id 1} {$id <= 2} {incr id} {
    foreach port_name {ijtag_tck ijtag_reset ijtag_ce ijtag_se \
                       ijtag_ue ijtag_si} {
      set port [create_port to_${port_name}_top${id} -direction output]
      create_connections coreb_rtl_tessent_sib_${id}_inst/${port_name} $port
    }
    delete_connections coreb_rtl_tessent_sib_${id}_inst/ijtag_from_so
    set port [create_port from_ijtag_so_top${id} -direction input]
    create_connections coreb_rtl_tessent_sib_${id}_inst/ijtag_from_so $port
    set port [create_port to_ijtag_sel_top${id} -direction output]
    create_connections coreb_rtl_tessent_sib_${id}_inst/ijtag_to_sel $port
  }
}
// The above callback procedure then gets invoked automatically after
// process_dft_spec
register_callback process_dft_specification.post_insertion connect_top_SIBs

// The below procedure declares a logical connection between feed_si and feed_so;
// It is very important, as the tools must trace through this logical connection
// later on, e.g. when generating patterns for a higher-level module
proc add_feedthrough_path { args } {
  # The next line can only work in 2017.4 or more recent versions
  #   as it requires the ijtag_logical_connection attribute to be R/W in all modes
  set_attribute_value [get_current_design -icl] -name ijtag_logical_connection \
                      -value { {feed_si feed_so} }
  write_icl ./tsdb_outdir/dft_inserted_designs/coreb_rtl.dft_inserted_design/\
            coreb.icl -replace
}
// The above callback procedure then gets invoked automatically just before saving
// ICL
register_callback extract_icl.pre_write add_feedthrough_path

set dft [create_dft_spec]
// Let's just insert two SIBs here - the process can be generalized anyway
read_config_data -from_string {
  DftSpecification(coreb,rtl) {
    IjtagNetwork {
      HostScanInterface (client) {
        Sib(2) {
        }
        Sib(1) {
        }
      }
    }
  }
}
report_config_data $dft
process_dft_specification

// The add_icl_ports command is then used to declare the newly-added scan
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// interface ICL ports
for {set id 1} {$id <= 2} {incr id} {
  add_icl_scan_interfaces host_top${id}
  foreach {port function} {reset to_reset_n sel to_select ce to_capture_en \
                           se to_shift_en ue to_update_en} {
    add_icl_ports to_ijtag_${port}_top${id} -type $function
  }
  add_icl_ports to_ijtag_si_top${id} -type scan_out
  add_icl_ports from_ijtag_so_top${id} -type scan_in
  set_icl_scan_interface_ports -name host_top${id} -ports [get_ports *_top${id}] \
                               -allowed_no_source
}
extract_icl

set pat [create_patterns_spec]
report_config_data $pat
process_patterns_spec

run_testbench_simulations
check_testbench_simulations -report_status

exit
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following is the extracted ICL output for the coreb module in this example:

```
//-------------------------------------------------
//  File created by: Tessent Shell
//          Version: 2017.4
//       Created on: Mon Nov 27 13:43:32 PST 2017
//-------------------------------------------------


Module coreb {
   // Created by ICL extraction
   ToTCKPort feed_tck;
   ScanInPort from_ijtag_so_top1 {
      Attribute connection_rule_option = "allowed_no_source";
   }
   ScanInPort from_ijtag_so_top2 {
      Attribute connection_rule_option = "allowed_no_source";
   }
   CaptureEnPort ijtag_ce;
   ResetPort ijtag_reset {
      ActivePolarity 0;
   }
   ShiftEnPort ijtag_se;
   SelectPort ijtag_sel;
   ScanInPort ijtag_si;
   ScanOutPort ijtag_so {
      Source coreb_rtl_tessent_sib_2_inst.ijtag_so;
   }
   TCKPort ijtag_tck;
   UpdateEnPort ijtag_ue;
   ToCaptureEnPort to_ijtag_ce_top1 {
      Attribute connection_rule_option = "allowed_no_destination";
   }
   ToCaptureEnPort to_ijtag_ce_top2 {
      Attribute connection_rule_option = "allowed_no_destination";
   }
   ToResetPort to_ijtag_reset_top1 {
      ActivePolarity 0;
      Attribute connection_rule_option = "allowed_no_destination";
   }
   ToResetPort to_ijtag_reset_top2 {
      ActivePolarity 0;
      Attribute connection_rule_option = "allowed_no_destination";
   }
   ToShiftEnPort to_ijtag_se_top1 {
      Attribute connection_rule_option = "allowed_no_destination";
   }
   ToShiftEnPort to_ijtag_se_top2 {
      Attribute connection_rule_option = "allowed_no_destination";
   }
   ToSelectPort to_ijtag_sel_top1 {
      Source coreb_rtl_tessent_sib_1_inst.ijtag_to_sel;
      Attribute connection_rule_option = "allowed_no_destination";
   }
   ToSelectPort to_ijtag_sel_top2 {
      Source coreb_rtl_tessent_sib_2_inst.ijtag_to_sel;
      Attribute connection_rule_option = "allowed_no_destination";
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
}
ScanOutPort to_ijtag_si_top1 {
   Attribute connection_rule_option = "allowed_no_destination";
}
ScanOutPort to_ijtag_si_top2 {
   Source coreb_rtl_tessent_sib_1_inst.ijtag_so;
   Attribute connection_rule_option = "allowed_no_destination";
}
ToTCKPort to_ijtag_tck_top1 {
   Attribute connection_rule_option = "allowed_no_destination";
}
ToTCKPort to_ijtag_tck_top2 {
   Attribute connection_rule_option = "allowed_no_destination";
}
ToUpdateEnPort to_ijtag_ue_top1 {
   Attribute connection_rule_option = "allowed_no_destination";
}
ToUpdateEnPort to_ijtag_ue_top2 {
   Attribute connection_rule_option = "allowed_no_destination";
}
ScanInterface client {
   Port ijtag_ce;
   Port ijtag_reset;
   Port ijtag_se;
   Port ijtag_sel;
   Port ijtag_si;
   Port ijtag_so;
   Port ijtag_tck;
   Port ijtag_ue;
}
ScanInterface host_top1 {
   Port from_ijtag_so_top1;
   Port to_ijtag_ce_top1;
   Port to_ijtag_reset_top1;
   Port to_ijtag_se_top1;
   Port to_ijtag_sel_top1;
   Port to_ijtag_si_top1;
   Port to_ijtag_tck_top1;
   Port to_ijtag_ue_top1;
}
ScanInterface host_top2 {
   Port from_ijtag_so_top2;
   Port to_ijtag_ce_top2;
   Port to_ijtag_reset_top2;
   Port to_ijtag_se_top2;
   Port to_ijtag_sel_top2;
   Port to_ijtag_si_top2;
   Port to_ijtag_tck_top2;
   Port to_ijtag_ue_top2;
}
Attribute icl_extraction_date = "Mon Nov 27 13:43:32 PST 2017";
Attribute ijtag_logical_connection = " {feed_si feed_so} ";
Attribute created_by_tessent_icl_extract = "true";
Attribute tessent_design_id = "rtl";
Attribute tessent_design_level = "physical_block";
Attribute tessent_is_physical_module = "true";
Instance coreb_rtl_tessent_sib_1_inst Of coreb_rtl_tessent_sib_1 {
   InputPort ijtag_reset = ijtag_reset;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
         InputPort ijtag_sel = ijtag_sel;
         InputPort ijtag_si = ijtag_si;
         InputPort ijtag_ce = ijtag_ce;
         InputPort ijtag_se = ijtag_se;
         InputPort ijtag_ue = ijtag_ue;
         InputPort ijtag_tck = ijtag_tck;
         InputPort ijtag_from_so = from_ijtag_so_top1;
         Attribute tessent_design_instance = "coreb_rtl_tessent_sib_1_inst";
      }
      Instance coreb_rtl_tessent_sib_2_inst Of coreb_rtl_tessent_sib_1 {
         InputPort ijtag_reset = ijtag_reset;
         InputPort ijtag_sel = ijtag_sel;
         InputPort ijtag_si = coreb_rtl_tessent_sib_1_inst.ijtag_so;
         InputPort ijtag_ce = ijtag_ce;
         InputPort ijtag_se = ijtag_se;
         InputPort ijtag_ue = ijtag_ue;
         InputPort ijtag_tck = ijtag_tck;
         InputPort ijtag_from_so = from_ijtag_so_top2;
         Attribute tessent_design_instance = "coreb_rtl_tessent_sib_2_inst";
      }
   }
```

Related Topics

create_dft_specification

delete_icl_ports

iRead

iWrite

set_current_design

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_icl_scan_interfaces

Context: patterns -ijtag, dft

Mode: setup, insertion (dft context only)

Adds the specified scan interfaces to the new ICL top module that is created during ICL extraction.

Usage

add_icl_scan_interfaces ***name_list*** [-chains *chain_name_list*]

Description

You only need to use this command if the top module on which ICL extract is running has more than one ScanInterface. In most cases, even when the top module has more than one ScanInterface, ICL extract is able to infer them correctly from the connections to and from the lower-level ICL modules. Only in complicated cases, the heuristic used by ICL extraction may fail; in this case, you use this command to describe the ScanInterface of the top module and they are reflected in the generated ICL of the top module.

Every ScanInterface must have a port list, so after using the add_icl_scan_interfaces command, use the set_icl_scan_interface_ports command to set the list of ports for each ScanInterface.

You must set the current design with set_current_design before using the add_icl_scan_interfaces command.

Arguments

- ***name_list***

  A required list that specifies the names of scan interfaces that ICL extraction creates in the new ICL top module.

- -chains *chain_name_list*

  An optional switch and value pair that specifies the list of scan chain names used to create client scan interfaces with multiple chains. If you specify this argument, the scan interfaces are created with the specified scan chains.

Note

Every ScanInterface must have a port list. After using the add_icl_scan_interfaces command, use the set_icl_scan_interface_ports command to set the list of ports for each ScanInterface. If the -chains argument is used, the set_icl_scan_interface_ports command must be called multiple times: once to specify the common ports of the scan interface, and once for each of the scan chains to specify their scan_in and scan_out port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

In the following example, the add_icl_scan_interfaces command creates ScanInterfaces named TAP and Internal in the new ICL top module during ICL extraction. The set_icl_scan_interface_ports command adds the names of the ports to the TAP and Internal ScanInterfaces.

```
set_current_design tap1
add_icl_scan_interfaces {TAP Internal}
set_icl_scan_interface_ports -name TAP -ports {tck tdi tdo tms trst}
set_icl_scan_interface_ports -name Internal -ports {tdo tdrEn1 fromTdr1 ce se ue}
```

The resulting ICL model would look like this:

```
Module tap1 {

   TCKPort         tck;
   ScanInPort      tdi;
   ScanOutPort     tdo       {  Source IRMux; }
   TMSPort         tms;
   TRSTPort        trst;
   ToSelectPort    tdrEn1    {  Source sel1;  }
   ScanInPort      fromTdr1;
   ToCaptureEnPort ce;
   ToShiftEnPort   se;
   ToUpdateEnPort  ue;

   ScanInterface TAP {Port tck; Port tdi; Port tdo; Port tms; Port trst;}
   ScanInterface Internal {Port tdo; Port tdrEn1; Port fromTdr1; Port ce;
                           Port se; Port ue;}
```

Example 2

In the following example, the add_icl_scan_interfaces command is used to create a client scan interface with two scan chains.

```
set_current_design top
add_icl_scan_interfaces { intf_1 } -chains { chain_1 chain_2 }
set_icl_scan_interface_ports -name intf_1 -ports { ce se ue }
set_icl_scan_interface_ports -name intf_1 -ports { si_1 so_1 } -chain chain_1
set_icl_scan_interface_ports -name intf_1 -ports { si_2 so_2 } -chain chain_2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The resulting ICL model looks like this:

```
Module top{
   ScanInPort si_1;
   ScanInPort si_2;
   ScanOutPort so_1 {
      Source i1.so;
   }
   ScanOutPort so_2 {
      Source i2.so;
  }
   CaptureEnPort ce;
   ShiftEnPort se;
  UpdateEnPort ue;
   ScanInterface intf_1 { Port ce; Port se; Port ue;
      Chain chain_1 { Port si_1; Port so_1; }
      Chain chain_2 { Port si_2; Port so_2; }
   }
```

Related Topics

delete_icl_scan_interfaces

get_icl_scan_interface_list

get_icl_scan_interface_port_list

set_current_design

set_icl_scan_interface_ports

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_icl_ssn_datapaths

Context: all contexts

Mode: setup, insertion

Specifies the SSN datapath identifiers that Tessent Shell will associate with certain ICL ports of the new ICL top module during ICL Extraction.

Usage

add_icl_ssn_datapaths ***name_list*** [-type {inout | input | output}]

Description

The bus data ports and the clock ports of a SSN datapath must be associated with this datapath by means of the icl_port attribute tessent_ssn_datapath_ids. The position of a bus data port within the bus must be described by means of the icl_port attribute tessent_ssn_datapath_bus_index.

When the top module ICL is elaborated, Tessent Shell analyzes the existing SSN datapaths and attaches the attributes to the ports, if possible and unambiguous. Usually, Tessent Shell is capable of deducing the attribute values from the topology of the SSN datapaths and from the existing SSN datapath attributes deeper down the ICL hierarchy. However, sometimes the association between the ICL top module ports and the SSN datapaths is ambiguous, for example, if there are extra output paths. In this case, the commands add_icl_ssn_datapaths and set_icl_ssn_datapath_ports must be used to instruct the tool during ICL extraction to attach the appropriate attributes to the ICL ports of the newly-created ICL top module.

The command add_icl_ssn_datapaths introduces new SSN datapath identifiers. Only SSN datapath names that have been registered by means of add_icl_ssn_datapaths can later be used in the set_icl_ssn_datapath_ports command.

Arguments

- ***name_list***

  A Tcl list of names that are registered as SSN datapath identifiers and that will later be used as values in the icl_port attribute tessent_ssn_datapath_ids. Names can contain only numbers, letters, and underscores. Including other characters in names in the list causes the tool to report an error.

- -type {inout | input | output}

  An optional switch and literal pair that specifies the type of the datapath identifiers. If the type is set to “inout” (the default), the tool uses the identifiers in the <*name_list*> as ordinary SSN datapath identifiers. In this case, they uniquely identify the SSN datapath in commands like set_ssn_datapath_options, and they inform the tool which outputs and inputs are intended for default access to the SSN. The ICL ports that belong to datapaths with this type of identifier have the attribute ‘tessent_ssn_datapath_id = "<*identifier*>"’ after ICL extraction.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the type is set to “input”, the tool uses the identifiers in the <*name_list*> as “alternate input identifiers,” and the ICL ports affected by the command “set_icl_ssn_datapath_ports -name <*identifier*>” (with <*identifier*> as one of the elements in <*name_list*>) have the attribute ‘tessent_ssn_datapath_input_id = "<*identifier*>"’ after ICL extraction.

If the type is set to “output”, the tool uses the identifiers in the <*name_list*> as “alternate output identifiers,” and the ICL ports affected by the command “set_icl_ssn_datapath_ports -name <*identifier*>” (with <*identifier*> as one of the elements in <*name_list*>) have the attribute ‘tessent_ssn_datapath_output_id = "<*identifier*>"’ after ICL extraction.

For more information about the use model for alternate input identifiers and alternate output identifiers, refer to set_ssn_datapath_options.

Examples

```
SETUP> add_icl_ssn_datapaths {A B}
SETUP> set_icl_ssn_datapath_ports -name A -clock_inputs bus_clock \
     -bus_data_inputs bus_a_in[11:0] -bus_data_outputs bus_a_out[11:0]
SETUP> set_icl_ssn_datapath_ports -name B -clock_inputs bus_clock \
     -bus_data_inputs bus_b_in[23:0] -bus_data_outputs bus_b_out[23:0]
SETUP> extract_icl
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_iddq_exceptions

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables you to specify individual sites where the tool does not apply the Iddq restrictions.

Usage

add_iddq_exceptions ***ZVAL_gate…***

Description

You can achieve maximum quiescence by using the appropriate IDDQ checks, and adding exceptions for as few sites as possible while still obtaining patterns that are not rejected for violation of IDDQ checks during fault simulation.

For more information on IDDQ, refer to "IDDQ Test Set Creation" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- ***ZVAL_gate…***

  A required repeatable string that declares the gate name or gate identification number of a site for which the tool could report Iddq check violations. (See the set_iddq_checks command for specifics about setting Iddq checks). These sites are always roots of tri-state buses (ZVAL gates at the primitive level).

  Note

  If a site does not have a unique name, you must provide a gate identification number.

  The "report_gates -type zval" command gives a list of all candidates, but typically you only need to use the gate identification numbers reported by the analyze_restrictions command.

Examples

The following example adds Iddq restrictions, uses the analyze_restrictions command to locate the sources of violations, then adds an Iddq exception to remove one of them.

set_iddq_checks -int_float
simulate_patterns
analyze_restrictions

```
...
//  Will attempt to locate and report specific -int_float violators.
//  No single bus is constrained to Z (or already Z).  Will analyze
//  further.
//  The following ZVAL gates cause -int_float to fail by themselves.
//    ZVAL gate /cpu/alu/U2392/ (54417).  ABORTed.
//    ZVAL gate /cpu/alu/U4177/ (63712).  ABORTed.
...
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_iddq_exceptions /cpu/alu/U2392/

To use the analyze_restrictions command most effectively, you must have design knowledge; in some cases, design knowledge is required in order to pinpoint the source of the violation.

Related Topics

analyze_restrictions

report_gates

report_primary_outputs

set_iddq_checks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_ijtag_logical_connection

Context: patterns -ijtag

Mode: setup

Creates a new logical connection path through the current design by specifying the source and destination of the logical path.

Usage

add_ijtag_logical_connection **-from *source_pin_or_port_name***
[-from_pin_direction {input | output}]
-to *destination_pin_or_port_name*
[-to_pin_direction {input | output}]
[-inverted]

Description

During ICL connection tracing, logical connections are followed in preference to the actual connections. When tracing reaches an instance pin with a logical connection, it follows the logical connection across the design hierarchy. A forward trace only follows logical connections that start on the pin being traced and then skip to the destination of the logical connection. A backward trace only follows logical connections that have a destination on the pin being traced and then skip to the source of the logical connection.

See "Example 1" on page 297 for an example of a logical connection.

Use this command with care as it basically bypasses the structural tracing of the circuit and uses an act of faith on the behavior of the specified logical source. If the behavior turns out to be inconsistent with the specified logical sources, you only detect the issue in simulation. Use this command to model logical sources through blackbox modules or to model a path where the logic is too complex. A circuit that gates TCK with capture and shift enable provides proper capture and shift behavior but would not be properly traceable by ICL extraction. In such a case, using a logical source to enable tracing through the tck gating logic is needed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following attributes are used to store the source...destination relationship for the two ends of the logical connection:

Table 3-15. Logical Connection Attributes

Attribute Name	Usage
ijtag_logical_hier_connection_from_src	Associated with the top-level module port or instance pin that is the destination of a hierarchical logical connection. A string list of top level module port names and hierarchical instance pin path names to the source ends of these logical connections. This attribute is not inherited but does cause the hierarchical boundary to be preserved.
ijtag_logical_hier_connection_from_src_inv	Associated with the top-level module port or instance pin that is the destination of a hierarchical logical connection. A string list of top level module port names and hierarchical instance pin path names to the source ends of these logical connections. This attribute is not inherited but does cause the hierarchical boundary to be preserved.
ijtag_logical_hier_connection_to_dst	Associated with the top-level module port or instance pin that is the source of a hierarchical logical connection. A string list of top level module port names and hierarchical instance pin path names to the destination ends of these logical connections. This attribute is not inherited but does cause the hierarchical boundary to be preserved.

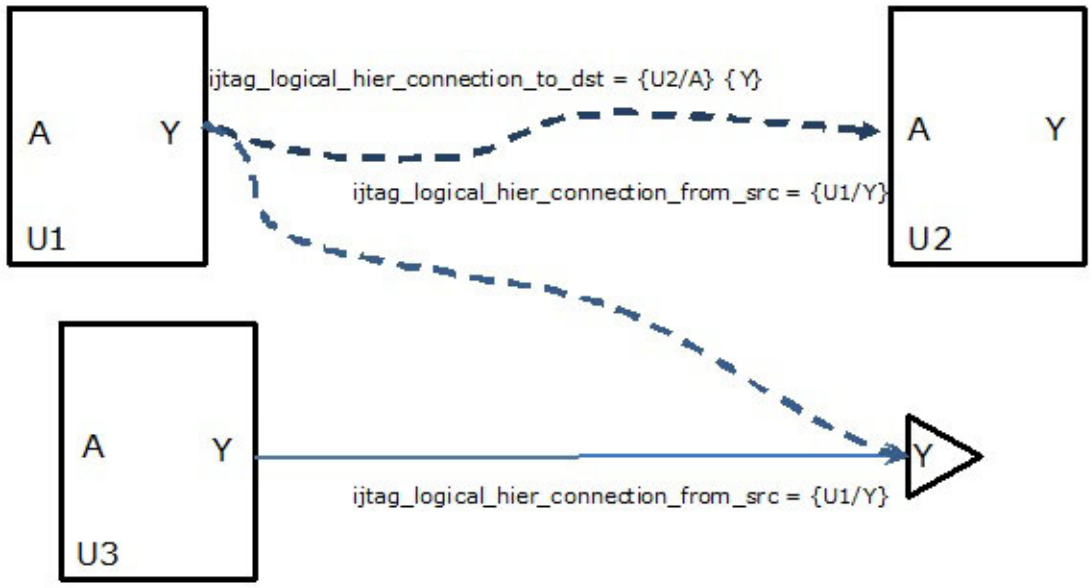
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-15. Logical Connection Attributes (cont.)

Attribute Name	Usage
ijtag_logical_hier_connection_to_dst_inv	Associated with the top-level module port or instance pin that is the source of a hierarchical logical connection. A string list of top level module port names and hierarchical instance pin path names to the destination ends of these logical connections. This attribute is not inherited but does cause the hierarchical boundary to be preserved.

The following figure illustrates the usage of these attributes.

Figure 3-23. Logical Connection Attributes



These attributes are available for direct read only introspection by the user, like the other predefined ICL extraction attributes. They are all cleared when the set_current_design command is run.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- **-from *source_pin_or_port_name***

  A required switch and value pair that specifies the source point of the logical connection, which should be a top level module port or a hierarchical instance pin in the current top design module.

- -from_pin_direction {input | output}

  An optional switch-value pair that enables you to specify the pin direction of the source of the ijtag_logical_connection, if the pin specified by the "-from" switch is a bidirectional pin.

- **-to *destination_pin_or_port_name***

  A required switch and value pair that specifies the destination point of the logical connection, which should be a top level module port or a hierarchical instance pin in the current top design module.

- -to_pin_direction {input | output }

  An optional switch-value pair that enables you to specify the pin direction of the destination of the ijtag_logical_connection, if the pin specified by the "-to" switch is a bidirectional pin.

- -inverted

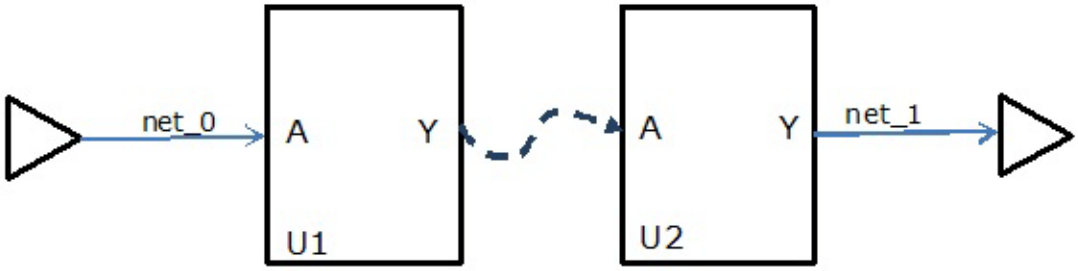
  An optional switch that specifies that the logical connection is inverted.

Examples

Example 1

The following example illustrates a logical connection from pin U1/Y to pin U2/A. A forward trace that arrives at pin U1/Y continues on from pin U2/A. A backward trace that arrives at pin U2/A continues on from pin U1/Y.

Figure 3-24. Logical Connection Example



At the netlist level all interconnections start at an instance pin or a top level module port and end at an instance pin or top level module port. Establishing a logical connection within the design hierarchy does not alter the way that design hierarchy is traversed by ICL extraction tracing when that design hierarchy is used as part of a larger design hierarchy, from a different top-level module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The add_ijtag_logical_connections command creates logical connections between the pins /i2/in and /i2/out, /i3/in and /i3/out and i4/in and /i4/out, and an inverted logical connection between /i5/in and /i5/out. The report_ijtag_logical_connections command reports all logical connections in the current design.

add_ijtag_logical_connection -from /i2/in -to /i2/out

add_ijtag_logical_connection -from /i3/in -to /i3/out

add_ijtag_logical_connection -from /i4/in -to /i4/out

add_ijtag_logical_connection -from /i5/in -to /i5/out -inverted

report_ijtag_logical_connections

```
//  IJTAG Logical Connections:
//  From Source  To Destination   Inversion
//  ===========  ==============   =========
//  /i2/in       /i2/out          no
//  /i3/in       /i3/out          no
//  /i4/in       /i4/out          no
//  /i5/in       /i5/out          yes
```

Related Topics

delete_ijtag_logical_connection

report_ijtag_logical_connections

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_input_constraints

Context: all contexts

Mode: setup

Constrains primary inputs to specified values.

Usage

Context: dft -scan and dft -test_points

add_input_constraints
{{[-ALl | *primary_input_pin_name*… | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs]
constraint_switch} |
{[-EQuivalent *reference_pin* [*equivalent_pin*]… [-INvert *inverted_pin*…]]
[**constraint_switch**]}}

Context: patterns -scan

add_input_constraints
{{[-ALl | *primary_input_pin_name*… | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs]
constraint_switch} |
{[-EQuivalent *reference_pin* [*equivalent_pin*]… [-INvert *inverted_pin*…]]
[**constraint_switch**]}}
[-HOld] [-NO_Z] [-SLow_pad [-CELL *model_name*]]

Context: patterns -ijtag

add_input_constraints
{{[-ALl | *primary_input_pin_name*… | -BIDI_Only | -BIDI_Exclude]
constraint_switch} |
{[-EQuivalent *reference_pin* [*equivalent_pin*]… [-INvert *inverted_pin*…]]
[**constraint_switch**]}}

Description

The add_input_constraints command constrains primary input pins to specified values.

Note

Do not force a constrained pin to a new value in any cycle of a named capture procedure (NCP), even temporarily. To ensure state stability, a constrained pin's state must be kept constant for all cycles of NCPs.

In the patterns -ijtag context, you can constrain all Verilog ports unless the port is also an ICL port. Input constraints on ICL ports are enabled only for ICL ports of type DataInPorts, TRSTPort, and ClockPort. For more information, see "How to Constrain Inputs" in the *Tessent IJTAG User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Inferred input constraints

Some input constraints are automatically applied to certain input ports of the current design, depending on the current context and the ICL port function. There are also attributes that trigger the creation of input constraints.

Constraints derived from ICL port functions

If there is an ICL description for the current design, then the following constraints are applied during the processing of the command set_current_design in all contexts except “patterns -ijtag”:A port with ICL port function TMSPort is constrained to C0. A port with ICL port function TRSTPort is constrained to C1 unless there is already a CT1 constraint for that port. Ports with ICL port function ResetPort are constrained to their inactive values, using the constraint types “C0” or “C1”. All ports with ICL port functions CaptureEnPort, ShiftEnPort and UpdateEnPort are constrained to C0, unless the associated ICL ScanInterfaces contain ports with ICL port function SelectPort, in which case only the SelectPorts are constrained to C0. When the context is set to “patterns -ijtag”, the constraints are automatically removed. They are reapplied if the context is set to any other context but “patterns -ijtag”.

Constraints derived from ICL module attributes

If the ICL module, which is associated with the current design, contains the “forced_low_input_port_list” or “forced_high_input_port_list” icl_module attributes, then the ports referenced by those attributes are constrained appropriately, using the constraint types “C0” and “C1”. The constraints are applied during the processing of the command “set_current_design”.

Constraints derived from port attributes

Ports with the Port attribute “function” set to “power” or “ground” are constrained appropriately, using the constraint types “CT1” and “CT0”. The constraints are applied during the processing of the command “set_current_design”.

Constraints on ports with ICL port function DataInPort

At the end of a test_setup procedure that contains iCall or iMerge statements, all ports with ICL port function DataInPort are constrained to C0 or C1, according to the last values that have been assigned to them during the IJTAG retargeting of the iCall and iMerge commands. This behavior is suppressed, if the port has the icl_module attribute “tessent_no_input_constraints” set to “on”.

Arguments

- -ALl

  An optional switch that applies the constraint to all applicable primary input pins except clocks, read controls, and write controls. If -Slow_pad is also specified, the constraint is only applied to primary input pins that can take a Z value. This is the default.

- *primary_input_pin_name*

  An optional, repeatable string that specifies the name of a primary input pin to which to apply the constraint.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -BIDI_Only

  An optional switch that specifies to apply the constraint to bidirectional pins only.

- -BIDI_Exclude

  An optional switch that specifies to apply the constraint only to primary input pins that are not bidirectional.

- -SCAN_INputs

  An optional switch that specifies to apply the constraint to scan input pins only.

- -EQuivalent *reference_pin* [*equivalent_pin*]…

  An optional switch, string, and repeatable string triplet that specifies for the *primary_input_pin_name*s to take on the value of the *reference_pin*. If you optionally specify one or more of the *primary_input_pin_name*s as *equivalent_pin*s, those PIs in the *equivalent_pin* list take on the value of the *reference_pin*.You must list pins you want to be equivalent in value to the reference pin before the -Invert argument when used.

- -INvert *inverted_pin*…

  An optional switch and repeatable string that together with the -Equivalent switch specify for the primary input pins, *inverted_pin*, to take on the inverted value of the *reference_pin*. You must list the pins you want to be inverted in value after the -Invert argument.

- ***constraint_switch***

  A required switch that specifies the constant value to which to constrain the primary input pin(s). This switch is required unless you specify the -EQuivalent switch.

  The choices, from which you can specify only one, are as follows:

  -C0 — Specifies application of the constant 0 to the chosen primary input pin.

  The tools do not use constant 0 constraints to determine tied circuitry. Therefore, faults associated with line holds resulting from these constraints are classified as ATPG_untestable (AU), as opposed to Tied (TI) or Blocked (BL). Use a -C0 constraint if the constraint is valid in test mode, but not valid in system mode.

  Note

  If the constraint is valid in system mode as well, consider using -CT0 instead. Faults associated with tied circuitry resulting from a -CT0 constraint are not classified as ATPG_untestable, thereby improving test coverage compared to using a -C0 constraint. To ensure the improvement in coverage matches reality, however, be sure to use -CT0 only for valid system mode constraints.

  -C1 — Specifies application of the constant 1 to the chosen primary input pin.

  The tools do not use constant 1 constraints to determine tied circuitry. Therefore, faults associated with line holds resulting from these constraints are classified as ATPG_untestable (AU), as opposed to Tied (TI) or Blocked (BL). Use -C1 if the constraint is valid in test mode, but not valid in system mode.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

If the constraint is valid in system mode as well, consider using -CT1 instead. Faults associated with tied circuitry resulting from a -CT1 constraint are not classified as ATPG_untestable, thereby improving test coverage compared to using a -C1 constraint. To ensure the improvement in coverage matches reality, however, be sure to use -CT1 only for valid system mode constraints.

-CX — Specifies application of the constant X (unknown) to the chosen primary input pin. The tools do not use the specified primary input pin for control.

-CZ — Specifies application of the constant Z (high-impedance) to the chosen primary input pin. The CZ constraint is only applicable to bidi pin and PI that fans out only to Z consumer gates, such as PO, or bus. For a PI that fans out to non Z consumer gate, when CZ is added the tool reports a warning message during the flattening process and convert CZ constraint into CX constraint automatically. For example:

```
// Warning: Invalid constant Z pin constraint on pin te5 changed to
// constant X.
```

-CT0 — Specifies application of the constant TIE0. The ATPG process treats this constraint as if you had added a tied signal.

The tools classify faults associated with tied circuitry resulting from a -CT0 pin constraint as untestable (Tied (TI) or Blocked (BL)). These faults are not included in the calculation of test coverage, so they neither raise nor lower coverage. If instead, the faults were associated with a -C0 pin constraint, the tool would classify them as ATPG_untestable (AU) and they would reduce coverage. Therefore, it is advantageous to use a -CT0 instead of a -C0 constraint; but take care that any -CT0 constraint you apply for test is valid in system mode—to ensure the relative boost in coverage is real.

Note

You cannot override a -CT0 value (from a test procedure file for example). If you need this flexibility, use -C0 instead.

-CT1 — Specifies application of the constant TIE1. The ATPG process treats this constraint as if you had added a tied signal.

The tools classify faults associated with tied circuitry resulting from a -CT1 pin constraint as untestable (Tied (TI) or Blocked (BL)). These faults are not included in the calculation of test coverage, so they neither raise nor lower coverage. If instead, the faults were associated with a -C1 pin constraint, the tool would classify them as ATPG_untestable (AU) and they would reduce coverage. Therefore, it is advantageous to use a -CT1 instead of a -C1 constraint; but take care that any -CT1 constraint you apply for test is valid in system mode—to ensure the relative boost in coverage is real.

Note

You cannot override a -CT1 value (from a test procedure file for example). If you need this flexibility, use -C1 instead.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

-CTZ — Specifies application of the constant TIEZ. The ATPG process treats this constraint as if you had added a tied signal.

The tools classify faults associated with tied circuitry resulting from a -CTZ pin constraint as untestable (Tied (TI) or Blocked (BL)). These faults are not included in the calculation of test coverage, so they neither raise nor lower coverage. If instead, the faults were associated with a -CZ pin constraint, the tool would classify them as ATPG_untestable (AU) and they would reduce coverage. Therefore, it is advantageous to use a -CTZ instead of a -CZ constraint; but take care that any -CTZ constraint you apply for test is valid in system mode—to ensure the relative boost in coverage is real.

- -HOld

  An optional switch that, for clock sequential patterns only, holds the constant constraint value at the primary input(s) for all cycles of a clock_sequential or multiple_load pattern. Even if a pattern has multiple loads, the primary input's value cannot change among different load intervals. The hold does not apply to clocks, read controls and write controls. It also does not apply to launch off shift patterns, RAM sequential patterns, or patterns with a clock procedure.

  If the fault type is set to transition and you enter a "set_transition_holdpi On" command, the constraint defined by the add_input_constraints command is overridden by the set_transition_holdpi command; that is, all primary inputs except clocks and read/write controls are going to hold their values even if you have not applied a constraint to a particular primary input. Similar behavior occurs when the fault type is set to path delay and you enter a "set_pathdelay_holdpi On" command.

- -NO_Z

  An optional switch that specifies not to permit a Z value at the specified primary input pin(s) during test generation, even if the pins can support a Z value. During good machine simulation, a pattern is rejected if it assigns a Z value to any primary input that is specified to have no Z value. This switch is invalid with either the -CZ or -CTZ switch.

- -SLow_pad

  An optional switch that specifies to X out the value at a bidirectional pin if the pin is driven from within during test generation.

  -CELL *model_name*

  An optional switch and string pair that together with the -Slow_pad switch specifies the scope of the bidirectional pins to be included as slow pads. Only the bidirectional pins that drive the instances initiated from the specified library cell name are added as slow pads.

Examples

Example 1

The following example shows two differential clocks, CLK1 off state 0 and CLK2 off state 1. Because the pins are equivalent, you only need to add the clock to one of the pins. The

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

command order is important: ***you must*** first define equivalent pins, then define one of them as the clock, otherwise the tool reports an error. The tool infers CLK2 to be a clock.

add_input_constraints -equivalent CLK1 -invert CLK2

add_clocks 0 CLK1

```
// Note: Equivalent pin CLK2 also added to clock list.
```

Example 2

The following example shows how to add equivalent clocks that are inverted.

SETUP> add_clocks 0 mode[1] -pulse_always

SETUP> add_clocks 1 mode[2] -pulse_always

SETUP> report_clocks

```
User-defined Clocks (2):
========================

  Sync and Async Source Clocks
  ============================
   -------  ---------  ------------  --------
   Name     Off State   Constraints  Internal
   -------  ---------  ------------  --------
   mode[2]     1      Pulse always     No
   mode[1]     0      Pulse always     No
```

SETUP> add_input_constraints -equivalent mode[1] -inv mode[2]

SETUP> report_input_constraints

```
//  Primary Input  Equivalent  C-Type  Hold      No-Z      Slow-Pad
//  -------------  ----------  ------  ----  ------------  --------
//  mode[1]                     NONE    No   Undetermined     No
//                 ~mode[2]     NONE    No   Undetermined     No
```

Example 3

This example specifies multiple equivalent pins: clk, clk1, and clk2. Pin clk1 is equivalent to the reference pin, clk. Pin clk2 is inverted from the reference pin, clk.

SETUP> add_input_constraints -equivalent clk clk1 -invert clk2

SETUP> add_clocks 0 clk

```
// Note: Equivalent pin clk1 and clk2 are also added to clock list.
```

SETUP> report_clocks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
User-defined Clocks (3):
========================
  Sync and Async Source Clocks  ============================
   ---------  ------  ---------  -----------  --------
   Name       Label   Off State  Constraints  Internal
   ---------  ------  ---------  -----------  --------
   'clk'      clk         0                      No
   'clk2'     clk2        1                      No
   'clk1'     clk1        0                      No
```

Related Topics

delete_input_constraints

get_input_constraints

report_input_constraints

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_layout_core_information

Context: patterns -scan_diagnosis

Mode: analysis

In hierarchical layout-aware diagnosis, adds core instance information to an existing core-level LDB.

Usage

add_layout_core_information {-def *list_of_DEF_files* | -deflist *tcl_list* |
{-design *top_design_name* -die_area {*die_area*} -instance *instance* -placement {*x y*} -orientation *orientation*} }

Description

In hierarchical layout-aware diagnosis, adds core instance information to an existing core-level LDB. This information can come from three sources, which are mutually exclusive: a space-separated list of top-level DEF files, a Tcl list of top-level DEF files, or the actual core instance data chip-mapped to the top level.

To use this command you must have a valid LDB that you previously opened using the open_layout command. Normally, you also have top-level DEF files that instantiate the core represented by the currently opened LDB, and the DEF file for the core represented by the currently opened LDB. If you only have core-level DEF files, you can manually add chip-mapped core instance information.

When run, the command extracts the instance information for the core represented by the LDB from the top-level DEF file(s) and stores this information in the LDB. This makes the LDB instance-aware, which is a requirement for hierarchical layout-aware diagnosis.

This command enables you to store core instance information from multiple SoC designs in the same LDB, which is helpful when the same core is used is multiple designs.

See "Diagnosis for Hierarchical Design" in the *Tessent Diagnosis User's Manual* for more information.

Arguments

- -def *list_of_DEF_files*

  A switch and string pair that specifies a space-separated list of top-level DEF files that contain placement and orientation information for each core.

- **-deflist** *tcl_list*

  A switch and string pair that specifies a Tcl list of top-level DEF files that contain placement and orientation information for each core.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -design *design_name* -die_area {*die_area*} -instance *instance* -placement {*x y*} -orientation *orientation*

  A series of switches that must be specified together. Use these switches to manually add the instance information for a core to an existing LDB. You must specify separate add_layout_core_information commands for each core you want to manually add.

  -design *top_design_name*: The name of the top module of the design.

  -die_area {*die_area*}: Four integers that specify the X,Y coordinates of the core with origin at 0,0. The first and second integers specify the bottom-left corner of the core, and the third and fourth integers specify the top-right corner of the core.

  -instance *instance*: The core instance name.

  -placement {*x y*}: Integers that represent the x, y placement of the specified core in the top-level DEF, where x and y indicate the bottom right corner of the core.

  -orientation *orientation*: A string that specifies the orientation of the core instance, where the possible choices are: { N | W | S | E | FN | FW | FS | FE }.

Examples

The following example adds two core instances of coreA to an existing LDB.

```
add_layout_core_information top_chip -die_area {-4000 -4000 20000000 40000000} \
   -instance cpu_coreA -placement {2000 2000} -orientation N

add_layout_core_information top_chip -die_area {-2000 -2000 10000000 20000000} \
   -instance cpu_coreA -placement {50 100} -orientation FS
```

Related Topics

delete_layout_core_information

report_layout_core_information

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_lfsr_connections

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Connects an external pin or an internal pin to a Linear Feedback Shift Register (LFSR).

Usage

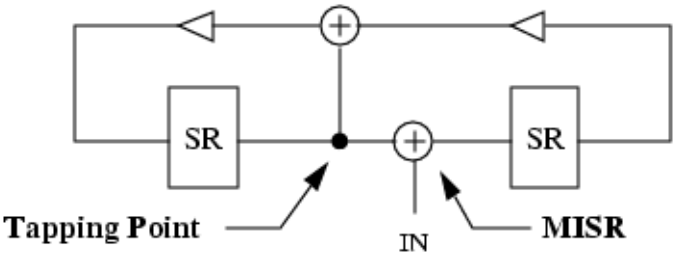
add_lfsr_connections ***pin_pathname lfsr_name position***…

Description

The add_lfsr_connections command connects a core logic pin to an LFSR. You specify this pin with the *pin_pathname* argument. LFSR bit positions have integer numbers, where 0 indicates the least significant bit position. The tool assumes that the output of the 0 bit position connects to the input of the highest bit position. If you select multiple bits of a Pseudo-Random Pattern Generator (PRPG) for the *position* argument, the tool assumes they are all exclusive-ORed together to create the value for the pin.

If you determine that multiple pin names must connect to a bit position of a Multiple Input Signature Register (MISR), you must issue a separate add_lfsr_connections command for each pin. The tool assumes the pins are all exclusive-ORed together to create the value for the next MISR input. The tool also assumes that the physical placement of the MISR connections is after the tapping points as shown in Figure 3-25.

Figure 3-25. MISR Placement



You can use the report_lfsrs command to display all the LFSRs with their current values and tap positions.

Arguments

- ***pin_pathname***

  A required string that specifies the pin pathname (<*instance pathname*>/<*pinname*>) of the core logic pin that you want to connect to the LFSR specified by *lfsr_name*.

- ***lfsr_name***

  A required string that specifies the name of the LFSR to which you want to connect the *pin_pathname*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ***position***

  A required repeatable integer that specifies the bit positions of the *lfsr_name* at whose outputs you want to place connections. A bit position is an integer number, where 0 indicates the least significant bit position. The tool assumes the output of the 0 bit position connects to the input of the highest bit position.

Examples

The following example connects an LFSR to a scan-in pin and another LFSR to a scan-out pin:

```
add_lfsrs lfsr1 prpg 5 10 -serial -in
add_lfsrs misr1 misr 5 15 -both -out
add_lfsr_taps lfsr1 1 3
add_lfsr_taps misr1 1 2
add_lfsr_connections scan_in.1 lfsr1 2
add_lfsr_connections scan_out.0 misr1 3
```

Related Topics

add_lfsrs

add_lfsr_taps

delete_lfsr_connections

report_lfsr_connections

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_lfsr_taps

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Adds the tap configuration to a Linear Feedback Shift Register (LFSR) or a PRPG of type cellular automata (CA).

Usage

add_lfsr_taps ***lfsr_name position***…

Description

The add_lfsr_taps command sets the tap configuration of an LFSR. LFSR bit positions have integer numbers, where 0 indicates the least significant bit position. The tool assumes the output of the 0-bit position connects to the selected tap points and that the 0-bit position cannot itself be a tap point.

If the LFSR is defined as a CA, the tap positions define the position of the self-feedbacks cells (those CA cells defined using rule 150). At least one tap must be defined; otherwise, the consistency check of the PRPG issues an error message.

You can use the report_lfsrs command to display all of the LFSRs with their current values and tap positions. You can change the default setting of the *tap_type* switches by using the set_lfsrs command.

Arguments

- ***lfsr_name***

  A required string that specifies the name of the LFSR on which you want to place the taps.

- ***position***

  A required, repeatable integer that specifies the bit positions of the *lfsr_name* at whose outputs you want to place the taps or, for CAs, the bit positions of the self-feedback cells.

Examples

Example 1

The following example places taps on the newly-added LFSRs:

```
add_lfsrs lfsr1 prpg 5 10 -serial -in
add_lfsrs misr1 misr 5 15 -both -out
add_lfsr_taps lfsr1 1 3
add_lfsr_taps misr1 1 2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example defines a 32-bit PRPG with self-feedback cells at positions 3, 7, 12, 15, 23, 25, and 30:

```
add_lfsrs my_prpg prpg 32 FFFFFFFF -CA
add_lfsr_taps my_prpg 3 7 12 15 23 25 30
```

Related Topics

add_lfsrs

delete_lfsr_taps

report_lfsrs

set_lfsrs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_lfsrs

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Adds LFSRs for use as PRPGs or MISRs.

Usage

add_lfsrs ***lfsr_name*** {**Prpg** | **Misr**} ***length seed*** [-Both | -Serial | -Parallel]
[-Out | -In | -Ca [-INCrement_step *value*]]

Description

The add_lfsrs command defines LFSRs, which the tool uses as PRPGs, to create pseudorandom values for the BIST patterns or as MISRs to compact responses.

You specify the LFSR's shift technique by using one of the following shift_type switches: -Both, -Serial, or -Parallel. You specify the placement of the exclusive-OR taps by using one of the following *tap_type* switches: -Out or -In; or you specify the type of LFSR as cellular automata using the *tap_type* switch -Ca. You can change the default setting of the *shift_type* and *tap_type* switches by using the set_lfsrs command.

Arguments

- ***lfsr_name***

  A required string that specifies the name that you want to assign to the LFSR. Note that you cannot use the keyword "all" for this name.

- **Prpg**

  A literal that indicates the LFSR functions as a PRPG.

- **Misr**

  A literal that indicates the LFSR functions as a MISR.

- ***length***

  A required integer, greater than 1, that specifies the number of bits in the LFSR.

- ***seed***

  A required, right-justified, hexadecimal number, greater than 0, that specifies the initial state of the LFSR.

- -Both

  An optional switch specifying that the LFSR shifts both serially and in parallel. This is the default unless you change it with the set_lfsrs command.

- -Serial

  An optional switch specifying that the LFSR shifts serially the number of times equal to the length of the longest scan chain for each scan pattern.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Parallel

  An optional switch specifying that the LFSR parallel shifts once for each scan pattern.

  The following lists the three *tap_type* switches of which you can only choose one.

- -Out

  An optional switch specifying that the exclusive-OR taps reside outside the register path. This is the default unless you change it with the set_lfsrs command.

- -In

  An optional switch specifying that the exclusive-OR taps reside in the register path.

- -CA

  An optional switch specifying that the PRPG is of type cellular automata. This switch is only valid if the LFSR type is specified as **Prpg**.

- -INCrement_step *value*

  An optional switch that specifies the number of times the PRPG is advanced for each shift cycle. The *value* argument is a required integer.

Examples

Example 1

The following example defines an LFSR to be a PRPG and another LFSR to be a MISR:

```
add_lfsrs lfsr1 prpg 5 10 -serial -in
add_lfsrs misr1 misr 5 15 -both -out
add_lfsr_taps lfsr1 1 3
add_lfsr_taps misr1 1 2
```

Example 2

The following example defines an LFSR to be a PRPG without a phase shifter:

```
add_lfsrs my_prpg prpg 24 FFFFFF -CA
add_lfsr_taps my_prpg 3 7 12 15 23
add_lfsrs connection scanin_1 my_prpg 0
add_lfsrs connection scanin_2 my_prpg 1
...
add_lfsrs connection scanin_24 my_prpg 23
```

Example 3

The following example defines an LFSR to be a PRPG with a phase shifter:

```
add_lfsrs my_prpg prpg 24 FFFFFF -CA
add_lfsr_taps my_prpg 3 7 12 15 23
add_lfsr_connections scanin_1 my_prpg 0 3 5
add_lfsr_connections scanin_2 my_prpg 1 7 11
...
add_lfsr_connections scanin_24 my_prpg 9 15 23
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

The following example defines an LFSR to be a PRPG and specifies to advance the PRPG eight times for each shift cycle.

```
add_lfsrs my_prpg prpg 24 FFFFFF -CA -inc 8
add_lfsr_taps my_prpg 3 7 12 15 23
add_lfsrs connection scanin_1 my_prpg 0
add_lfsrs connection scanin_2 my_prpg 1
...
add_lfsrs connection scanin_24 my_prpg 23
```

Related Topics

add_lfsr_taps

add_lfsr_connections

delete_lfsrs

report_lfsrs

set_lfsrs

set_lfsr_seed

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_lists

Context: dft -edt, patterns -scan

Mode: analysis

Adds pins to the list of pins on which to report.

Usage

add_lists ***pin_pathname***...

Description

The add_lists command enables you to create and add to a list of pins that the tool reports on during simulation.

In “dft -edt” context, you cannot report values on gates in the EDT logic with this command because values inside the EDT logic are not always accurate.

Tip

This command is a useful debugging aid, as it provides a way to check the values simulated for specific output pins of cells within the design.

To list all the pins on which the tool currently reports during simulation, use the report_lists command. To turn off reporting of a pin’s simulation values, use the delete_lists command. To direct the tool to write the pins’ simulation values to a file instead of displaying them on screen, use the set_list_file command.

When switching to Setup mode, the tool discards all pins from the report list.

Note

Multiprocessing does not support the add_lists command. If you use this command in distributed mode, the tool disables distribution of simulation, which may result in increased run time. When this occurs, or if you use the add_processors command after you have run the add_lists command, the tool reports the following warning:

```
//  Warning: The use of the add_lists command prevents the use of
//           distribution and multithreading during simulation, which may
//           have a noticeable effect on performance.
```

Note

The add_lists command is not supported in the diagnosis context and is automatically disabled. As a result, the tool does not report any simulation values on pins that were added using the add_lists command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***pin_pathname***

  A required, repeatable string that specifies the output pins on which to report during good or faulty machine simulation.

Examples

The following example writes to the file, *pinlist_goodsim.txt*, the values simulated for two output pins during simulation of an external pattern source:

```
set_system_mode analysis
read_patterns pattern1
add_lists /i_1006/o /i_102/o
set_list_file pinlist_goodsim.txt -replace
simulate_patterns
```

The following shows the format of the resulting list file: the first column lists the pattern index of each pattern simulated, the second column shows the value simulated on each monitored pin for each pattern. Note that the pattern index refers to the simulated parallel patterns rather than the effective patterns that are stored after ATPG.

```
//        /i_1006/o
//        |/i_102/o
//        ||
//        ||
//----------
//     0  11
//     1  10
//     2  00
//     3  10
//     4  01
//     5  01
...
```

Notice that in analysis system mode, the report shows the good value of the pin. The tools also shows the values at each test pattern.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following is example list file content for the same two pins when simulated in analysis system mode:

```
//        /i_1006/o
//        |/i_102/o
//        ||
//        ||
//----------
//     0  11
//     0  0- { /i_1006/o@0 }
//     1  10
//     1  0- { /i_1006/o@0 }
//     2  00
//     2  1- { /i_1006/o@1 }
//     3  10
//     3  0- { /i_1006/o@0 }
//     4  01
//     4  1- { /i_1006/o@1 }
//     5  01
//     5  1- { /i_1006/o@1 }
...
```

The report now includes two lines of information for each pattern simulated. The first line shows the good machine value of each pin as before. The second line shows the faulty value of each pin if different from the good value (a "-" indicates the faulty value was the same as the good value). Inside the "{}" is the fault site and its faulty machine stuck-at value the tool simulated to produce the faulty machine results.

Related Topics

delete_lists

report_lists

set_list_file

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_loadboard_loopback_pairs

Context: all contexts

Modes: setup

Prerequisites: Design must be elaborated prior to using this command.

Creates a loopback connection from specified output ports to specified input ports.

Usage

add_loadboard_loopback_pairs **-inputs** {***input_object_spec***} **-outputs** {***output_object_spec***} [-ac_delay_to_z *ac_delay_to_z*]

Description

This command creates a loopback connection from specified output ports to corresponding specified input ports. Loopback connections that are specified on compatible ICL ports are also added to the ICL data model during the next system mode transition into analysis mode. The following ICL port pairs are compatible with loopback connections: ScanOutPort to ScanInPort, DataOutPort to DataInPort and ToClockPort to ClockPort. An ac_delay time can be specified for the connections.This command provides direct connections of the DC and the CouplingCapacitor connections in the simulation testbenches of custom pattern (using open_pattern_set, close_pattern_set, and write_patterns).

Arguments

- -inputs {***input_object_spec***}

  A required switch and string pair that specifies the input ports for a loopback connection. The ***input_object_spec*** string can specify one or more input ports that get paired with an output port specified in the -outputs ***output_object_spec*** switch. If the input port count does not match the output port count, the tool issues an error.

- **-outputs** {***output_object_spec***}

  A required switch and string pair that specifies the output ports for a loopback connection.The ***output_object_spec*** string can specify one or more output ports. The output ports specified get paired with an input port specified in the -inputs ***input_object_spec*** switch. If the output port count does not match the input port count, the tool issues an error.

- -ac_delay_to_z *ac_delay_to_z*

  An optional switch and value pair that specifies the ac-delay-to-z value for all the loopback connections made in by this command. The value must be a valid time specification such as 10ns. This value triggers the creation of a coupling capacitor that tri-states (Z) the signal going back to the input after the amount of time on each level transition of the output signal.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

This example shows how this command is used to create a loopback pairs between two outputs and inputs. An ac_delay of 10ns is specified. The first output, dout[5], listed in the -outputs switch is connected to the first input, din[4], listed in the -inputs switch:

```
add_loadboard_loopback_pairs -inputs {din[4] ue} -outputs {dout[5] dout[0]} \
   -ac_delay_to_z 10ns
report_loadboard_loopback_pairs

-----------------------------
Inputs  Outputs  AC Time to Z
------  -------  ------------
din[4]  dout[5]  10ns
ue      dout[0]  10ns
-----------------------------

write_patterns pat.v -pattern_sets pat1 -verilog
```

The following statements are added to the Verilog file.

```
wire to_din_4_;
assign to_din_4_ = \din[4] ;
CouplingCapacitor #(10.000000) din_4__CC_INST(.in(\dout[5] ),
.out(to_din_4_));

wire to_ue;
assign to_ue = ue;
CouplingCapacitor #(10.000000) ue_CC_INST(.in(\dout[0] ), .out(to_ue));
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_misrs

Context: patterns -scan

Mode: setup, analysis

Specifies the MISR structure and initial seed.

Usage

add_misrs ***misr_name*** **-size *integer*** [-seed *hexadecimal*] [-taps *index*…]

Description

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

This command is included in the LogicBIST fault simulation dofile to describe the MISR logic synthesized.

Arguments

- ***misr_name***

  A required string that specifies the name of the MISR. The ***misr_name*** is subsequently used by the tool for referring to this MISR for specifying connections from scan chains, and in reporting.

- **-size *integer***

  A required switch and integer that specifies size of the MISR.

- -seed *hexadecimal*

  An optional switch and hexadecimal number pair that specifies the initial seed. Use this command if you intend on performing fault simulation with a different starting MISR seed.

- -taps *index*

  An optional switch and repeatable integers that specifies the tap points for the MISR feedback network. The tap index starts from 1 to be consistent with the decompressor used as a PRPG in the LogicBIST mode and associated decompressor commands.

Examples

This example shows the add_misrs command in context patterns -scan, mode setup. This example adds a 24-bit MISR named "misr" with initial seed of all 0-s:

```
add_misrs misr -size 24 -seed 000000 -taps 21 22 24
```

Related Topics

delete_misrs

report_misr_connections

report_misrs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_misr_connections

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_nofaults

Context: dft -edt, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup

Places nofault settings either on pin pathnames, pin names of specified instances, or modules.

Usage

add_nofaults {{***modulename*** ... **-Module** [-PIN *module_pin_pathname* ...] | ***object_expression*** ... [-PIN | -Instance]} [-Stuck_at {01 | 0 | 1}] [-Keep_boundary]} | -Wire [-VERBose] [(> | >>) *file_pathname*]

Description

By specifying pathnames of pins, instances, or modules while in Setup mode, the add_nofaults command places a nofault setting either on the specific pins or on boundary and internal pins of the instances or modules. All added nofault pin pathnames are in the user class.

Note

All switches except -Wire are valid only before flattening. The -Wire switch is valid only after flattening.

The -Stuck_at switch applies to either stuck-at faults or transition faults. For the latter, you specify 0 for “slow-to-rise” and 1 for “slow-to-fall” transition faults. If you do not specify a value, then the tool places a nofault setting on both stuck-at (transition) values. If you add faults with the add_faults command after you issue the add_nofaults command, the specified pin pathnames or boundary and internal pins of instances or modules cannot be sites for those added faults.

Note

When you add nofault settings, the tool deletes the flattened simulation model if it exists. This removes information you added after model flattening, such as ATPG functions. To avoid losing this information or having to wait for the tool to flatten the design again, add nofault settings prior to model flattening.

Arguments

- ***modulename***

  A repeatable string that specifies the name of a module to which you want to assign nofault settings. You must include the -Module switch when you specify a module name.

- **-Module**[-PIN *module_pin_pathname* ...]

  A switch that specifies to interpret the ***modulename*** argument as a module pathname. All instances of the module are affected. You can use the asterisk (*) and question mark (?) wildcards for the ***modulename*** argument, and the tool adds the nofault for all matching modules or library models.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When you follow the -Module switch with the optional -PIN switch, you can specify a list of module pin pathnames for the pin within the module. The tool automatically finds all the instances with the given module name and adds nofaults for every pin specified in module_pin_pathname list.

- ***object_expression***

  A string representing a list of pathnames of instances or pins for which you want to assign nofault settings. You can use regular expressions, which may include any number of asterisk (*) and question mark (?) wildcard characters.

  Pin pathnames must be Tessent Cell library cell instance pins, also referred to as design level pins. If the object expression specifies a pin within an instance of a Tessent Cell library model, the tool ignores it. By default, pin pathnames are matched first. If a pin pathname match is not found, the tool next tries to match instance pathnames. You can force the tool to match only pin pathnames or only instance pathnames by including the -Pin or -Instance switch after the *object_expression*.

- -PIN

  An optional switch that specifies to use the preceding object expression to match only pin pathnames; the tool then assigns nofault settings to all the pins matched.

- -Instance

  An optional switch that specifies to use the preceding object expression to match only instance pathnames; the tool then assigns nofault settings to all boundary and internal pins of the instances matched (unless you use the -Keep_boundary switch).

- -Stuck_at <u>01</u> | 0 | 1

  An optional switch and literal pair that specifies to which stuck-at or transition values you want to assign a nofault setting. The choices are as follows:

  <u>01</u> — A literal that specifies to place a nofault setting on both the “stuck-at-0” and “stuck-at-1” faults for stuck-at faults; or on both the “slow-to-rise” and “slow-to-fall” faults for transition faults. This is the default.

  0 — A literal that specifies to place a nofault setting on only the “stuck-at-0” faults for stuck-at faults; or on only the “slow-to-rise” faults for transition faults.

  1 — A literal that specifies to place a nofault setting on only the “stuck-at-1” faults for stuck-at faults; or on only the “slow-to-fall” faults for transition faults.

- -Keep_boundary

  An optional switch that specifies to apply nofault settings to the inside of the specified instance/module, but permit faults at the boundary pins of these instances/modules. This option does not apply to nofaults on pin pathnames.

- -Wire

  A switch that specifies for the tool to no-fault a cone of logic fanning out from a stem forward into a single wire gate. The tool no-faults the entire cone of logic between the stem

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

and the wire gate, but not the stem or the output of the wire gate. There must be no other fanout from this cone of logic. This switch is valid only after flattening.

Note

This switch affects all qualifying logic cones.

- -VERbose

  By default when you specify wildcard characters for instance (-INstance) or pin (-PIN) names, the tool outputs a summary message similar to the following:

  ```
  // Note:  Adding nofaults for 330 fault sites.
  ```

  When you specify the optional -VERbose switch, then the tool outputs the instance or pin names instead of the summary. You can optionally redirect this output to a file using the > or >> redirection operators.

  If you use actual instance or pin names instead of wildcards, then this switch has no effect.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

The following example defines nofault settings for all the pins in a particular instance, so when you add all faults to the circuit for an ATPG run, the tool does not place faults on the pins of that instance:

```
add_nofaults i_1006 -instance
set_system_mode analysis
create_patterns
```

Related Topics

add_faults

delete_faults

delete_nofaults

report_faults

report_nofaults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_nonscan_instances

Context: dft -scan, dft -test_points -no_rtl, dft -no_rtl, dft -rtl

Mode: setup, analysis

Sets a user-specified is_non_scannable attribute on the specified design objects.

Usage

Context: dft -scan

add_nonscan_instances {-instances *instance_spec* | -modules *module_spec* |
-scan_elements *scan_element_spec* | -control_signals *pin_spec* | -unmapped_flop_cells |
-all} [-force]

Context: dft -test_points -no_rtl

add_nonscan_instances {-instances *instance_spec* | -modules *module_spec* |
-scan_elements *scan_element_spec* | -control_signals *pin_spec* | -all} [-force]

Context: dft -no_rtl

add_nonscan_instances {-instances *instance_spec* | -modules *module_spec* |
-scan_elements *scan_element_spec* | -all} [-force]

Context: dft -rtl

add_nonscan_instances {-instances *instance_spec* | -modules *module_spec* |
-scan_elements *scan_element_spec* |
-rtl_reg *net_spec* | -all} [-force]

Description

Sets a user-specified is_non_scannable attribute on the specified design objects, thus excluding them from the scan insertion process.

The add_nonscan_instances command sets the user-specified is_non_scannable attribute on the specified instances, all sequential cells within the instances of the specified modules, the specified scan elements, the instances controlled by the specified control signals, or all instances.

Also, the tool does not perform the scannability rule checks on non-scan instances.

When adding non-scan instances with control signals, the tool adds all sequential instances controlled by the control signals to the non-scan instance list. Control pins can be any clock, set, or reset signal (primary input or pseudo-port) that controls the contents of sequential instances.

If TIE0 and TIE1 non-scan cells are scannable, they are considered for scan. However, if these cells are used to hold off sets and resets of other cells so that another cell can be scannable, you must use this command to make them non-scan.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The add_nonscan_instances command may invalidate data from other analysis stages. This command should be used before any X-bounding analysis, wrapper analysis, or scan chain analysis. If you have already completed any of these steps when you issue the add_nonscan_instances command, it results in an error. You can use the -force switch to override it, in which case the results of X-bounding analysis, wrapper analysis and scan chain analysis is reset and you must repeat them.

Arguments

- -instances *instance_spec*

  An optional switch and value pair that causes the tool to ignore the specified instances during scan insertion. The *instance_spec* can be a Tcl list of one or more instances or a collection of one or more instances. If the instance is hierarchical, then all instances beneath it are also flagged as non-scan.

- -modules *module_spec*

  An optional switch and value pair that causes the tool to ignore all sequential cells within the instances of the specified modules during scan insertion. The *module_spec* can be a Tcl list of one or more modules or a collection of one or more modules. Note: module may specify the name of a Verilog module, or a Tessent library model.

- -scan_elements *scan_element_spec*

  An optional switch and value pair that causes the tool to explicitly ignore the specified scan elements during scan insertion. The *scan_element_spec* can be a Tcl list of one or more scan elements or a collection of one or more scan elements.

  Note

  In analysis mode, the -scan_elements option is only available for leaf scan elements that are not part of a scan segment of a scan chain.

- -control_signals *pin_spec*

  An optional switch and value pair that causes the tool to ignore the instances controlled by the specified control signals. The *pin_spec* can be a Tcl list of pins and pseudo-ports or a collection of pins and pseudo-ports. You can use the -control_signals option in either setup or analysis mode.

  Note

  The -control_signals option is not available in the dft context with no sub-context.

- -rtl_reg *net_spec*

  An optional switch and value pair that specifies a Tcl list of one or more net names or a collection of one or more net objects. During quick synthesis, if the net ends up being synthesized to a flip-flop, the net ends up being directly connected to the Q pin of a DFF cell. The is_non_scannable attribute value is automatically transferred to the DFF cell instance. The is_non_scannable_reason attribute for these DFT cell instances returns user_specified_from_net. This switch is only available in the dft -rtl context (with no sub-

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

contexts). Use this switch when you want to declare an RTL register to be treated as non-scannable during DRC. You cannot specify the add_nonscan_instances command on the DFF cell instances directly because they do not exist before the quick synthesis has happened when going from setup to analysis mode.

- -all

  An optional switch that marks every scan cell in the design as a non-scan cell. When you issue the command with the -all switch, the tool completely clears the list of targeted cells, and you can then manually select an arbitrary subset of target cells using the add_scan_instances command.

- -unmapped_flop_cells

  An optional switch that marks every flop cell not already mapped to a scan cell in the design as a non-scan cell.

- -force

  An optional switch that directs the tool to forcefully issue the add_nonscan_instances command. In order to avoid using invalidated data, the tool resets the results of any X-bounding analysis and wrapper analysis and deletes all the scan modes and scan chain families. Re-run X-bounding and wrapper analysis and set the scan modes and scan chain families as needed after you run the add_nonscan_instances command.

Examples

Example 1

The following example specifies that the tool ignore the sequential i_1006 instance when identifying and inserting the required scan circuitry:

```
add_nonscan_instances -instances i_1006
```

Example 2

The following example clears all previously identified scan cells and then adds two instances to the list of scan cells to be stitched:

```
add_nonscan_instances -all
add_scan_instances -instances i_1006 i_1007
insert_test_logic
```

Related Topics

add_scan_instances

delete_nonscan_instances

insert_test_logic

report_seq_transparent_procedures

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_notest_points

Context: dft -scan, dft -test_points

Mode: setup, analysis

Prevents test logic insertion in the specified regions.

Usage

add_notest_points [-Control_only] [-OBserve_only]
{***pin_pathname***… | ***instance_pathname***… | ***instance_expression***} | **-Path *filename***

Description

The add_notest_points command is used to mark specific regions of the design to be excluded when inserting test logic (control points, observe points, fixing bidirectionals, set/reset, x-bounding and clock lines). This command is maintained to provide backward compatibility with previous versions of the tool, but the no_control_point and no_observe_point attributes actually provide a more flexible mechanism for accomplishing this task.

Internally, the add_notest_points command sets the no_control_point and no_observe_point attributes to True for the specified pin_pathname, instance_pathname, or instance_expression. When an instance is specified, the attribute is set only for that specific instance and not for any child instances because this is automatically covered by inheritance.

Note

The no_observe_point attribute prevents the addition of observe points at specific locations within the design. The tool does not consider the no_observe_point attribute while analyzing the clock network to identify the location to drive the clock port on an inserted memory element, such as the flop associated with an observe point.

If you specify a path file that contains delay paths, the tool marks each gate pin in the path by setting the no_control_point and no_observe_point attributes on each gate pin. This prevents control and observation points from being added to a critical path and thus prevents increasing the load on any of these gates in that path. The format of the file is the same as the file loaded with the read_fault_sites command. For more information on the format, refer to "The Path Definition File" in the *Tessent Scan and ATPG User's Manual*.

The report_notest_points command displays all the pins in this list or can list the defined critical paths.

Arguments

- -Control_only

  An optional switch that specifies that no control points are added in the specified regions.

- -OBserve_only

  An optional switch that specifies that no observe points are added in the specified regions.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

By default, this command prevents the insertion of both control and observe points.

- ***pin_pathname***

  A required, repeatable string that lists the pins that you do not want to use for insertion of controllability and observability.

- ***instance_pathname***

  A repeatable string that lists the instances whose pins you do not want to use for controllability and observability insertion. All pins within that (hierarchical) instance are added to the list of pins that should be excluded from consideration.

- ***instance_expression***

  A string representing a list of instances within the design. The string *instance_expression* is defined as:

  ```
  { string | string * } ...
  ```

  The asterisk (*) is a wildcard that matches many instances in a design. Any expression that does not contain an asterisk (*) matches exactly zero or one instance.

  This argument does not support pathnames to objects below the instance level of a Tessent Cell library model. You can use a pathname expression to select several instances and the tool then runs add_notest_points for all the pins on those instances; but if the expression specifies a location below the instance level of a Tessent Cell library model, the tool reports an error message.

- **-Path *filename* (dft -test_points context only)**

  A switch and filename pair that specifies the pathname to a file that contains critical path information. For more information on the format of the file, refer to "The Path Definition File" in the *Tessent Scan and ATPG User's Manual*.

Examples

The following example specifies output pins that the tool cannot use for testability insertion:

add_notest_points i_1006/o i_1008/o i_1009/o

Related Topics

delete_notest_points

set_test_point_insertion_options

report_notest_points

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_observe_points

Context: dft -test_points

Mode: setup, analysis

Specifies user-defined observe points during test point insertion.

Usage

add_observe_points **-location *pin/port_spec***
[-clock *pin/port_spec*]
[-enable *pin/port_spec*]

Description

Use this command to specify your own observe points during test point insertion.

Note

After running the add_observe_points command, you must run the analyze_test_points command before running the analyze_scan_chains, analyze_wrapper_cells, or analyze_xbounding commands.

Arguments

- **-location *pin/port_spec***

  A required string that specifies the pathname to the observe point location where ***pin/port_spec*** can be a Tcl list of one or more pins/ports or a collection of one or more pin/port objects.

- -clock *pin/port_spec*

  An optional switch and string pair that specifies the path name of the clock signal used to clock the observe point. This can be a collection or list, but should have only one element.

- -enable *pin/port_spec*

  An optional switch and string pair that specifies the path name of an existing net used to enable the observe point. This can be a collection or list, but should have only one element.

  If you want to specify the default enable signal for the automatically identified observe points instead of manually adding each enable signal, use the set_test_point_insertion_options command as follows:

  ```
  set_test_point_insertion_options -observe_points_enable
  ```

Examples

Example 1

The following example adds two observe points, and specifies the new scan cell model used to observe the inserted test points:

add_observe_points -location ss1/ix348/Z

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
add_observe_points -location ss1/ix302/Z
```

Example 2

The following example specifies the path names of existing nets that are used to enable the control points and observe points respectively. Then it adds a control point and an observe point, and proceeds to insert the test points into the netlist:

```
set_test_point_insertion_options -control_point_enable cp_enable \
        -observe_point_enable op_enable

add_control_points -location jtc/i1216/OUT -type and

add_observe_points -location jtc/i1645/OUT

...

set_system_mode analysis

...

insert_test_logic
```

Related Topics

add_control_points

delete_test_points

report_test_points

set_test_point_insertion_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_output_masks

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Ignores any fault effects that propagate to the primary output pins you name.

Usage

add_output_masks ***primary_output***… | **-All**

Description

patterns -scan Details

The tool uses primary output pins as the observe points during the fault detection process. When you mask a primary output pin, you inform the tool to mark that pin as an invalid observation point during the fault detection process. This command enables you to flag primary output pins that do not have strobe capability. The tool classifies the faults whose effects only propagate to that observation point as ATPG_untestable (AU).

dft -scan Details

The tool uses primary output pins as the observe points during the scan identification process. When you mask a primary output pin, you inform the tool to mark that pin as an invalid observation point during the identification process.

There are specific design practices that mask specific primary output pins in order to group the sequential elements associated with those primary outputs (for example, if you are using wrapper chains). For more information on wrapper chains, refer to "About Wrapper Chains" in the *Tessent Scan and ATPG User's Manual*.

You can set a default mask for all output and bidirectional pins using the set_output_masks command. You can add a hold value to a default mask with the add_output_masks command, or remove a hold value using the delete_output_masks command. To turn off the default masks for all output pins, you must use the set_output_masks command with the Off literal.

When you use the add_output_masks command in the dft -scan context, the wrapper analysis for that pin is excluded.

Note

The pin constraints added when you use this command can be deleted using the delete_output_masks command. Pin constraints added in setup mode cannot be deleted in analysis mode and vice versa.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***primary_output***

  A repeatable string that specifies the name of the primary output pin you want to mask.

- **-All**

  A switch that specifies to mask all primary output pins.

Examples

The following example specifies the primary output pins that do not have the strobe capability on the hardware tester:

```
add_output_masks qb1 qb2 qb3
```

Related Topics

analyze_restrictions

delete_output_masks

report_output_masks

set_output_masks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_pade_logical_connection

Context: dft

Mode: setup, analysis, insertion

Creates a new logical connection path through the current design by specifying the source and destination of the logical path.

Usage

add_pade_logical_connection **-from *source_pin_or_port_name***
[-from_pin_direction {input | output}]
-to *destination_pin_or_port_name*
[-to_pin_direction {input | output}]
[-inverted]

Description

During pad extraction tracing, logical connections are followed in preference to the actual connections. When tracing reaches an instance pin with a logical connection, it follows the logical connection across the design hierarchy. A forward trace follows a logical connection whose source is on the last pin traced and jumps to the destination of that logical connection. A backward trace follows a logical connection whose destination is on the last pin traced and jumps to the source of that logical connection.

See Figure 3-26 for an example of a logical connection.

Use this command with care as it basically bypasses the structural tracing of the circuit and uses an act of faith on the behavior of the specified logical source. If the behavior turns out to be inconsistent with the specified logical sources, you only detect the issue in simulation. Use this command to model a path through untraceable RTL or through a black-box module.

This command is useful for the following types of tracing:

- Extraction of the pads of a top-level port
- Selection of keyports for boundary scan cell insertion
- Extraction of constant drivers on pad pins

Arguments

- **-from *source_pin_or_port_name***

  A required switch and value pair that specifies the source point of the logical connection, which should be a top-level module port or a hierarchical instance pin in the current top design module.

- -from_pin_direction {input | output}

  An optional switch-value pair that enables you to specify the pin direction of the source of the ijtag_logical_connection, if the pin specified by the “-from” switch is a bidirectional pin.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-to *destination_pin_or_port_name***

  A required switch and value pair that specifies the destination point of the logical connection, which should be a top-level module port or a hierarchical instance pin in the current top design module.

- -to_pin_direction {input | output }

  An optional switch-value pair that enables you to specify the pin direction of the destination of the ijtag_logical_connection, if the pin specified by the "-to" switch is a bidirectional pin.

- -inverted

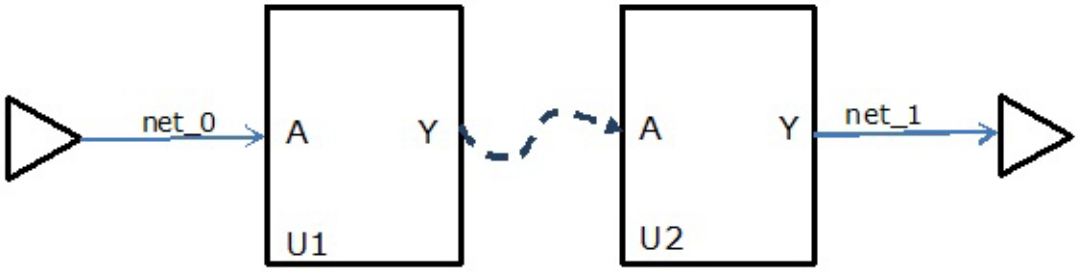
  An optional switch that specifies that the logical connection is inverted.

Examples

Example 1

The following example illustrates a logical connection from pin U1/Y to pin U2/A. A forward trace that arrives at pin U1/Y continues on from pin U2/A. A backward trace that arrives at pin U2/A continues on from pin U1/Y.

Figure 3-26. Logical Connection Example



At the netlist level all interconnections start at an instance pin or a top level module port and end at an instance pin or top level module port.

Example 2

The add_pade_logical_connections command creates logical connections between the pins /i2/in and /i2/out, /i3/in and /i3/out and i4/in and /i4/out, and an inverted logical connection between /i5/in and /i5/out. The report_pade_logical_connections command reports all logical connections in the current design.

```
add_pade_logical_connection -from /i2/in -to /i2/out
add_pade_logical_connection -from /i3/in -to /i3/out
add_pade_logical_connection -from /i4/in -to /i4/out
add_pade_logical_connection -from /i5/in -to /i5/out -inverted
report_pade_logical_connections
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  PADE Logical Connections:
//  From Source  To Destination   Inversion
//  ===========  ==============   =========
//  /i2/in       /i2/out          no
//  /i3/in       /i3/out          no
//  /i4/in       /i4/out          no
//  /i5/in       /i5/out          yes
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_pattern_trigger

Context: patterns -silicon_insight

Mode: all modes

Adds a trigger for additional equipment for measurement at a specific pattern location. This command applies only to OpalKelly adaptors using smart mode.

Usage

add_pattern_trigger ***pattern_name*** **-vector_address** ***vector_address*** [-type *type*]

Arguments

- ***pattern_name***

  A required string that specifies the name of the pattern in which to add the trigger.

- **-vector_address** ***vector_address***

  A required switch and nonnegative integer that specifies the vector address for the trigger. For XEM6310, the pin position is AB9. For XEM8350, the pin position is AJ39.

- -type *type*

  An optional switch and string that specifies the type of the trigger. This describes the waveform on the trigger pin. The legal value for the type is “pulse”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_primary_inputs

Context: all contexts

Mode: setup

Adds a primary input (PI) also known as an input pseudo_port to the specified pins or nets.

Usage

add_primary_inputs ***pin_or_net_pathname***... [-pseudo_port_name *user_name*] [-silent]

Description

Adds an input pseudo_port to the specified nets or pins. This command also designates the pseudo_port as user-class primary inputs (PI) rather than system-class PIs described in the original netlist. By default, the tool appends the names of these created pseudo ports with suffix "_pport". This is done to distinguish the pseudo port from the internal pin path name and avoid creating two gates with the same name.

If -pseudo_port_name *user_name* is specified, then the pseudo port name is the *user_name* as is. No "_pport" suffix is added.

When you issue the command, the tool disconnects the original drivers of the net such that the added PI becomes the only driver of the pin or net. The tool omits these PIs from the top-level interface when saving patterns.

Note

The tool does not fault PIs created with this command. This enables interchange of fault lists between runs of the tool across different modes where you may have used different cut points.

When you use the write_patterns -Mode_internal switch when saving serial or parallel simulation patterns, the testbench still forces the internal nodes on *pin_or_net_pathname* even though the corresponding PIs are not included in the top-level interface. When you save the patterns using the -Mode_external switch, the testbench does not force those internal nodes. In this case, unless the circuit generates values on the internal nodes corresponding to what ATPG assumed was forced on the corresponding PIs, you may get simulation mismatches.

Use the -pseudo_port_name switch to specify a list of internal pin pathnames to connect to a single, new primary input pin. You can display the user-class, system-class, or full-class PIs by using the report_primary_inputs command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***pin_or_net_pathname***

  A required, repeatable string that specifies the pathname of a pin or net to which you want to add a primary input. The *pin_or_net_pathname* may include any number of asterisk (*) and question mark (?) wildcard characters in a name.

- -pseudo_port_name *user_name*

  An optional switch and string pair that specifies the name for a new primary input pin name that drives all the internal pins specified with the *pin_or_net_pathname* argument. The *user_name* must be a simple name that begins with an alpha character and can contain only alphanumeric characters and underscore characters ('_'). Wildcards are not permitted.

- -silent

  An optional switch that suppresses the transcription of notes the tool issues.

Examples

Example 1

The following example adds two new primary inputs to the circuit and places them in the user class of primary inputs:

add_primary_inputs indata2 indata4

Example 2

The following examples demonstrate various uses of the add_primary_inputs command:

SETUP> add_primary_inputs final_carry/A0 // cell input

```
//  Warning: Primary input 'final_carry/A0_pport' is added at pin
//  '/final_carry/A0'
```

SETUP> add_primary_inputs final_carry/Y // cell output

```
//  Warning: Primary input 'final_carry/Y_pport' is added at pin
//  '/final_carry/Y'
```

SETUP> add_primary_inputs ha1/a // module input

```
//  Warning: Primary input 'ha1/a_pport' is added at pin '/ha1/a'
```

SETUP> add_primary_inputs ha1/s // module output

```
//  Warning: Primary input 'ha1/s_pport' is added at pin '/ha1/s'
```

SETUP> report_primary_inputs -class user

```
USER:    '/ha1/s_pport' (internal pin)
USER:    '/ha1/a_pport' (internal pin)
USER:    '/final_carry/Y_pport' (internal pin)
USER:    '/final_carry/A0_pport' (internal pin)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

The following example adds a PI to the net originally driven by the output of a NAND gate, disconnecting the output of the NAND from the net. This happens within the tool only; the tool does not modify the original source netlist.

set_gate_level primitive
create_flat_model
report_gates /ix157/Y

```
//  /ix157 (94)  NAND
//      A0     I  25-/reg_0_/Q
//      A1     I  33-/reg_1_/Q
//      Y      O  142-/d_2_mult_0_/A1
```

add_primary_inputs /ix157/Y

report_gates /ix157/Y

```
//  /ix157 (95)  NAND
//      A0     I  26-/reg_0_/Q
//      A1     I  34-/reg_1_/Q
//      Y      O
```

Example 4

The following example connects internal nets “xyz/abc/pina xyz/def/pina xyz/ghi” to a single, new primary input and disconnects the original drivers of the nets, making the added primary input the only driver of the nets. This is the default behavior. The name of the new primary input pin is “group_pi”.

add_primary_inputs xyz/abc/pina xyz/def/pina xyz/ghi -pseudo_port_name group_pi

Related Topics

add_clocks

add_primary_outputs

delete_primary_inputs

report_primary_inputs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_primary_outputs

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup

Adds primary outputs (PO) to the specified net.

Usage

add_primary_outputs ***pin_or_net_pathname***... [-pseudo_port_name *user_name*] [-silent]

Description

The add_primary_outputs command adds an additional primary output to each specified pin or net. Once added, the tool defines them as user-class primary outputs. These primary outputs are not added to the top-level interface when saving mode_external patterns, such as STIL and WGL.

Note

The tool does not fault POs created with the add_primary_outputs command. This enables interchange of fault lists between runs of the tool across different modes where you may have used different cut points.

The tool defines the primary outputs described in the original netlist as system class primary outputs. You can display the user class, system class, or full classes of primary outputs using the report_primary_outputs command.

Arguments

- ***pin_or_net_pathname***

  A required, repeatable string that specifies the pin or net to which you want to add primary outputs. The *pin_or_net_pathname* may include any number of asterisk (*) and question mark (?) wildcard characters in a name.

- -pseudo_port_name *user_name*

  An optional switch and string pair that specifies the name for a new primary output pin name that is driven by the internal pin specified with the *pin_or_net_pathname* argument. The user_name must be a simple name that begins with an alphabetic character and can contain only alphanumeric and underscore characters ('_'). Wildcards are not valid.

- -silent

  An optional switch that suppresses the transcription of notes the tool issues.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example adds a new primary output to the circuit and places it in the user class of primary outputs:

```
add_primary_outputs outdata1
```

Related Topics

add_primary_inputs

delete_primary_outputs

report_primary_outputs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_processors

Context: unspecified, dft -edt, patterns -scan, patterns -scan_retargeting

Mode: setup, analysis, insertion (dft -edt context only)

Enables the tool to run multiple processors in parallel on multiple machines to reduce runtime.

Usage

add_processors {***hostname***[:{*cpus* | MAXcpu}]}…

Description

The add_processors command enables the tool to use multiple processors simultaneously on specified host machines when running multiprocessing commands for ATPG or fault simulation. Remote processes can be created on the primary host and on other hosts. Each process, whether on the primary or secondary host, can optionally use multiple processors through multithreading.

For more information, refer to "Multiprocessing for ATPG and Simulation" in the *Tessent Scan and ATPG User's Manual.*

Running multiple processes in parallel can sometimes reduce runtime significantly, especially for large designs. Be sure to load your design in the tool before using the add_processors command. By doing this, the tool more accurately estimates memory requirements for the added processes.

Typically, the tool runs as a single primary process. When you make additional processors available with the add_processors command, there exists a single primary process. The tool then starts additional secondary processes and adds additional processors (threads) to existing processes. This enables parallel running and coordination of communication between all processors.

Multiprocessing commands that use remote processes alert you with a message when the design is being sent to remote processors. This message may not appear during subsequent commands if the remotes have already received the design and you have issued no commands in the interim that would affect the information received. There is currently a limit of 1024 processors per add_processors command.

Tip

To stop the command from waiting any longer for requested remote processes not yet received, and proceed with those it has, use the Control-C key sequence.

Arguments

- ***hostname***

  A required, repeatable string or literal that specifies a host machine or job scheduler that the tool can use for remote processes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The hostname choices are as follows:

machine_name — A string that specifies a host machine by its network name.

LOCALHOST — A literal that specifies to use the machine you invoked the tool on as the host machine.

IPaddress — A string that specifies a host machine by its IP address.

LSF — A literal that specifies to use available Load Sharing Function (LSF) host machines.

SGE — A literal that specifies to use available Sun Grid Engine (SGE) host machines.

Note

This documentation refers to SGE, which is a product of Altair.

GENERIC — A literal that specifies to use a host machine as determined by a custom job scheduler. This option requires that you first issue a set_multiprocessing_options command with the generic_scheduler argument to inform the tool how to launch the custom job scheduler.

Note

When you add SGE or LSF remote hosts with the add_processors command, the tool estimates the available host memory and swap space needed for the remotes based upon the current memory usage of the tool and the number of requested processors. The tool includes these requirements in its requests for processors from the grid system. For this reason, it is best that you specify any setup commands that may have significant effects on memory usage (such as leaving setup mode and reading SDC and SDF files) prior to using the add_processors command. In the Tessent Shell tool, the design should be loaded before adding remote processors.

Note

The use of job schedulers requires certain environment prerequisites to be satisfied before you invoke the tool. See "Multiprocessing Requirements" in the *Tessent Scan and ATPG User's Manual* for more information.

- :{*cpus* | MAXcpu}

  A colon (:) followed by a positive integer or the MAXcpu keyword enables you to optionally specify the number of processors to use on the specified host machine or through a job scheduler. You can specify up to 32 processors per host, and the tool gives a warning if you specify 0 processors.

  If you specify a positive integer, the tool adds that number of processors.

  If using a job scheduler, the tool uses as many processors as the scheduler provides in the time allotted. The time allotted is determined by the setting of the scheduler_timeout variable, which you can change using the set_multiprocessing_options command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If you specify the MAXcpu keyword, the tool automatically determines the number of physical processors on the specified host and adds enough additional processors to bring the total number of running processors on that host up to the number of physical processors. The default for the LSF scheduler is eight processors while the default for all others is one processor.

Note

The MAXcpu keyword does not apply to grid scheduling or generic scheduling. The tool's cpu calculations are based on currently running processors without considering other remote processor specifications on the same command line. For example, if you use the MAXcpu keyword twice in the same command for a given host, the number of remote processors on that machine could exceed the number of available processors.

Examples

Example 1

The following example adds processors to the primary host, which increases the thread count instead of adding additional remotes (unless multithreading has been disabled with the set_multiprocessing_options command):

add_processors localhost:4
 # adds 4 threads to the primary process for a total thread count of 5

```
// Adding 4 threads to birdeye (primary process)
// Primary process with 5 threads running.
```

report_processors

```
//  hosts              threads   arch      CPU(s)     %idle   free RAM  process size
//  ----------------   -------  ------  -----------  -----  ---------  -----------
//  birdeye (primary)        5  x86-64  8 x 2.9 GHz    89%  113.91 MB  232.56 MB
//  Primary process with 5 threads running.
```

Example 2

The following example starts a distributed remote process on the remote host "odin" that runs with four total threads. The add_processors command normally starts only one process per remote host, and adds additional processors (to primary or secondary host) as additional threads. (Exception: If multithreading is disabled with the set_multiprocessing_options command, add_processors creates multiple single-threaded remotes.)

add_processors odin:4
 # adds 4 total threads to the secondary process on the host named odin
 # for a total thread count of 5 that includes the primary process thread

```
// Adding 4 threads to odin (new secondary process)
// Primary process with 1 thread and 1 secondary process with 4 threads running
//  (5 total threads).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

compress_patterns

create_patterns

delete_processors

order_patterns

report_processors

set_multiprocessing_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_read_controls

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup

Specifies the off-state value for specified RAM read control lines.

Usage

add_read_controls {**0** | **1**} ***pin_pathname***...

Description

The add_read_controls command defines the circuit read control lines and assigns their off-state values. The off-state value of the pins or ports that you specify must be sufficient to keep the RAM outputs stable. The following are considerations for using this command:

- When the read control is already added as a clock, there is no need to also use add_read_controls for that port. This is the typical case, which means that for most usage it is unnecessary to explicitly use the add_read_controls command.
- If you want to have NR (non-clock) timing for read control lines that drive level-sensitive RAMs, you must add the read control using the add_read_control command; otherwise, the read control cannot pass DRC. NR timing for read control is not common.
- If a port drives the clock port of an edge-triggered RAM, the tool automatically infers that the port is a clock and adds it to the clock list. This is regardless of whether you added the port as a read control or not. For clocks that the tool infers, you must ensure that your timeplate includes the waveform for the newly defined clock.

Arguments

- **0 | 1**

  A required literal that specifies 0 or 1 as the off-state value for the RAM read control lines.

- ***pin_pathname***

  A required, repeatable string that specifies the pins or ports you want to designate as RAM read control lines, and to which you are assigning the given off-state value.

Examples

The following example assigns an off-state value of 0 to two read control lines, r1 and r2:

```
add_read_controls 0 r1 r2
set_system_mode analysis
create_patterns
```

Related Topics

analyze_control_signals

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_read_controls

report_read_controls

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_register_value

Context: all contexts

Mode: setup

Creates a static or dynamic variable through which you specify a register identifier, *value_name*, and, optionally, the state string, *value_string*.

Usage

add_register_value ***value_name*** {[*value_string* [-Dynamic]
[-Radix {Binary | Decimal | Octal | Hexadecimal}] {-Width *integer*} |
{-Pattern_count {SCan_test | CHain_test | ALl}}}
[-LSB_shifted_first] [-INput_pin_inversion] [-OUtput_pin_inversion]

Description

When you specify a *value_string*, the variable is static by default unless you use the -Dynamic switch.

See "Serial Register Load and Unload for LogicBIST and ATPG" in the *Tessent Shell User's Manual* for complete information.

Arguments

- ***value_name***

  A required string that specifies register value variable. The variable holds a value to load into the register. The register value variable is used in the procedure file to load the value into a register through the data input pin.

- *value_string*

  An optional state string whose default radix is binary. You can specify Z or X values in addition to numerical values appropriate for the radix. When you specify a *value_string*, the variable is static by default unless you use the -Dynamic switch. You cannot specify a *value_string* when you use the -Pattern_count switch.

- -Dynamic

  An optional switch that forces the variable to be dynamic when you provide an optional *value_string*. By default, when you provide a *value_string*, the variable is static.

- -Radix {Binary | Decimal | Octal | Hexadecimal}

  An optional switch and literal that specifies the radix of the *value_string*. If you specify the -Radix switch, then you must specify the register width using the -Width switch.

- -Width *integer*

  A switch and integer pair that specifies the register width. When defining a dynamic variable, you must use this switch.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The maximum width of a register value is 1,000,000,000. A specified width larger than this produces an error.

- -Pattern_count {SCan_test | CHain_test | ALl}

  A switch and literal pair that specifies using the total number of patterns generated by the tool when the variable is dynamic. Note that when you use this switch, you cannot specify a *value_string*. Choose from one of the following:

  SCan_test — Scan patterns.

  CHain_test — Chain patterns.

  ALl — Both scan and chain patterns.

- -LSB_shifted_first

  An optional switch that specifies shifting the less significant bit first into the register.

- -INput_pin_inversion

  An optional switch that specifies an inversion on the input pin when loading the register.

- -OUtput_pin_inversion

  An optional switch that specifies an inversion on the output pin when unloading the register.

Examples

The following example deletes all register value variables and then specifies two dynamic register value variables with a decimal radix and register width of 37.

delete_register_value -all

add_register_value setup_TDI1 0 -dynamic -radix decimal -width 37

add_register_value setup_TDI2 10 -dynamic -radix decimal -width 37

report_register_value

```
//  Register value name                value string                width
//  -------------------  ---------------------------------------  -----
//       setup_TDI1      00000000000000000000000000000000000000    37
//       setup_TDI2      00000000000000000000000000000000001010    37
```

Related Topics

delete_register_value

report_register_value

set_register_value

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_rtl_to_gates_mapping

Context: all contexts

Mode: all modes

Specifies custom rules for RTL to post-synthesis and post-layout name mapping for registers, instances, and ports.

Usage

add_rtl_to_gates_mapping {[-register_name_prefix *register_name_prefix*]
[-register_name_suffix *register_name_suffix*]
[-local_register_ancestry_name_map_list *local_register_ancestry_name_mappings*]} |
[-instance_name_map_list *string_pair_list*] |
[-port_name_map_list *port_name_mappings*] |
[-port_name_substring_map_list *port_name_substring_mappings*]

Description

During the flow from RTL to the final gate-level netlist, the tool stores a number of files, for example, Tessent Core Descriptions (TCDs), ICL files, and so forth. These files contain design instance path names, RTL register instance path names, and complex port names. During synthesis and layout, the RTL register names are mapped to flip-flop instances, the design hierarchy may be altered through grouping or ungrouping of various hierarchy levels, and complex port names are mapped to one-dimensional ports. To handle the renaming, the tool performs name mapping to find the objects in the netlist using the names they had in the RTL. By default, the tool is configured to handle most common name-mapping cases. If the default name-mapping rules are not sufficient, you must provide additional name-mapping rules using the add_rtl_to_gates_mapping command. You typically do not need to adjust the mapping rules unless you used “ungroup -simple_names” during synthesis as shown in the example below.

You use the following commands and switches to return a collection of pins, ports, or instances that use the RTL post-synthesis/post-layout name mapping:

- get_instances -match_rtl_reg
- get_instances -use_path_matching_options
- get_pins -use_path_matching_options
- get_ports -use_path_matching_options
- get_ports -of_icl_ports
- get_pins -use_path_matching_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- -register_name_prefix *register_name_prefix*

  An optional switch and string pair that specifies a prefix to be applied before the first component of an RTL register name when performing introspection with the "get_instances -match_rtl_reg" command. The tool applies the same default matching rules described in the get_pins and get_ports commands.

- -register_name_suffix *register_name_suffix*

  An optional switch and string pair that specifies a suffix that is applied after the first component of an RTL register name when performing introspection with the "get_instances -match_rtl_reg" command. The tool applies the same default matching rules described in the get_pins and get_ports commands.

The two preceding options are applied using the following name mapping rule:

```
<prefix>%name<suffix>{[%d]}{.%name{[%d]}}
```

The elements of this syntax are:

```
<prefix>        - The value of the -register_name_prefix option.

<suffix>        - The value of the -register_name_suffix option.

<%name>         - A placeholder for a component name of a register name.
```

{.%name{[%d]}} - The component names and associated indexes that follow the root name. The braces ({}) indicate that this portion of the rule maps to zero or more components of a register name.

The tool applies this name mapping rule to the name patterns of all invocations of the get_instances command that use the -match_rtl_reg option. After the tool transforms the name pattern with the name mapping rule, it matches it against the names in the netlist as described in "Default Matching Rules for the get_pins, get_ports, and get_instances -match_rtl_reg Commands" in the *Tessent Shell User's Manual*.

The default register name mapping rules are:

```
%name_reg{[%d]}{.%name{[%d]}}
rtlcreg_%name{[%d]}{.%name{[%d]}}
```

You can apply both the prefix and suffix options in the same mapping rule. For example, given the following commands:

```
(1) add_rtl_to_gates_mapping -register_name_prefix "AAA_"
(2) add_rtl_to_gates_mapping -register_name_suffix "_BBB"
(3) add_rtl_to_gates_mapping -register_name_prefix "AAA_" \
         -register_name_suffix "_BBB"
```

Rule 1 matches "AAA_<*registername*>", and rule 2 matches "<*registername*>_BBB". Rule 3 is needed to match "AAA_<*registername*>_BBB".

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

You cannot add prefixes and suffixes as regular expressions. The tool does not support wildcards in prefixes and suffixes.

- -local_register_ancestry_name_map_list *local_register_ancestry_name_mappings*

  An optional switch and string pair list that specifies path mappings between the last level of module hierarchy and the instantiation of the register.

  - The first path in a pair is the existing RTL path.
  - The second path in a pair is the path to which you want to map the first path.

  Apply the option at the same time you specify the -register_name_prefix and -register_name_suffix options. If you do not specify those options, the value of those options is assumed to be an empty string.

- -instance_name_map_list *string_pair_list*

  An optional switch and string pair list that specifies instance name-mapping rules. By default, you do not need to specify mapping because the tool already maps constructs such as “corea_i1/coreb_i1” to “\corea_i1/coreb_i1” and “corea_i1_coreb_i1”. Even a hierarchy path that is part of a generated block, such as “b1.b2[3].u2”, is automatically mapped to “\b1.b2[3].u2” and “b1_b2_3__u2”. If you used “ungroup -simple_name” to ungroup coreb_i1 into corea_i1, you must specify a mapping from “corea_i1/coreb_i2” to “corea_i2” as shown in the following example.

  Note

  You cannot use this option to specify name mapping for SystemVerilog interface instances.

- -port_name_map_list *port_name_mappings*

  An optional switch and string pair list that specifies port name-mapping rules. The first name in a pair is the RTL or gate name that is to be mapped. When the introspection command is “get_ports -of_icl_ports” or “get_pins -of_icl_pins,” the first name must exactly match either the tessent_design_gate_ports or the tessent_design_rtl_ports attribute of the ICL object. When the introspection command is “get_ports -use_path_matching_options” or “get_pins -use_path_matching_options”, the first name must be an exact match with the port name argument or, in the case of get_pins, the pin name argument’s leaf name.

  The second name in a pair is the new gate name to be looked up in the netlist. It must be an exact match with a port name in the netlist; that is, the default mapping rules described in “Default Matching Rules for the get_pins, get_ports, and get_instances -match_rtl_reg Commands” in the *Tessent Shell User’s Manual* are not applied when you use this option.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Both names are complete bit-select port names, not components of a port name. Provide the mapping pair list as a flat list as follows:

```
{<name_to_be_mapped_1> <new_netlist_name_1>
 <name_to_be_mapped_2> <new_netlist_name_2> ...
 <name_to_be_mapped_n> <new_netlist_name_n>}
```

The number of names in the list must be an even number.

- -port_name_substring_map_list *port_name_substring_mappings*

  An optional switch and string pair list that specifies substring mapping to be applied to the name being introspected. That name may be the value of the tessent_design_gate_ports or the tessent_design_rtl_ports attribute of the ICL object specified in a "get_ports -of_icl_ports" or "get_pins -of_icl_pins" command. Otherwise, it is the port name argument of a "get_ports -use_path_matching_options" command or the leaf name of the pin name argument of a "get_pins -use_path_matching_options" command.

  The option takes a list of substring pairs. If the first substring in a pair appears in the name being introspected, then all substring occurrences are replaced with the second substring in the pair. All substring pairs are applied to the name. At that point, the new name lookup occurs in the netlist. If the name is not found, the default mapping rules are applied as described in "Default Matching Rules for the get_pins, get_ports, and get_instances -match_rtl_reg Commands" in the *Tessent Shell User's Manual*.

  Provide the substring mapping pair list as a flat list as follows:

```
{<substring_to_be_replaced_1> <substring_replacement_1>
 <substring_to_be_replaced_2> <substring_replacement_2> ...
 <substring_to_be_replaced_n> <substring_replacement_n>}
```

Examples

Example 1

This example specifies whether objects within instance coreb_i1 were ungrouped into their parent instance corea_i1 using the "ungroup -simple_name" command by using add_rtl_to_gates_mapping command. The report_rtl_to_gates_mapping command invocation shows the added mapping. As described above, you typically never need to add mapping unless you have used the "ungroup -simple_name" command with your synthesis tool.

add_rtl_to_gates_mapping -instance_name_map_list {corea_i1/coreb_i1 corea_i1}

report_rtl_to_gates_mapping

```
// Index  Register Name Prefix  Register Name Suffix
   -----  --------------------  --------------------
    1                                          _reg
    2                rtlcreg_
    3
// Index Original instance name Mapped instance name
// ----- ---------------------- --------------------
// 1     corea_i1/coreb_i1      corea_i1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

This example maps the RTL design name of a port bit to the corresponding netlist name by using the add_rtl_to_gates_mapping command with the -port_name_map_list option. The default mapping rules are sufficient to perform needed mappings in most cases. This command enables you to create a mapping when the default rules are unable to find a match. Consider the following design:

```
package defs;
    typedef struct packed {
    logic [1:0] joe;
    logic sam;
} my_type1;
endpackage

module core (input defs::my_type1 clka);
...
endmodule
```

The clka port has three bits:

```
clka.joe[1]
clka.joe[0]
clka.sam
```

After renaming clka to clkz in Genus, the following appears in the netlist:

```
module core (\clkz[sam] , \clkz[joe] );
input \clkz[sam] ;
input [1:0] \clkz[joe] ;
...
endmodule
```

If the three bits of clka are ICL ports, you must provide mappings from the RTL name clka to the gate name clkz in the gate-level flow. These mappings must be at the bit level:

```
add_rtl_to_gates_mapping -port_name_map_list \
{{clka.joe[1]} {\clkz[joe] [1]} {clka.joe[0]} {\clkz[joe] [0]} \
 {clka.sam} {\clkz[sam] }}
```

You must run this command before elaborating the design.

When you elaborate the design with “set_current_design core”, the tool uses these mappings to update the tessent_design_gate_ports attributes of the ICL ports clka[2], clka[1], and clka[0]. The three updated attribute values are as follows:

```
\clkz[joe] [1]
\clkz[joe] [0]
\clkz[sam]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

To introspect the design port corresponding to the ICL port clk1[1], use the following commands:

set port1 [get_icl_ports clka[1] -of_modules core]
get_ports -of_icl_ports $port1

The tool uses the tessent_design_gate_ports attribute of the ICL port (which is updated during set_current_design) to look up the corresponding design port in the netlist.

You can also perform the introspection with the following command:

```
get_ports clka.joe[0] -use_path_matching_options
```

The tool uses the mappings provided by the add_rtl_to_gates_mapping command to map the given name to the corresponding netlist name and look up the design port in the netlist.

Example 3

This example maps a substring in your RTL design name to a replacement substring by using the add_rtl_to_gates_mapping command with the -port_name_substring_map_list option. You may need to do this if you made such a replacement in your synthesis tool. Consider the following design:

```
module core( input joezz, input johnzz);
...
endmodule
```

After you replace all occurrences of “zz” with “yy”, the following appears in the netlist:

```
module core(joeyy, johnyy);
  input joeyy;
  input johnyy;
  ...
endmodule
```

If joezz and johnzz are ICL ports, you must provide a substring mapping that maps all occurrences of the substring “zz” to “yy” in the gate-level flow:

add_rtl_to_gates_mapping -port_name_substring_map_list {zz yy}

You must run this command before elaborating the design.

During elaboration of the design with the “set_current_design core” command, the tool updates the tessent_design_gate_ports attributes of the ICL ports joezz and johnzz with that mapping.

```
joeyy
johnyy
```

To introspect the design port corresponding to the ICL port joezz, use the following commands:

```
set port1 [get_icl_ports joezz -of_modules core]
get_ports -of_icl_ports $port1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The tool uses the tessent_design_gate_ports attribute of the ICL port (which is updated during set_current_design) to look up the corresponding design port in the netlist.

You can also perform the introspection with the following command:

```
get_ports joezz -use_path_matching_options
```

The tool uses the mappings provided by the add_rtl_to_gates_mapping command to map the given name to the corresponding netlist name and look up the design port in the netlist.

Related Topics

delete_rtl_to_gates_mapping

get_instances

get_pins

get_ports

report_rtl_to_gates_mapping

Default Matching Rules for the get_pins, get_ports, and get_instances -match_rtl_reg Commands

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_scan_chains

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup

Prerequisites: You must define the scan chain group with the add_scan_groups command prior to using this command.

Specifies a name for a pre-existing scan chain within the design.

Usage

add_scan_chains [-Internal | -Load_only] {***chain_name group_name input_pin output_pin***}...

For "dft -edt" context

add_scan_chains [-Internal] {***chain_name group_name input_pin output_pin***}...

Description

The add_scan_chains command defines a scan chain that exists in the design. A scan chain references the name of a scan chain group, which you must define prior to issuing this command.

A pre-existing scan chain is a serially connected set of scan cells that has been stitched together in a previous scan insertion operation, or that you created manually. In order for the tool to be able to correctly insert scan elements in the remainder of the design or to run rules checking on existing scan circuitry, you need to notify the tool of this circuitry, so it can treat those elements in the chain differently than the elements in the rest of the design.

When you use this command to notify the tool of a pre-existing scan chain, "dft -scan" context removes the scan cells in the scan chain from the eligible scan elements list. Because the tool removes the scan cells in the scan chain from the scan candidate list, the rules checker does not perform the usual scannability checks on those scan cells. However, the rules checker does perform additional rules checking based on the declaration of pre-existing scan elements that pertain to the checking of the validity and the operation of that scan chain.

If the design does have a pre-existing scan chain that you declare with the add_scan_chains command, you need to provide information on the operation of that scan chain in a test procedure file.

Arguments

- ***chain_name***

  A required repeatable string that specifies the name of the scan chain to add to the specified scan group.

- ***group_name***

  A required repeatable string that specifies the name of the scan chain group to add the specified scan chain to.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ***input_pin***

  A required repeatable string that specifies the input pin of the specified scan chain.

- ***output_pin***

  A required repeatable string that specifies the output pin of the specified scan chain.

- -Internal

  An optional switch that defines internal scan chains.

  Note

  The -Internal switch specifies compressed scan chains when using Tessent TestKompress with EDT logic. Do not use it to define uncompressed scan chains (chains connected directly to a primary input and primary output and not driven by or observed through the Tessent TestKompress logic). See the "Scan Chain Insertion" section in the *Tessent TestKompress User's Manual*.

- -Load_only (Not supported in the hybrid TK/LBIST flow)

  An optional switch that specifies the definition of a load-only scan chain, which is an uncompressed scan chain with the following properties:

  - The scan input is a primary input.
  - The scan output is an internal pin that is specified only to enable scan chain tracing.

  A load-only scan chain does not need to be unloaded, which saves chip-level pins and scan data. You can specify a load-only scan chain for special cases such as on-chip clock controllers that only need to be loaded but do not need to be unloaded.

  The tool automatically adds OX constraints on all cells of the load-only scans that are not also in observable scan chains, using the cell masking mechanism, because those cells are not unloaded and observed. In the generated patterns or mode_external testbench, the internal scan chain outputs are ignored. These chains can be added with other normal scan chains, EDT blocks, or both. However, the load-only chains must be uncompressed and loaded only from ports, not a decompressor.

  Note

  This switch does not support mapping EDT patterns to single bypass chain mode. Also, just like normal uncompressed chains, load-only uncompressed chains should not be added during the EDT IP phase.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example defines two scan chains (chain1 and chain2) that belong to the scan group (grp1):

```
add_scan_groups grp1 atpg.testproc
add_scan_chains chain1 grp1 si1 so1
add_scan_chains chain2 grp1 si2 so2
```

Example 2

The following example shows add_scan_chains commands similar to those Tessent TestKompress would write in the *_edt.dofile dofile, during the IP Creation Phase, for the two scan chains of the preceding example. By default, the top level module name "/my_design" is prefixed to the pin names.

```
add_scan_groups grp1 *_edt.testproc
add_scan_chains -internal chain1 grp1 /my_design/si1 /my_design/so1
add_scan_chains -internal chain2 grp1 /my_design/si2 /my_design/so2
```

Related Topics

add_scan_groups

delete_scan_chains

report_scan_chains

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_scan_groups

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup

Specifies a group name for a set of pre-existing scan chains in a design.

Usage

add_scan_groups {***group_name test_procedure_filename***}...

Description

Specifies a group name for a set of pre-existing scan chains in a design. Only one group name can be specified per session.

If the design has pre-existing scan chains, those scan chains must belong to a scan group. The add_scan_groups command also specifies the corresponding test procedure file for that scan group. A sequence of procedures in the test procedure file defines the operation of the scan chains within a scan group. The scan group must have a corresponding test procedure file.

When needed, you can also use the add_scan_groups command to provide initialization values for non-scan memory elements. If the design does not have pre-existing scan chains and you need to initialize some non-scan memory elements for scannability checking, you can specify "dummy" as the group_name along with a test_procedure_filename. Then, within the test procedure file, you can use the optional test_setup procedure to initialize the non-scan memory elements. For complete information, see "Test Procedure File" in the *Tessent Shell User's Manual*.

As a result, in dft -scan context, the tool models the non-scan memory elements as a TIE1 or TIE0.

Note

You must also define these same elements as non-scan using the add_nonscan_instances command.

Arguments

- ***group_name***

  A required string that specifies the name of the scan chain group that you want to add to the system.

- ***test_procedure_filename***

  A required string that specifies the name of the test procedure file that contains the information for controlling the scan chains in the specified scan chain group.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example defines a scan chain group, group1, which loads and unloads a set of scan chains, chain1 and chain2, by using the procedures in the test procedure file, *atpg.testproc*:

add_scan_groups group1 ../data/atpg.testproc
add_scan_chains chain1 group1 indata2 testout2
add_scan_chains chain2 group1 indata4 testout4

Contents of *atpg.testproc* file:

```
timeplate gen_tp1 = force_pi 0 ;
  measure_po 100 ;
  pulse clk_ 200 100;
  pulse tck 200 100;
  pulse shift_clock 200 100;
  pulse reset 200 100;
  pulse refclk 200 100;
  pulse ramclk 200 100;
  period 400 ;
end;

procedure shift = scan_group grp1 ;
  timeplate gen_tp1 ;
  cycle = force_sci ;
    measure_sco ;
    pulse shift_clock ;
  end;
end;

procedure load_unload = scan_group grp1 ;
  timeplate gen_tp1 ;
  cycle = force clk_ 0 ;
    force reset 0;
    force ramclk 0 ;
    force scan_enable 1 ;
  end ;
apply shift 16;
end;

procedure test_setup = scan_group grp1;
  timeplate gen_tp1;
  cycle = force test_logic_reset 1;
  end;
  cycle = force test_logic_reset 0;
  end;
end;
```

Related Topics

add_scan_chains

delete_scan_groups

report_scan_groups

read_procfile

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_procfile

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_scan_instances

Context: dft -scan, dft -test_points

Mode: setup

Adds specified instances to the scannable instance list.

Usage

add_scan_instances -INStances *instance_spec* | -Modules *module_spec*

Description

When inserting scan chains, the add_scan_instances command ensures that the tool includes specific sequential instances in the scan list. The tool generates the scan list during the scan identification process. If the specified instance does not pass the rules checking process, it is not included in the identified scan list.

You can use the delete_scan_instances command to remove the exception status that you have attached to an instance with the add_scan_instances command. In this case, although the instance is converted back into a non-scan instance, the effects of its previous unknown state on downstream instances is not reset.

Note

The add_scan_instances command can be used to override the default behavior for the S4 rule (when handling is set to Note), which forces cells with S4 violations to remain non-scan. Likewise, the add_scan_instances command can override the behavior of the S7 rule when its handling is set to Warning, which forces cells with S7 violations to remain non-scan. This provides a mechanism to override the S4 and S7 rule behavior for specific cells.

Arguments

- -Instances *instance_spec*

  An optional switch and value pair that causes the tool to include the specified instances during scan insertion. *instance_spec* can be a Tcl list of one or more instances or a collection of one or more instances. If the instance is hierarchical, all instances beneath it are also flagged as scannable.

- -Modules *module_spec*

  An optional switch and value pair that causes the tool to include all instances within the specified modules during scan insertion. *module_spec* can be a Tcl list of one or more modules or a collection of one or more modules.

  Note

  You may specify the name of a Verilog module or a Tessent library model.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example adds two sequential instances to the identified scan list and then runs the scan identification process.

```
add_scan_instances -instances i_1006 i_1007
add_scan_mode int_mode -chain_count 10
analyze_scan_chains
insert_test_logic
```

Related Topics

delete_scan_instances

report_seq_transparent_procedures

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_scan_mode

Context: dft -scan

Mode: analysis

Specifies a chain scan mode, which defines one or more scan chains.

Usage

add_scan_mode ***name***
[-chain_length { *integer* | unlimited} | -chain_count { *integer* | unlimited }]
[-type { internal | external | unwrapped }]
[-associate_chains *obj_spec*]
[-include_chain_families *obj_spec*]
[-include_elements *obj_spec*]
[{-enable_connections *obj_spec* [-enable_code *binary_string* | auto } |
-enable_dft_signal *string*]
[-si_timing { leading_edge | trailing_edge | any_edge }]
[-so_timing { leading_edge | trailing_edge | any_edge }]
[-edt_instances *obj_spec* |
{-ssh_instance *obj_spec* [-ssh_chain_groups *ssh_chain_group_list*]} |
{-si_connections *obj_spec* -so_connections *obj_spec* }]
[-si_associated_ports *obj_spec* -so_associated_ports *obj_spec*]
[-si_port_format *format*]
[-so_port_format *format*]
[-port_index_start_value *integer*]
[-port_scalar_index_modifier *integer*]
[-attribute_set_for_chain_assembly { *attribute_list* }]
[-single_attribute_set_chains { on | off }]
[-single_class_chains { On | Off }]
[-single_clock_domain_chains { On | Off }]
[-single_clock_edge_chains { On | Off }]
[-single_power_domain_chains { On | Off }]
[-single_cluster_chains { On | Off }]
[-single_wrapper_type_chains { On | Off }]
[-lockup_cell_type { latch | flop }]
[-si_lockup_cell_type { latch | flop }]
[-so_lockup_cell_type { latch | flop }]
[-unused_chain_connection_handling { auto | insert_pipelining_flop |
tie_off_chain_output | dont_touch }]
[-si_pipelining { wrapper_chains_only | all_chains | off }]
[-si_pipelining_type {non_scan_flop | holding_scan_flop}]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Description

Use this command to specify a chain scan mode, which defines one or more scan chains. Chain configurations specific to each scan mode are achieved by multiplexing scan path at certain inflection points computed by the scan insertion tool.

Arguments

- ***name***

  A required string that uniquely describes the scan mode. When using DFT signals, if the specified scan mode name corresponds to the name of a DFT scan mode signal name, then the command implicitly infers both the type (-type) and the enable signal (-enable_dft_signal) for the mode; if the specified scan mode name does not match the name of a DFT scan mode signal, then you must specify the enable signal using the option "-enable_dft_signal" and the command implicitly infers the mode type (-type) from it.

- -chain_length { *integer* | unlimited }

  An optional integer that specifies the target length of scan chains for a particular mode. By default, the tool imposes no limit (unlimited) on the chain length. The minimum scan chain length is 1.

- -chain_count { *integer* | unlimited }

  An optional integer that specifies the number of scan chains that should be constructed for the specified mode. By default, the chain count is set to the number of connections specified with the -si/so_connections or by the number of pins of the EDT controller(s) specified with the -edt_instances option, otherwise if the latter options were not used, then no limit (unlimited) is imposed on the chain count.

- -type { internal | external | unwrapped }

  An optional string and literal pair that specifies the type of scan mode. A mode of type external only considers wrapper scan elements in its default population, whereas internal and unwrapped modes consider all scan elements. For more information, see "Unwrapped Cores Versus Wrapped Cores" in the *Tessent Scan and ATPG User's Manual*.

- -associate_chains *obj_spec*

  An optional value that specifies a list of existing chains to associate in this mode but not included for scan distribution. It defaults to all usable existing chains when the mode type is "internal" or "unwrapped", or to wrapper usable existing chains when the mode type is "external".

  Only scan_element objects with a state "usable" get added to a population; accordingly, it may be necessary to explicitly set the "active_child_scan_mode" attribute to specific child mode values to include the existing chains in a scan mode population.

  This switch enables you to associate a specific subset of usable existing chains (or even an empty subset) with a scan mode instead of all the usable ones ([get_scan_elements -type chain]). You must include all existing chains of a mode connected to a given EDT controller

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

when you use -associate_chains. Also, never include EDT segments/chains from different child modes in the same parent scan mode.

By default, the tool automatically associates usable existing chains to every “internal” or “unwrapped” mode, but only associates wrapper existing chains to “external” modes.

- -include_chain_families *obj_spec*

  An optional value that specifies a list of chain families to include in this mode. It defaults to all chain families if no “-include_” option is specified.

- -include_elements *obj_spec*

  An optional value that specifies a list of scan elements to include in this mode. It defaults to all scan elements if no “-include_" option is specified.

- -enable_connections *obj_spec*

  An optional value that specifies one or more top-level port(s) or existing pin(s) to enable this mode. By default, the tool creates a scan_mode enable port with a name of the form “ts_stm%ds0”, where %d corresponds to the mode’s numeric ID. If you specify more than one connection, you must specify the same set of connections for each defined mode, each requiring a unique enable_code; specify the latter explicitly using “-enable_code <binary>” or let the tool assign it (“-enable_code auto”).

- -enable_code { *binary_string* | auto }

  An optional value that specifies a binary number (for example, “00110”) to be explicitly used to decode this mode using the specified enable_connections. By default or when the value “auto” is supplied, the tool automatically assigns a unique code to decode this mode based on the total number of scan modes. Each mode must have a unique code that uses all of the specified connections, or the tool reports an error.

- -enable_dft_signal *string*

  An optional value that specifies the name of a dft scan mode signal as defined with the command “get_dft_signal <name> -usage scan_mode” to enable this mode. The tool uses the scan mode type associated with the dft signal if the -type option was not specified, or checked for compatibility otherwise.

- -si_timing { leading_edge | trailing_edge | any_edge }

  An optional string and literal pair that provides the ability to control the capture edge of the first scan cell in a chain. The tool inserts a lockup (re-timing) cell between the scan chain input pin and the first scan cell in chain to control the capture edge of the first cell as leading or trailing edge. No lockup cell is inserted if “any_edge” is provided as the argument. The default edge type is “leading_edge”.

- -so_timing { leading_edge | trailing_edge | any_edge }

  An optional string and literal pair that provides the ability to control the change edge of the last scan cell in a chain for this scan mode. The tool inserts a lockup (re-timing) cell between the scan chain output pin and the last scan cell in chain to control the change edge of the last

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

cell as leading or trailing edge. No lockup cell is inserted if “any_edge” is provided as the argument. The default edge type is “trailing_edge”.

- -edt_instances *obj_spec*

  An optional value that specifies EDT instances to connect the scan chains to. When the tool detects core descriptions for the specified instances, it automatically sets the -si/so_connections to the instance’s scan pins.

  When you use this option, the tool automatically infers the -chain_count value to be the number of available scan connection(s) on the specified EDT controller(s), and reports a message, as follows:

```
//  command: add_scan_mode int_mode -type internal -si_pipelining off -edt_instance
// {test[0][0].my_white/point_scalar_mult_rtl_tessent_edt_lbist_c2_inst}
//  Reading core description file ./tsdb_outdir/instruments/
point_scalar_mult_rtl_edt_lbist.instrument/
point_scalar_mult_rtl_tessent_edt_lbist_c2.tcd
//  Setting the chain count to the number of specified scan connections (50).
//  Automatically inferring '-enable_dft_signal int_mode' from the specified scan
//  mode name.
```

  This inferred mode chain_count now also accounts for the EDT scan connections from families included in the mode, as follows:

```
//  Increasing the chain count to 140 in order to account for chain family EDT scan
//  connections (90)
```

- -ssh_instance *obj_spec* [-ssh_chain_groups *ssh_chain_group_list*]

  Optional string and literal pair that enables you to identify a single ScanHost node instance for the add_scan_mode command. When the tool detects a core description for the specified instance, it automatically sets the -si/so_connections to the scan pins of the instance. Use this option when you want to create uncompressed scan chains and have them driven directly by the SSH. If you are creating compressed scan chains, connect them to the EDT instance using the -edt_instances switch and not directly to the SSH.

  -ssh_chain_groups *ssh_chain_group_list* — specify one or more subsets of the ScanHost connections using the SSH chain group ID(s). All ScanHost connections are selected when no SSH chain group ID is specified. When there is more than one ChainGroup associated with an SSH, include all the ChainGroups to be included in the scan mode.

- -si_connections *obj_spec* -so_connections *obj_spec*

  Optional string and literal pairs that specify SI/SO connections to existing pins or ports to be used first when connecting newly allocated scan chains. The specified values can be a Tcl list of one or more port/pin names, a collection of one or more pin/port objects, or a port/pin Verilog name (of a certain width). They may be existing top-level input port(s) for -si_connections and existing top-level output port(s) for -so_connections, or alternatively -si_connections may correspond to existing output pin(s) and -so_connections to existing input pin(s). By default, -chain_count is implicitly set to the number of si/so_connections

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

specified; if this is not the required behavior, then you should also set an explicit -chain_count.

- -si_associated_ports *obj_spec* -so_associated_ports *obj_spec*

  Optional string and literal pairs that specify existing top-level port(s) associated to the internal connections identified using -si/so_connections. These are used in the .tcd file to describe the scan ports of the chains at the core boundary.

- -si/so_port_format *format*

  An optional string that specifies the port format. The default si/so_port_format is set to "ts_%s_si/so[%d]" when there are more than one scan mode, otherwise it defaults to "ts_si/so[%d]". The mode placeholder "%s" gets replaced by the chain mode name, whereas the index placeholder "%d" gets substituted by the chain index. A precision number may be used when specifying the index placeholder, for example the format "lsi_si%2d" would cause port names such as "lsi_si00", "lsi_si01" to be created. Note that leading zeros are inserted automatically when needed making "%2d" implicitly equivalent to "%02d".

- -port_index_start_value *integer*

  An optional integer that specifies the starting port index. The starting index is 0 by default, but can be overridden by this option.

- -port_scalar_index_modifier *integer*

  An optional switch and integer that specifies the increment used between each chain port indices. The default is 1.

- -attribute_set_for_chain_assembly { *attribute_list* }

  An optional switch used to specify one or more string attributes to control scan chain assembly. When you use this switch and specify several string attributes, the tool creates clusters for every value combination of the specified attributes. For example, assume that you have defined the following quadrant and testpoint_type name attributes, and that all scan elements are attributed accordingly:

```
>register_attribute \
-name quadrant \
-value_type string \
-object_types scan_element \
-enum { NE NW SE SW }

>register_attribute \
-name testpoint_type \
-value_type string \
-object_types scan_element \
-enum { control observation }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Scan elements that are not test points are not assigned a “testpoint_type” attribute value, whereas all scan elements are assigned a “quadrant” attribute value. This implies that each scan element has one of the following attribute value combinations:

Table 3-16. Attribute Value Combinations

“quadrant” attribute value	“testpoint_type” attribute value
NE	control
NE	observation
NE	--
NW	control
NW	observation
NW	--
SE	control
SE	observation
SE	--
SW	control
SW	observation
SW	--

Accordingly, you can specify a mode that creates separate scan chains for each of the above 12 attribute value combinations as follows:

```
>add_scan_mode foo \
-attribute_set_for_chain_assembly { quadrant testpoint_type }
```

- -single_attribute_set_chains { on | off }

  An optional switch and literal pair that restricts a scan chain to a single set of attribute values. The default is on.

- -single_class_chains { on | off }

  An optional switch and literal pair that controls whether to mix wrapper and non-wrapper cells in a single chain. The default is off, which means the tool distributes wrapper and internal cells onto the same chains.

- -single_clock_domain_chains { on | off }

  An optional switch and literal pair that restricts a scan chain to a single clock domain. The default is off.

- -single_clock_edge_chains { on | off }

  An optional switch and literal pair that restricts a scan chain with flops having a single edge. The default is off.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -single_power_domain_chains { on | off }

  An optional switch and literal pair that restricts a scan chain to a single power domain using the built-in attribute "power_domain_island". The default is off. For attribute details, see the table "Built-in Attributes on Scan Element (SE) Object Type" in the "Scan Element Object Type" on page 3813.

- -single_cluster_chains { on | off }

  An optional switch and literal pair that restricts a scan chain to a single cluster using the built-in attribute "cluster_name". The default is on. For usage examples, see Scan Mode and Scan Insertion for Unwrapped Core sections in the *Tessent Scan and ATPG User's Manual*. For attribute details, see the table "Built in Attributes on Scan Element (SE) Object Type" in the Scan Element Object Type section of the *Tessent Shell Reference Manual*.

- -single_wrapper_type_chains { on | off }

  An optional switch and literal pair that specifies that "-single_class_chains ON" controls whether to mix input wrapper and output wrapper cells in the same chain. The default is off.

- -lockup_cell_type { latch | flop }

  An optional switch and string that controls the type of lockup (re-timing) cell to use between timing-constrained scan elements of new scan chains. The default cell type is latch.

- -si_lockup_cell_type { latch | flop }

  An optional switch and string that controls the type of lockup (re-timing) cell to use between the scan chain input pin and the first scan cell. The default cell type is latch.

- -so_lockup_cell_type { latch | flop }

  An optional switch and string that controls the type of lockup (re-timing) cell to use between the scan chain output pin and the last scan cell. The default cell type is latch.

  Note

  By default, the tool adds a pipeline flop in front of the wrapper chains to facilitate launching a transition from the first scan cells in the wrapper chains.

- -unused_chain_connection_handling {auto | insert_pipelining_flop | tie_off_chain_output | dont_touch}

  An optional switch and literal pair that selects how to handle unused scan in and scan out connections. Options are the following:

  auto — selects "insert_pipelining_flop" for EDT controllers with bypass chains, "tie_off_chain_output" for all other EDT controllers, and "dont_touch" for other connections. This option is the default.

  insert_pipelining_flop — inserts scan chains composed of a single pipelining flop.

  tie_off_chain_output — ties off the so_connection or the EDT compactor input when you specify an EDT instance.

  dont_touch — do not touch the unused scan chain connections.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -si_pipelining {wrapper_chains_only | all_chains | off }

  An optional switch and literal that controls whether the tool inserts a pipelining flop at the beginning of each chain if the first scan element of the chain does not have the attribute "no_si_pipelining" set and is not an OCC segment. When set to all_chains, the tool adds a pipelining flop to all scan chains. When set to wrapper_chains_only, the tool adds pipeline flops only at the beginnings of chains that begin with a wrapper flop in internal or external mode. The default is wrapper_chains_only when you have specified "set_scan_insertion_options -se_pipeline_count 0" (the default value for this switch), but the default changes to all_chains when you have specified "set_scan_insertion_options -se_pipeline_count 1".

  Input wrapper cells and output wrapper cells are held in shift during internal test and external test respectively, while the internal flops are capturing normally and able to launch a transition (that is, no need for pipeline). Therefore, when you select the "wrapper_chains_only" option, the tool only adds pipeline flops at the beginning of chains that begin with a wrapper flop, such that a mixed internal chain (with core and wrapper elements) only gets a pipelining flop if its first element is a wrapper scan element.

- -si_pipelining_type {non_scan_flop | holding_scan_flop}

  An optional switch and string that controls whether to implement pipelining using a simple non-scan data flop or a holding flop constructed using a scan flop with its output connected to its data input. The default is holding_scan_flop.

  The requirement for the holding pipeline comes from LOS (Launch-Off-Shift) patterns generated by TestKompress. A pipeline flop is usually added at the beginning of every chain so that it can be used to launch a transition. When generating LOS patterns, if you add a pipeline flop, you can get coverage for the LOS patterns, but the first flop always captures an X, which impacts compression. This may not affect coverage but is likely to impact pattern count. The addition of hold flops instead of simple pipeline flops at the beginning of scan chains helps prevent this issue.

Related Topics

create_scan_chain_family

delete_scan_modes

get_scan_modes

set_scan_insertion_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_scan_segments

Context: dft -scan, dft -test_points -no_rtl

Mode: setup

Adds one or more existing scan segment(s).

Usage

add_scan_segments ***scan_segment_name***
[-no_trace] [-length *integer*] [-on_module *obj_spec*]
-si_connections *obj_spec* -so_connections *obj_spec*
-clock_pins *obj_spec* [-clock_update_edges { leading | trailing }...]
-scan_enable_pins *obj_spec* [-scan_enable_inversion { false | true }...]
[-set_pins *obj_spec* [-set_inversion { false | true }...]]
[-reset_pins *obj_spec* [-reset_inversion { false | true }...]]
[-ste_pins *obj_spec* [-ste_inversion { false | true }...]]
[-se_clock *clock_port_name* ...] [-se_pipeline_count { 0 | 1 }...]

Description

Use this command to add one or more existing scan segment(s).

See Pre-Existing Scan Segments and Scan Insertion for Wrapped Core in the *Tessent Scan and ATPG User's Manual* for usage examples.

Arguments

- ***scan_segment_name***

  A required string that specifies the name of a scan segment.

- -no_trace

  An optional switch that specifies to skip tracing even if a scan segment procedure file was read in; it can only be set if -length is specified. Defaults to OFF.

  Note

  When your design has unconfined scan segments, you must *not* use the "-no_trace on" option. Without tracing, the tool treats all elements inside the scan segment container as part of the chain, which omits elements when the segment spans multiple instances. Tracing the segment enables the tool to properly identify the complete list of instances in the segment as well as the scan in and scan out pins along the path.

- -length *integer*

  An optional integer that specifies the length of the scan segment; a non-null value is required if -no_trace is used.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -on_module *obj_spec*

  An optional switch and value pair that specifies an anchor module from which port specifications are relative to. Defaults to the top module.

- -si/so_connections *obj_spec*

  A required switch and value pair that specifies connections to existing SI/SO pins or ports. The specified value can be a Tcl list of one or more port/pin names, a collection of one or more pin/port objects, or a port/pin Verilog name (of a certain width). The specified values corresponds to existing top-level input port(s) or output pin(s) for *-si_connections* and existing top-level output port(s) or input pin(s) for *-so_connections*.

- -clock_pins*obj_spec*

  A required switch and value pair that specifies a collection of list of pins. Once provided, the tool can perform S rule checks for the clocks from the scan segment boundary. The first and the last elements of the collection correspond to the clocks for the first and last scan cells of a scan segment.The intermediate elements correspond to the clocks for the remaining scan cells in the scan segment. If there is just one element, then the tool assumes that the scan segment is using one clock.

- -clock_update_edges { <u>leading</u> | trailing }...

  An optional repeatable switch and literal pair that specifies a list of clock edges at which the scan cells are capturing or the output is changing (for example, LE or TE). It is a list of edges. The first and the last elements of the list correspond to the edges for the first and the last scan cells in a sub-chain. There is a no need to have more than two elements in the list. If there are more than two elements, the tool ignores them. If there is only one element, then the tool assumes the first and last scan cells to change at the same edge.

- -scan_enable_pins *obj_spec*

  A required switch and value pair that specifies a collection or list of scan_en pins driving the scan segment.

- -scan_enable_inversion { <u>false</u> | true }...

  An optional repeatable switch and literal pair that specifies whether there is inversion in the scan_en pin between the cell and the segment container. It is a list of boolean values.

- -set_pins *obj_spec*

  An optional switch and value pair that specifies a collection or list of set pins for the scan cells in an existing segment.

- -set_inversion { <u>false</u> | true }...

  An optional repeatable switch and literal pair that specifies whether it is an active high or low set pin. The argument returns a list of Boolean values. The number of elements should match the collection size of set pins.

- -reset_pins *obj_spec*

  An optional switch and value pair that specifies a collection or list of reset pins for the scan cells in an existing segment.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -reset_inversion { false | true }...

  An optional repeatable switch and literal pair that specifies whether it is an active high or low set pin. The argument returns a list of Boolean values. The number of elements should match the collection size of set pins.

- -ste_pins *obj_spec*

  An optional switch and value pair that specifies a collection or list of scan test enable pins driving the scan segment.

- -ste_inversion { false | true }...

  An optional repeatable switch and literal pair that specifies whether it is an active high or active low scan test enable pin. The argument returns a list of Boolean values. The number of elements should match the collection size of set pins. The default is false.

- -se_clock *clock_port_name* ...

  An optional repeatable switch and literal pair that specifies the clock port name associated with the scan_en pin of the segment container. All the sequential cells driven by the scan_en pin must be part of the clock domain of the specified clock port name; otherwise, an empty string should be used. The default clock name is “”.

- -se_pipeline_count { 0 | 1 }...

  An optional repeatable switch and literal pair that specifies the pipeline count inside the segment container for the scan_en pin. It is a list of enumerated (0, 1) integer values.

Related Topics

report_scan_segments

delete_scan_elements

get_scan_elements

report_scan_elements

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_schematic_callout

Context: all contexts

Mode: all modes

Add an informational callout to a Tessent Visualizer schematic window.

Usage

add_schematic_callout ***object_spec*** **-message** ***string*** [-collapsed] [-display *window_name*]

Description

Adds informational callouts with the specified message string to objects in a Tessent Visualizer schematic window.

The string can contain "\n" to break the message into multiple lines.

Arguments

- ***object_spec***

  A Tcl list or Tcl collection of one or more names of pin, port, pseudo_port, gate_pin, instance, or net objects.

  Tip

  If your ***object_spec*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- **-message string**

  A required string that specifies the message to be displayed in the callout. The string supports the use of the newline character sequence "\n" to break the message into multiple lines.

- -collapsed

  Add the callouts as collapsed. If the mouse pointer is hovered over the callout, a tooltip with the callout message is displayed.

- -display *window_name*

  An optional switch and literal that specifies the Tessent Visualizer schematic window in which to display the callout. The valid options for the *window_name* argument are as follows:

  flat_schematic — A literal that specifies the Flat Schematic window.

  hierarchical_schematic — A literal that specifies the Hierarchical Schematic window.

  ijtag_network — A literal that specifies the IJTAG Network window.

  icl_schematic — A literal that specifies the ICL Schematic window.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

By default, the callout is added to the Flat Schematic window.

Examples

Example 1

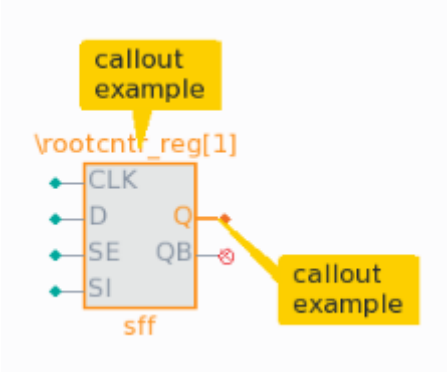
This example shows how to add a callout with the message:

```
callout
example
```

to a gate and a pin in the Hierarchical Schematic window:

**add_schematic_callout { U_0/CSEEblk/rootcntr_reg[1] U_0/CSEEblk/rootcntr_reg[1]/Q } \
-message "callout\nexample" -display hierarchical_window**

Figure 3-27. Schematic Callouts

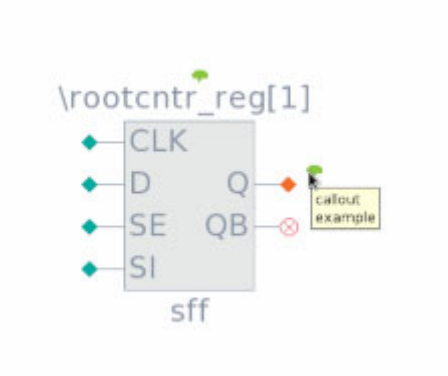


Example 2

This example shows the callouts from Example 1 added as collapsed.

**add_schematic_callout { U_0/CSEEblk/rootcntr_reg[1] U_0/CSEEblk/rootcntr_reg[1]/Q } \
-message "callout\nexample" -collapsed -display hierarchical_window**

Figure 3-28. Schematic Callouts (Collapsed)



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_schematic_connections

Context: all contexts

Mode: setup, analysis

Displays design objects in the Tessent Visualizer schematic window and enables you to trace visually through the design.

Usage

add_schematic_connections ***object_spec*** {-to ***object_spec***
| -forward {decision_point | end_point}
| -backward {decision_point | end_point}}
[-display *window_name*]
[-distance_limit *distance*]
[-connections_limit *max*]
[-highlight ***color***]

Description

Displays design objects in the Tessent Visualizer Flat or Hierarchical Schematic window and enables you to trace visually through the design.

Arguments

- ***object_spec***

  A Tcl list or Tcl collection of one or more names of pin, port, pseudo_port, gate_pin, or net objects. This is the tracer starting point.

  Tip

  If your ***object_spec*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- -to *object_spec*

  An optional Tcl list or Tcl collection of one or more names of pin, port, pseudo_port, gate_pin, or net objects. This is the tracer target point when the tracer needs to find connectivity between pairs of design pins. The -to option is mutually exclusive to the -forward and -backward options.

- -forward {decision_point | end_point}

  Trace forward from the starting pin(s) using the chosen strategy. The -forward option is mutually exclusive to the -to and -backward options.

- -backward {decision_point | end_point}

  Trace backward from the starting pin(s) using the chosen strategy. The -backward option is mutually exclusive to the -to and -forward options.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -distance_limit **int**

  The threshold on hierarchy levels for tracing in the hierarchical schematic window (default: 100).

- -connections_limit **int**

  The maximum number of connections to display (default: 100).

- -highlight ***color***

  One of none, green, blue, orange, yellow, red, pink, turquoise, or brown. “none” means to reset to the default color.

- -display *window_name*

  An optional switch and literal that specifies the Tessent Visualizer schematic window in which to perform the tracing. The valid options for the *window_name* argument are as follows:

  flat_schematic — A literal that specifies the Flat Schematic window.

  hierarchical_schematic — A literal that specifies the Hierarchical Schematic window.

  ijtag_network — A literal that specifies the IJTAG Network window.

  icl_schematic — A literal that specifies the ICL Schematic window.

  By default, the callout is added to the Flat Schematic window.

Examples

This example shows how to trace forward from a pin to the path’s end point and highlight the nets on that path in red:

```
add_schematic_connections U_0/CSEEblock/rootcntr_reg[1]/Q -forward end_point \
-highlight red
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_schematic_objects

Context: all contexts

Mode: all modes

Displays provided objects in the specified Tessent Visualizer schematic window.

Usage

add_schematic_objects ***object_spec***
[-display *window_name*]
[-highlight *color*]

Description

Displays provided objects in the specified schematic window, optionally highlighting them with one of the predefined colors. The command behavior for multiple objects is equivalent to issuing subsequent commands for each object, for example:

```
add_schematic_objects {a b c} -display flat_schematic
```

is equivalent to:

```
add_schematic_objects a -display flat_schematic
add_schematic_objects b -display flat_schematic
add_schematic_objects c -display flat_schematic
```

If the provided object is defined in the context of the flat model and the -display option is set to Hierarchical Schematic, an automated translation is applied if possible. If the object arrives from the hierarchical model and is to be displayed in the Flat Schematic window, an automated translation is applied if possible.

Arguments

- ***object_spec***

  A Tcl list or Tcl collection of one or more names of pin, port, pseudo_port, gate_pin, instance, or net objects.

  Tip

  If your ***object_spec*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- -display *window_name*

  An optional switch and literal that specifies the Tessent Visualizer schematic window in which to display the object. The valid options for the *window_name* argument are as follows:

  flat_schematic — A literal that specifies the Flat Schematic window.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

hierarchical_schematic — A literal that specifies the Hierarchical Schematic window.

ijtag_network — A literal that specifies the IJTAG Network window.

icl_schematic — A literal that specifies the ICL Schematic window.

By default, the object is added to the Flat Schematic window.

- -highlight ***color***

  One of none, green, blue, orange, yellow, red, pink, turquoise, or brown. “none” means to reset to the default color.

Examples

add_schematic_objects { bsr_i1 bsr_i2 } -display hierarchical_schematic

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_schematic_path

Context: all contexts

Mode: all modes

Display the specified path in one of the Tessent Visualizer schematic windows.

Usage

add_schematic_path ***object_spec*** **-direction** {***forward*** | ***backward***} [-display *window_name*] [-highlight ***color***]

Description

Displays the specified path in the Tessent Visualizer Flat or Hierarchical Schematic window.

Arguments

- ***object_spec***

  A Tcl list or Tcl collection of one or more names of pin or gate_pin objects. Given lists of pins must represent existing connections in the hierarchical or flat data model.

  Tip

  If your ***object_spec*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- **-direction** {**forward** | **backward**}

  A required switch and literal pair that specifies the direction of the path provided in ***object_spec***.

- -display *window_name*

  An optional switch and literal that specifies the Tessent Visualizer schematic window in which to display the path. The valid options for the *window_name* argument are as follows:

  flat_schematic — A literal that specifies the Flat Schematic window.

  hierarchical_schematic — A literal that specifies the Hierarchical Schematic window.

  ijtag_network — A literal that specifies the IJTAG Network window.

  icl_schematic — A literal that specifies the ICL Schematic window.

  By default, the path is added to the Flat Schematic window.

- -highlight ***color***

  One of none, green, blue, orange, yellow, red, pink, turquoise, or brown. "none" means to reset to the default color.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

This example shows how to add a path to the Flat Schematic window and highlight it with the default color:

```
add_schematic_path { U_0/CSEEblock/rootcntr_reg[1]/Q U_0/CSEEblock/U50/A0 } \
-direction forward
```

Example 2

This example shows how to add two paths to the Hierarchical Schematic window and highlight them in yellow:

```
add_schematic_path { U_0/CSEEblock/rootcntr_reg[1]/Q \
U_0/CSEEblock/U50/A0 U_0/CSEEblock/U50/Y U_0/CSEEblock/U49/A1 } \
-direction forward -display hierarchical_window -highlight yellow
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_simdut_fault

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Injects a stuck-at-0 fault or a stuck-at-1 fault into the specified signal for SimDUT simulation.

Usage

add_simdut_fault **-signal** *signal_name* **-fault [0 | 1]**

Description

SimDUT uses the Verilog simulation capabilities to force the signal to 0 or 1. The Verilog capability has the following behavior: When you specify a pin pathname for the fault you want to inject, Verilog forces the associated signal to the specified value.

When you inject a fault at a specific pin pathname, SimDUT actually injects the fault at the whole net, not only on the specific pin. The fault acts like a fault injected on the upstream driver pin of the associated net. If you specify the input port of a cell and this port is attached to a net with fanout, then the fault acts like a fault injected on the driver port of that net.

When you run a test, SimDUT simulates the injected fault as if it were an actual defect in silicon, collects the simulated failing cycles, and returns the results.

You can run this command multiple times. When you specify a particular signal more than once, the last command that specifies the signal overrides all previous specifications. For example, if you first add a stuck-at-0 fault to signal A and later add a stuck-at-1 to signal A, SimDUT only adds the stuck-at-1 fault to signal A.

Arguments

- **-signal** *signal_name*

  A required switch and string pair that specifies the name of a signal from the Verilog netlist. Refer to "SimDUT Signal Naming Convention" in *Tessent SiliconInsight User's Manual for Tessent Shell* for more information

- **-fault [0 | 1]**

  A required switch and literal that specifies the fault type as follows:

  0 - Specifies stuck-at fault type SA0.

  1 - Specifies stuck-at fault type SA1.

Examples

The following example injects an SAO fault into the SIMDUT_TB.DUT_inst.memory00.q_a signal.

```
add_simdut_fault -signal SIMDUT_TB.DUT_inst.memory00.q_a -fault 0
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

delete_simdut_fault

report_simdut_faults

execute_cdp_test

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_simulation_context

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist.

Creates a new user-defined simulation context.

Usage

add_simulation_context ***context_name*** [-copy_from *source_context_name*] [-silent]

Description

The add_simulation_context command enables you to create a new user-defined simulation context in which the tool can perform simulation and introspection. Initially, the tool propagates tied values, such as power and ground, and then sets the remaining simulation values to X. The command automatically sets the new simulation context to be the current simulation context.

Arguments

- ***context_name***

  A required string that specifies the name of the context. The ***context_name*** has to be different than one of the predefined context names listed in Table 8-6 on page 2434.

- -copy_from *source_context_name*

  An optional switch and string that creates a new simulation context by copying an existing simulation context. Using this switch, you can bypass the initialization work done by the tool and build up a new simulation context. See “Example 3” on page 388.

- -silent

  An optional switch that turns off the display of the message:

```
// 'sim_x' is the current simulation context.
```

Examples

Example 1

The following example shows how to create a new simulation context:

SETUP> add_simulation_context myContext

```
//  'myContext' is the current simulation context.
```

SETUP> get_simulation_context_list -user_defined

```
{myContext}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example shows creates a new simulation context by generating an empty simulation context and subsequently copying another simulation context:

```
add_simulation_context $context
copy_simulation_context -from $other_context
set_current_simulation_context $context
```

Example 3

The following example illustrates using the -copy_from switch to create a new simulation context by copying an existing simulation context:

```
add_simulation_context $context -copy_from $other_context
set_current_simulation_context $context
```

Related Topics

add_simulation_forces

copy_simulation_context

delete_simulation_contexts

delete_simulation_forces

get_current_simulation_context

get_simulation_context_list

get_simulation_value_list

report_simulation_contexts

report_simulation_forces

set_current_simulation_context

simulate_clock_pulses

simulate_forces

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_simulation_forces

Context: dft, patterns

Mode: setup, analysis

Prerequisite: The flat model must already exist.

Forces one or more *gate_pin* objects to the specified value(s).

Usage

add_simulation_forces ***obj_spec*** -values ***values***

Description

The add_simulation_forces command forces the specified *gate_pin* object(s) to the specified force value(s) within the current simulation context. The actual simulation is not triggered until you issue the simulate_forces or simulate_clock_pulses command. If a specified *gate_pin* object has already been forced by an earlier add_simulation_forces command, and the tool has not yet simulated the previous force, the new force value overwrites the existing one in case of a conflicting value. Note that adding a simulation force also updates the gate_pin attribute named simulation_value.

This command is permitted only for user-defined simulation contexts.

Arguments

- ***obj_spec***

  A required value that specifies one or more *gate_pin* objects.

- **-values *values***

  A required switch and list of values, which specifies the force value applied to the specified *gate_pin* objects. ***values*** can be a Tcl list of values or a string of values.

  The number of ***values*** must be one or must match the number of specified *gate_pin* objects. Valid values are: X, 0, 1, Z.

  If only one value is specified, then the tool applies this value to all gates in the *gate_pin* collection.

Examples

The following example shows the effect of adding a force on a particular *gate_pin* object, and then triggering the tool to simulate that force:

```
ANALYSIS> set_current_simulation_context myContext
ANALYSIS> add_simulation_forces [get_gate_pins I1] -value 1
ANALYSIS> report_simulation_forces

Value   Object
------  ------
1       I1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ANALYSIS> get_simulation_value_list [get_gate_pins I1]

```
{ X}
```

ANALYSIS> simulate_forces
ANALYSIS> get_simulation_value_list [get_gate_pins I1]

```
{ 1 }
```

Related Topics

add_simulation_context

copy_simulation_context

delete_simulation_contexts

delete_simulation_forces

get_simulation_context_list

get_simulation_value_list

report_simulation_contexts

report_simulation_forces

simulate_clock_pulses

simulate_forces

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_synchronous_clock_group

Context: dft, dft -scan, dft -test_points, patterns

Mode: setup

Defines a set of clocks that are synchronous (compatible) with each other.

Usage

add_synchronous_clock_group {***object_spec*** | **-all**} [-scope {capture | shift | all}]

Note

Synchronous clock groups have no impact on scan chain analysis, scan insertion, test point identification, and EDT IP insertion (in the dft -test_points and dft -edt sub-contexts).

Description

You can define the set for shift, for capture, or for both. Clocks defined as synchronous during shift affect lockup latch insertion. Clocks defined as synchronous during capture primarily impact test point clock selection.

The add_synchronous_clock_group command can define specified interacting clocks as synchronous during capture cycles (that is, there is no clock skew between them). By specifying synchronous clocks explicitly, ATPG is permitted to pulse them at the same time to potentially reduce the generated test set size and improve test coverage.

The add_synchronous_clock_group command can also define a group of balanced clocks as synchronous during shift and load_unload cycles. This prevents DRC from issuing violations that apply to different shift clocks for elements that are clocked by the balanced clocks (for example, T24).

You can use “add_synchronous_clock_group -scope shift -all” to avoid inserting lockup cells between synchronous clock groups. This also permits the tool to create LE->TE chains.

Note

Simulation mismatch errors are likely if you erroneously define clocks as synchronous that are not.

During MemoryBIST insertion, compatible memories including multi-port memories, can be assigned to the same memory controller. From a clocking perspective, memories are compatible when:

- All memory clock pins are connected to the same clock label.

OR

- Memory clock pins are connected to different clock labels, but are assigned to the same synchronous clock group and have the same frequency.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Compatible memories can be assigned to the same controller as long as all partitioning rules are met. If memories are assigned to a different controller, it can be for reasons other than their clock source.

Arguments

- ***object_spec***

  A Tcl list or collection of objects:

  - Net, port, or pseudo-ports that correspond to defined clocks.
  - Design instances that correspond to defined OCC core instances either directly by the add_core_instances command or indirectly by the import_scan_mode command.

    The tool performs an OCC instance check when in SETUP mode. The tool displays an error message like the following example when it cannot map such instances to a defined OCC core instance:

    ```
    Error: <name> is not a defined clock or OCC core instance.
    ```

- **-all**

  A switch that specifies all defined clocks.

- -scope {capture | shift | all}

  An optional argument and literal pair that specifies whether the clock groups are defined for capture, for shift, or for both. By default, groups are defined only for capture.

Examples

Example 1

The following example creates clock groups from different sets of compatible clocks:

SETUP> report_clocks

```
User-defined Clocks (4):
========================

  Sync and Async Source Clocks
  ============================
   ----  ---------  -----------  --------
   Name  Off State  Constraints  Internal
   ----  ---------  -----------  --------
    I1       0                      No
    I2       0                      No
    I3       0                      No
    I4       0                      No
```

SETUP> add_synchronous_clock_group {I1 I2}
SETUP> report_synchronous_clock_groups

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
User-defined Synchronous Clock Group
=====================================
//  No.  Clocks (Labels)   Scope
//  ---  ---------------  -------
//   1   I1               capture
//       I2
```

SETUP> add_synchronous_clock_group {I3 I4}
SETUP> report_synchronous_clock_groups

```
User-defined Synchronous Clock Groups
=====================================
//  No.  Clocks (Labels)   Scope
//  ---  ---------------  -------
//   1   I1               capture
//       I2
//   2   I3               capture
//       I4
```

SETUP> add_synchronous_clock_group -scope shift {I1 I2}
SETUP> report_synchronous_clock_groups

```
User-defined Synchronous Clock Groups
=====================================
//  No.  Clocks (Labels)   Scope
//  ---  ---------------  -------
//   1   I1               capture
//       I2
//   2   I3               capture
//       I4
//  ---  ---------------  -------
//   1   I1                shift
//       I2
```

Note

For the following command, the capture clock group already exists. Therefore, the shift clock group is not created when it is specified with “-scope all”.

SETUP> add_synchronous_clock_group -scope all {I3 I4}

```
// Error: This user-defined synchronous clock group already exists in the
// scope of capture.
```

SETUP> report_synchronous_clock_groups

```
User-defined Synchronous Clock Groups
=====================================
//  No.  Clocks (Labels)   Scope
//  ---  ---------------  -------
//   1   I1               capture
//       I2
//   2   I3               capture
//       I4
//  ---  ---------------  -------
//   1   I1                shift
//       I2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

There are only the OCC instances in SETUP mode but no actual clocks from the OCCs. In the SETUP mode report, none of the clocks has a label name different from the clock name. The report does not have a Labels column.

```
SETUP> add_synchronous_clock_group {occ_inst_1 tck rst}
SETUP> report_synchronous_clock_groups
User-defined Synchronous Clock Groups
=============================================
// No. Clocks  OCC Inst    Scope
// --- ------ ----------- -------
//  1         occ_inst_1  Capture
//     tck
//     rst
```

In ANALYSIS mode, the tool defines clocks automatically on the OCC output. Clocks are pseudo ports inside the tool. In the ANALYSIS mode report, the OCC clock has a different label name from its clock name. The report has a Labels column.

```
SETUP> add_synchronous_clock_group {occ_inst_1 tck rst}
...
ANALYSIS> report_synchronous_clock_groups
User-defined Synchronous Clock Groups
==============================================================
// No. Clocks              Labels  OCC Instances   Scope
// --- ----------------- -------- -------------- -------
//  1  occ_inst_1/cut_ck   c1     occ_inst_1     Capture
//     tck
//     rst
```

Related Topics

add_clocks

delete_synchronous_clock_group

get_synchronous_clock_groups

report_clock_domains

report_synchronous_clock_groups

set_clock_restriction

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_tcl_variables_to_flat_model

Context: All

Mode: All

Adds the specified Tcl variables to the flat model file.

Usage

add_tcl_variables_to_flat_model ***tcl_var_name_list***

Description

Use this command to store the values of the specified Tcl variables to the flat model file. Running the add_tcl_variables_to_flat_model command schedules the specified variables to be stored in the flat model file when it is created. Then, they get automatically restored into their respective namespaces when you read the flat model file with the read_flat_model command. If the namespace does not exist, the tool automatically creates it. You can use this command multiple times to add more variables.

Note

The write_flat_model command automatically stores all Tcl variables from the global namespace in the flat model file. This command enables you to store variables from other namespaces.

Tip

This command is useful to store variables that hold the state of registered Tcl commands you may create, especially when the command is only allowed to be used in setup mode. That way, when you restore the flat model, which automatically sets the system mode to analysis, your settings used before writing out the flat model are restored.

Arguments

- ***tcl_var_name_list***

  A required string that specifies a Tcl list of variable names to be stored in the flat model file. Use the full namespace prefix of each Tcl variable you want to store. When calling this command within a namespace, any variables not starting with "::" are assumed to be defined in or below the current namespace, as if you had used "[namespace current]::" in front of its name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example sets up a Tcl variable to be stored in the flat model file when it is created so that it gets restored when the flat model is read back into the tool:

```
namespace eval yourNS {
  variable yourVar
  set yourVar "Value"
  add_tcl_variables_to_flat_model {yourVar}
}
...
write_flat_model yourFile.flat
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_tied_signals

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup

Adds a value to floating signals or pins.

Usage

add_tied_signals {**0** | **1** | **X** | **Z**} ***floating_object_name***... [-Pin]

Description

The add_tied_signals command assigns a specific value to not-clearly-defined floating signals or pins. If there are floating signals or pins in the design, a warning displays when you leave the setup mode. If you do not assign a specific value, the tool ties the signal or pin values to the default value. To change the default tied value, use the set_tied_signals command.

When you add_tied_signals or pins, the tool places them into the user class. When the netlist ties signals or pins to a value, the tool places them into the system class.

Note

The tool does not tie a signal that is connected to bidirectional pins. For example, with a Verilog netlist that has a top level pin named Vdd of type “inout”, you cannot use “add_tied_signals 1 Vdd -pin” to tie Vdd high.

Arguments

- **0**

  A literal that ties the floating nets or pins to logic 0 (low to ground).

- **1**

  A literal that ties the floating nets or pins to logic 1 (high to voltage source).

- **X**

  A literal that ties the floating nets or pins to unknown.

- **Z**

  A literal that ties the floating nets or pins to high-impedance.

- ***floating_object_name***

  A required, repeatable string that specifies the floating nets or pins to which you want to assign a specific value. The tool assigns the tied value to all floating nets or pins in all modules that have the names you specify.

  For pin names, you must include the -Pin switch on the command line; otherwise, the tool assumes the name is a net name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Pin

  An optional switch specifying that the *floating_object_name* argument that you provide is a floating pin name.

Examples

The following example ties all floating signals in the circuit that have the net names vcc and vdd, to logic 1 (tied to high):

```
add_tied_signals 1 vcc vdd
```

Related Topics

delete_tied_signals

report_tied_signals

set_tied_signals

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_to_collection

Context: unspecified, all contexts

Mode: all modes

Adds objects to a base collection and returns the new collection without affecting the original base collection.

Usage

add_to_collection ***base_collection object_spec*** [-unique]

Description

By default, objects that exist in both the ***base_collection*** and the ***object_spec*** are duplicated in the resulting collection. You can use the -unique switch to remove duplicates.

Arguments

- ***base_collection***

  A required value that specifies the base collection to which objects are to be added. This base collection is copied to a new collection, and objects matching object_spec are added to the new collection. The ***base_collection*** can be an empty collection (empty string).

- ***object_spec***

  A required value that specifies a Tcl list of one or more object names, or a collection of one or more objects to add to the ***base_collection***. The ***object_spec*** can be an empty collection (empty string). If ***object_spec*** is empty, the result is a copy of the base collection.

- -unique

  An optional switch that removes duplicate objects from the resulting collection. By default, duplicate objects are not removed.

Examples

Example 1

The following example adds two collections, a and b, to create a new collection called sum.

```
set a [get_ports a*]
set b [get_ports b*]
set sum [add_to_collection $a $b]
```

Example 2

The following example creates a collection of all nets connected to pins of design instances matching u1/myPrefix*. Note that using the -unique option automatically removes duplicate nets.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set sum “”
foreach_in_collection inst [get_instances u1/myPrefix* -of_type design] {
        set inputs [get_pins -of_instance $inst -direction input]
        set outputs [get_pins -of_instance $inst -direction output]
        set sum [add_to_collection $sum [get_fanins $inputs -stop_on net] -unique]
        set sum [add_to_collection $sum [get_fanouts $outputs -stop_on net] -unique]
}
```

Note

This example could have used the append_to_collection command instead, which would have been more efficient because in this example, the intermediate collections are destroyed as soon as they are added.

Related Topics

append_to_collection

compare_collections

copy_collection

filter_collection

foreach_in_collection

index_collection

is_collection

range_collection

remove_from_collection

sizeof_collection

sort_collection

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_wave_generator_pins

Context: all

Mode: all

Displays provided pin objects in the Tessent Visualizer Wave Generator tab.

Usage

add_wave_generator_pins ***object_spec***

Description

A flat model must be available in order to use this command. If Tessent Visualizer is not started, the command automatically starts both server and client and opens the Wave Generator tab.

Arguments

- ***object_spec***

  A Tcl list or Tcl collection of one or more names of pin, port, pseudo_port, or gate_pin objects.

Examples

add_wave_generator_pins {inst/clk inst/scan_en}

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_write_controls

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup

Specifies the off-state value for specified RAM write control lines.

Usage

add_write_controls {**0** | **1**} ***pin_pathname***...

Description

The add_write_controls command defines the circuit write control lines and assigns their off-state values. The off-state value of the pins or ports that you specify must be sufficient to keep the RAM outputs stable. The following are considerations for using this command:

- When the write control is already added as a clock, there is no need to also use add_write_controls for that port. This is the typical case, which means that for most usage it is unnecessary to explicitly use the add_write_controls command.
- If you want to have NR (non-clock) timing for write control lines that drive level-sensitive RAMs, you must add the write control using the add_write_control command; otherwise, the write control cannot pass DRC. NR timing for write control is not common.
- If a port drives the clock port of an edge-triggered RAM, the tool automatically infers that the port is a clock and adds it to the clock list. This is regardless of whether you added the port as a read control or not. For clocks that the tool infers, you must ensure that your timeplate includes the waveform for the newly defined clock.

Arguments

- **0 | 1**

  A required literal that specifies 0 or 1 as the off-state value for the RAM write control lines.

- ***pin_pathname***

  A required, repeatable string specifying the pins that are write control lines for the RAM and to which you want to assign an off-state value.

Examples

The following example assigns an off-state to two write control lines, w1 and w2:

```
add_write_controls 0 w1 w2
set_system_mode analysis
create_patterns
```

Related Topics

analyze_control_signals

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_write_controls

report_write_controls

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_atpg_constraints

Context: dft -edt, patterns -scan

Mode: analysis

Checks the ATPG constraints you created for their satisfiability or mutual exclusivity.

Usage

analyze_atpg_constraints {-AUto | -ALl| [{*pin_pathname* | gate_id# | *function_name*}... [-Satisfy | -Exclusive]]} [-Bus]

Description

If you issue the analyze_atpg_constraints command without any arguments, the default is -All. When the command finishes, the tool displays a message that indicates whether the analysis passed, failed, or aborted the ATPG constraint analysis.

Note

With EDT on, Tessent TestKompress support of this command is limited: it does not tell you if a constraint cannot be satisfied with EDT compression. It may also falsely report a constraint can be satisfied.

Arguments

The following lists the three methods for naming the objects for which you want to analyze the constraints. You can use any number of the three argument choices, in any order.

If you only specify an object name when you issue this command, by default, the tool performs the satisfiability (-Satisfy) analysis.

- -Auto

  An optional switch that automatically tries to locate the atpg constraint that cannot be satisfied. The analysis checks to see if any single constraint cannot be satisfied, then reports each constraint that cannot be satisfied (given the current abort limit and other restrictions). Sometimes, each constraint can be satisfied by itself, but some set of constraints cannot all be satisfied. In this case, -Auto switch proceeds to a second analysis where it adds atpg constraints to a set to create a minimal set that can't be satisfied.

- -ALl

  An optional switch that specifies for the tools to perform the ATPG analysis simultaneously for all the current ATPG constraints. If you do not specify an object name, this is the command default.

- *pin_pathname*

  A repeatable string that specifies the pathname to the pin on which you are analyzing the constraints.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- gate_id#

  A repeatable integer that specifies the gate identification number of the gate on which you want to analyze the constraints.

- *function_name*

  A repeatable string that specifies the name of a function you created with the add_atpg_functions command. If you generated the ATPG function with the -Cell option and added constraints with the -Cell option, then the tool also analyzes the constraints on all the cells affected by that ATPG function.

- -Satisfy

  An optional switch that specifies for the ATPG process to attempt to create a pattern that satisfies the selected ATPG constraint. This is the default if you specify an object name without a switch. During the ATPG process, the test generator does not consider the effect of other ATPG constraints or bus contention prevention (unless you use the -Bus switch).

  When the command finishes, the tools display a message that indicates whether the analysis passed (the ATPG process successfully generated a pattern), failed (the ATPG process could not find any possible pattern), or aborted (the ATPG process gave up on trying to find a successful pattern). If the analysis passes, the data that the tool simulated for the pattern is available in parallel pattern zero (0).

- -Exclusive

  An optional switch that specifies for the ATPG process to attempt to create a pattern that sets the selected ATPG constraint at a value different from its constrained value. This test's intent is to ensure that such a pattern does not exist. During the ATPG process, the test generator does not consider the effect of other ATPG constraints or bus contention prevention (unless you use the -Bus switch).

  When the command finishes, the tool displays a message that indicates whether the analysis passed (the ATPG process could not find any possible pattern), failed (the ATPG process found another possible pattern), or aborted (the ATPG process gave up on trying to find a successful pattern). If the analysis fails, the data that the tool simulated for the pattern is available in parallel pattern zero (0).

- -Bus

  An optional switch that specifies for the tool to consider bus contention prevention during the ATPG process.

Examples

The following example creates an ATPG constraint and then checks for mutual exclusivity:

add_atpg_constraints 1 435
analyze_atpg_constraints 435 -exclusive

```
// ATPG constraint 435=1 failed mutual exclusivity check (data
// in parallel pattern 0).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

add_atpg_constraints

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_bus

Context: dft -edt, patterns -scan

Mode: analysis

Analyzes the specified bus gates for contention problems.

Usage

analyze_bus {***gate_id#***... [-Exclusivity | -Prevention | -Zstate]} |
-Drc_check | -ALl | -Auto | -ANAlyze_sequentially

Description

If the bus passes the analysis, the tool displays a message indicating that it did so. If the analysis aborts, the tool displays a message that identifies the tri-state drivers (TSDs) the tool was analyzing at abort time. If the bus fails the analysis, the tool displays a message that identifies the two offending TSDs (the tri-state drivers capable of being on simultaneously, while driving different values).

Note

After using the analyze_bus command, the "set_gate_report parallel_pattern 0" command may display conflicting values if failures are found. That is, it is possible to have TIE-X gates with a 1/0 as an output. This is due to the analyze_bus command setting values to determine contention states. This analysis continues until a conflict is found.

In the tool, analyze_bus only shows the values that were used by the ATPG, and which the command considers relevant to the bus conflict. As the purpose of the analysis is to determine TSD input combinations that result in bus conflicts, the command does not simulate the output values of the TSD.

Note

With EDT on, Tessent TestKompress support of this command is limited: it does not tell you if a constraint cannot be satisfied with EDT compression. It may also falsely report a constraint can be satisfied.

The commands "set_contention_check On -Atpg" and "set_iddq_checks -Bus" both cause ATPG to ensure that every bus is contention free during deterministic test generation. Sometimes, this requirement cannot be met for many or all of the faults targeted by ATPG, preventing you from obtaining adequate fault coverage. When this happens, the analyze_bus command determines which bus or buses cannot be made contention free, so that you can investigate the circuit around this bus to find out what is preventing contention- free tests.

When you issue this command, you must either specify a *gate_id#* value or one of the global switches (-Drc_check or -All).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***gate_id#*** -Exclusivity | -Prevention | -Zstate

  A repeatable integer with an optional switch that specifies the identification number of the bus gate and the type of analysis you want the tool to perform.

  The available switch choices are as follows:

  -Exclusivity — An optional switch that specifies for the tool to analyze the bus gate to see if it has mutual exclusivity. Mutual exclusivity means that only one driver can simultaneously force a strong signal onto the bus. Exclusivity is the default behavior when you specify a *gate_id#* value without a corresponding switch.

  -Prevention — An optional switch that specifies for the tool to analyze the bus gate for its ability to attain a state of non-contention.

  -Zstate — An optional switch that specifies for the tool to analyze the bus gate for its ability to attain a high-impedance (Z) state.

- **-Drc_check**

  A switch that specifies for the tool to run the design rule check process again to categorize all buses and display the results. This is useful if you are changing constraints or the abort limit in an attempt to pass bus checks rather than abort.

- **-ALl**

  A switch that specifies for the tool to use the more extensive ATPG process to place all the fail and abort buses in a noncontentious state. The internal simulation data for this pattern is available in parallel pattern 0 of the gate report.

- **-AUto**

  A switch that automatically tries to locate the bus that cannot be made contention free. The analysis checks to see if any single bus cannot be made contention free. Each bus that cannot be made contention free (given the current abort limit and other restrictions) is reported to the user. Sometimes, each bus can be satisfied by itself, but some set of buses cannot all be satisfied. In this case, -Auto switch proceeds to a second analysis where it creates a minimized set of buses that can't be satisfied. The final, reduced set of buses, which cannot all be made contention free, is reported.

- **-ANAlyze_sequentially**

  A switch that specifies to perform a sequential analysis to find bus contention problems. For each clock defined using the add_clocks command, the sequential analysis applies a 1-cycle test, then a 2-cycle test, and so on, up to the reporting depths for which the tests fail. It then analyzes and lists the specific buses causing the failures at those depths.

  Note

  The same sequential analysis is performed automatically by the analyze_restrictions command if that command finds bus contention is the problem preventing success.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example analyzes a bus that failed the regular bus contention checking:

set_system_mode analysis
analyze_bus 493

Example 2

The following example displays the current categorization of bus gates, and then performs the prevention check on a specific bus gate:

set_system_mode analysis
analyze_bus -drc_check

```
// ATPG bus checking results: pass=1, bidi=1, fail=0, abort=0,
// CPU time=0.00.
```

analyze_bus 495 -prevention

```
// Controllability justification was successful (data accessible using
// parallel_pattern 0).
// Pattern type: Basic_scan
```

Example 3

The following example analyzes a bus that failed the regular bus contention checking. Notice the input values of gates 7 & 8 (the two TSDs) show that ATPG has correctly illustrated a conflict condition:

```
//  Warning: BUS gate /Y (9) has possible contention on drivers 7 and 8.
//  (E10-1)
```

analyze_bus 9

```
//  ATPG bus checking performed on /Y (9) for 2 bused TSDs.
//  Failure occurred while checking TSDs 7 and 8 (data in
//  parallel_pattern 0).
```

set_gate_report parallel 0
report_gates 9

```
//  /Y (9)  BUS
//      "I0"   I  (X)  7-/tri01/Y
//      "I1"   I  (X)  8-/tri02/Y
//      "OUT"  O  (X)  10-/Y
```

report_gates 7

```
//  tri01 (7)  TSD
//      E      I  (1)  3-/E
//      "D"    I  (1)  5-
//      Y      O  (X)  9-
```

report_gates 8

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  tri02 (8)  TSD
//      E      I  (1)  4-/E1
//      "D"    I  (0)  6-
//      Y      O  (X)  9-
```

Related Topics

analyze_restrictions

report_bus_data

set_contention_check

set_gate_level

set_gate_report

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_compression

Prerequisites: Scan-inserted netlist

Context: dft (with no subcontext), dft -edt, patterns (with no subcontext), patterns -scan

Mode: analysis

Analyzes the compression for an application in two steps.

Usage

Determine maximum possible compression (maximum chain:channel ratio)

```
analyze_compression [-Preserve_faults_from_analysis]
    [-NO_EDT] [-NO_COMPACTor]
    [-OVerride_user_settings | -NO_AUto]
    [-COMpaction_effort {Low | High | Maximum | integer}] [-NO_Analyze]
    [-COVerage_effort {Low | Medium | High}]
    [-NO_REDundancy_analysis]
    [-NO_TERMINATE_ineffective_atpg]
    [-PATterns_per_pass integer]
    [-EXEcute_after_atpg tcl_command]
```

Emulating a virtual single block EDT configuration

```
analyze_compression [-Preserve_faults_from_analysis]
    [-COMPACTOR_type Xpress | Basic]
    [-ENAble_edt_power_controller [-MIN_Switching_threshold_percentage int]]
    [-SEParate_control_data_channels]
    [-Uncompressed_chains chain_names ...]
    [-NO_Keep_uncompressed_chains]
    [-CHAINs N] {[-CHANNELs M] | [{-INPut_channels I} {-OUtput_channels O}]}
    [-NO_EDT] [-NO_COMPACTor]
    [-OVerride_user_settings | -NO_AUto]
    [-COMpaction_effort {Low | High | Maximum | integer}] [-NO_Analyze]
    [-COVerage_effort {Low | Medium | High}]
    [-NO_REDundancy_analysis]
    [-NO_TERMINATE_ineffective_atpg]
    [-PATterns_per_pass integer]
    [-EXEcute_after_atpg tcl_command]
```

Emulating a virtual modular EDT configuration

```
analyze_compression [-Preserve_faults_from_analysis]
    [-COMPACTOR_type Xpress | Basic]
    [-ENAble_edt_power_controller [-MIN_Switching_threshold_percentage int]]
    [-SEParate_control_data_channels]
    [-Uncompressed_chains chain_names ...]
    [-NO_Keep_uncompressed_chains]
    [-REPORT_EDT_Pins]
    [-NO_EDT] [-NO_COMPACTor]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

{-Edt_block *block1_name* [-CHAINs *int*] {[-CHANNELs *int*] |
[-INPut_channels *int* -OUtput_channels *int*]}}
[{-Edt_block *block2_name* [-CHAINs *int*] {[-CHANNELs *int*] |
[-INPut_channels *int* -OUtput_channels *int*]}} ...]
{[-Broadcast_all_channels_to_identical_blocks [{*block1_name block2_name* ...}]]} ...
{[-SHARE_data_channels [{*block3_name block4_name* ...}]
[-DATA_and_control_channels [*int*]]]} ...
[-OVerride_user_settings | -NO_AUto]
[-COMpaction_effort {Low | High | Maximum | *integer*}] [-NO_Analyze]
[-COVerage_effort {Low | Medium | High}]
[-NO_REDundancy_analysis]
[-NO_TERMINATE_ineffective_atpg]
[-PATterns_per_pass integer]
[-EXEcute_after_atpg *tcl_command*]

Description

Analyzes the compression for an application in two steps. The two steps are as follows:

1. Analyzes a scan-inserted design and returns a range of maximum chain:channel ratios where the test coverage begins to decline.
2. Emulates a hardware configuration for a specified chain:channel ratio, generates temporary test patterns, and returns test data statistics for the compression configuration.

Note

Before using the analyze_compression command, the target design should be setup for ATPG including the definition of any controls for internal clocks and the definition of all scan chains. In most cases, you can simply replace the create_patterns command in your dofile with the analyze_compression command and the switches you want. Detailed instructions on how to use the analyze_compression command to perform compression analysis are presented in the "Analyzing Compression" in the *Tessent TestKompress User's Manual*.

The analyze_compression command uses the fault model type specified by the set_fault_type command. If no fault model type is specified, the tool uses the "stuck" fault model by default. The analyze_compression command uses the current fault population. If no faults are added, the tool operates on all faults or a subset of sampled faults that are determined by the fault sampling rate specified by the "set_fault_sampling *rate*" command.

To enable N-detection during analysis, you must first issue the set_multiple_detection command. After enabling N-detection, you can run the analyze_compression command with the -Preserve_faults_from_analysis switch and the tool preserves the N-detection information for all the detected faults; if you do not specify the -Preserve_faults_from_analysis switch, the tool restores the N-detection information for all detected faults to the states prior to the execution of the analyze_compression command. For an example of enabling N-detection, see Example 16.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

During emulation, if the emulated EDT block has the same name as the original one, the tool preserves all the scan cells in the original EDT block; if you specify new EDT block names, the tool tries to allocate all scan cells into the new EDT blocks to get balanced scan chains. You cannot mix the names of the original EDT blocks with the names of the newly emulated EDT blocks. This is illustrated in Example 17.

For more information, see "Analyzing Compression" in the *Tessent TestKompress User's Manual*.

Note

When you perform compression analysis, the final order of scan cells in each scan chain in the design is not known. The order of the scan cells used as deterministic bits in the test cube impacts ATPG results. If many of the deterministic bits are clustered in consecutive cycles, more EDT_ABORT faults may appear because the clustered deterministic bits may not be compressed. When this occurs, ATPG test coverage may drop.

During compression analysis, TestKompress assumes the scan chains are organized in a balanced way based on an existing scan cell ordering. However, in the real design, scan cells may be in an arbitrary position depending on which scan stitching tool was used. If the clustering of deterministic bits occurs during compression analysis but not in the real design or if the clustering occurs in the real design but not in compression analysis, a discrepancy in the ATPG results may occur.

Arguments

- -Preserve_faults_from_analysis

  An optional switch that preserves the fault status at the end of the analyze_compression operation based on the APTG run of the command even though no patterns are stored. By default, without the -Preserve_faults_from_analysis switch, the fault status at the end of the analyze_compression operation is returned to its status prior to the execution of the command. This switch can be used to experiment with analyzing faults remaining after an analysis run.

  If the -Preserve_faults_from_analysis switch is specified, the tool preserves the N-detection information for all the detected faults during the analyze_compression operation.

  Note

  When you specify this option, if there are already internal patterns generated by a previous create_patterns command, the tool deletes all internal patterns and issues a warning stating that the previously generated patterns are deleted because they are inconsistent with the fault population.

- -OVerride_user_settings

  For information about this switch, see the create_patterns command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -NO_AUto

  For information about this switch, see the create_patterns command.

- -COMpaction_effort {Low | High | Maximum | *integer*}

  For information about this switch, see the create_patterns command.

- -NO_Analyze

  For information about this switch, see the create_patterns command.

- -COVerage_effort {Low | Medium | High}

  For information about this switch, see the create_patterns command.

- -NO_REDundancy_analysis

  For information about this switch, see the create_patterns command.

- -NO_TERMINATE_ineffective_atpg

  For information about this switch, see the create_patterns command.

- -PATterns_per_pass integer

  For information about this switch, see the create_patterns command.

- -COMPACTOR_type Xpress | Basic

  An optional switch that specifies the compactor to use for the analysis. By default, the Xpress compactor is used. The specified compactor is applied to all blocks in a block-level application. This switch cannot be specified more than once; otherwise, an error message is generated.

- -ENAble_edt_power_controller [-MIN_Switching_threshold_percentage *int*]

  An optional switch and switch-integer pair that enables emulation of the low-power controller and specifies the minimum switching threshold for the low-power controller being used for compression emulation. You must specify the -Channel option when using this option.

  If this switch is specified, the tool uses the specified low-power controller for all blocks. If no low-power usage is specified, then no low-power is used. This switch cannot be specified more than once; otherwise, an error message is generated.

  The behavior of the analyze_compression command, when used with this switch, can be effected by the switching threshold in several ways.

  - When using this switch, but the “set_power_control shift” command is not used, at the start of analyze_compression, the tool issues the following message:

    ```
    //  Note: Emulating low-power patterns with <N>% shift switching
    //  threshold.
    //  To use a different low power shift threshold or disable
    //  low power shift, use the 'set_power_control' command.
    ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- When using this switch, and meanwhile the "set_power_control shift off" command is used, the tool does not emulate low-power patterns, even if specified with analyze_compression, the tool issues this message:

```
//  Warning: Switching threshold for low power controller was
//  specified but is ignored because 'set_power_control shift' is
//  set to OFF.
//  Emulating patterns without low power control.
```

- When using "-enable_edt_power_controller -min_switching_threshold_percentage X", but meanwhile "set_power_control shift on Y is used, and X < Y, the tool uses the setting of the set_power_control command (Y% in this case) as opposed to the hardware setting with analyze_compression (X% in this case). No message is issued.
- If set_power_control is used, but switching threshold is not used with analyze_compression, the tool issues this message:

```
//  Note: Switching threshold was set using 'set_power_control
//  shift on -switching_threshold_percentage <int>' but is
//  ignored because the -min_switching_threshold_percentage was
//  not used with the analyze_compression command to specify the
//  configuration of the low power controller. Emulating patterns
//  without low power control.
```

- -SEParate_control_data_channels

  An optional switch that, if used, tells the tool to separate control and data channels for all EDT blocks during emulation, regardless of whether channel sharing is used. If this switch is not used, the tool does not separate control and data channels for all EDT blocks unless the block is in a channel sharing group specified with the -SHARE_data_channels switch.

- -Uncompressed_chains *chain_names* ...

  An optional switch and Tcl list of chain names that specifies uncompressed scan chains. You must define these chain names with the add_scan_chains command. If a chain name is not found, the tool issues an error on the first unfound chain name.

  If EDT is ON, then the following applies:

  - If you specify -Uncompressed_chains without specifying the optional -NO_Keep_uncompressed_chains, then the tool treats the specified uncompressed chains and the original uncompressed chains as uncompressed chains.
  - If you specify -Uncompressed_chains with the optional -NO_Keep_uncompressed_chains, then the tool only treats the specified uncompressed chains as uncompressed chains.

  If the current design has no EDT or EDT is OFF, then the following applies:

  - If you specify -Uncompressed_chains, then the tool only treats the specified chains as uncompressed chains while using the scan cells in the remaining uncompressed chains for compression analysis.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -NO_Keep_uncompressed_chains (**EDT ON only**)

  An optional literal that instructs the tool not to treat any original uncompressed chains in the design as uncompressed unless they are specified in the -Uncompressed_chains.

  By default if it is an EDT design, the tool keeps any uncompressed scan chains as uncompressed and does not merge those cells with the compressed chains. If this switch is used on a design that does not contain any uncompressed chains, then the tool issues a warning.

  If the current design has no EDT or EDT is OFF, then this switch is ignored.

- -REPORT_EDT_Pins

  An optional literal that instructs the tool to call the "report_edt_pins -all -group_by_pin_name" command at the end of the compression analysis operation. The tool reports example pin names, and provides one example on how top-level pins are connected with the EDT block channels so that channel sharing/broadcasting can be emulated.

- -NO_EDT

  An optional switch that disables EDT when running compression analysis. This can help you determine the baseline ATPG pattern count. You can use this in conjunction with the results of the -no_compactor switch to explore why EDT pattern count is higher than pattern count without EDT. This switch is not compatible with any switch that is related to EDT (such as -compactor_type or -edt_block). To emulate a design with the specified number of chains, use "analyze_compression -chains <*N*> -no_edt".

- -NO_COMPACTor

  An optional switch that disables compactor emulation when running compression analysis. If the EDT pattern count is higher than the pattern count without EDT (generated with the -no_edt switch), it could be due to X values going into the compactor, encoding capacity, or both. Turning off compactor emulation enables you to evaluate the impact of X values going into the compactor. You can check the encoding capacity by specifying more input channels. This switch is incompatible with the -compactor_type and "-output_channels <*O*>" arguments; however, you can supply "-input_channels <*I*>" with -no_compactor.

- {-Edt_block *block1_name*} ...

  An optional, repeatable, switch and string pair that specifies the name of a design block containing compression logic (compression block). This option specifies multiple compression blocks for the analysis of a block-level architecture design.

  If you only specify one compression block, the design is treated as a chip-level architecture for analysis.

  Chain or channel specifications must be defined for each block individually.

  If you do not specify any compression blocks for a block-level design, the design is analyzed with the configurations defined in the design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Only the associated scan cells are used to analyze existing compression blocks. If you do not specify any scan channels for existing blocks, the tool uses those specified in the design.

If you specify any new compression blocks, the tool completely ignores the scan chain and channel configurations defined in the design. For the analysis, the tool distributes the scan cells across all specified blocks in a way that balances the scan chain length for all blocks. In this case, you must specify both chains and channels for each block. You cannot define both new and existing compression blocks in one design.

- -CHAINs *int*

  An optional switch and integer pair that specifies how many scan chains to use when emulating a chain:channel ratio for test data volume and fault coverage analysis. If no scan chains are specified, the scan chains defined in the design are used.

  By default, using -chains *int* leads to reordering scan chains and scan cells to make the scan chains as balanced as possible. There is a way to skip the reordering.

  - When you emulate a virtual single block EDT configuration:

    If you do not use -chains *int*, the tool maintains the order of the scan chains and scan cells. For example the original design has 100 chains.

    analyze_compression -channel 2

    The tool emulates one EDT block with 2 channels and 100 chains without reordering the scan chains and scan cells.

  - When you emulate a virtual modular EDT configuration:

    If you do not use -chains *int* and use the original EDT block names, the tool maintains the order of the scan chains and scan cells. For example there are two blocks in the original design, blk1 (2 channels, 100 chains) and blk2 (2 channels, 200 chains).

    analyze_compression -edt_block blk1 -channel 3 -edt_blk blk2 -channel 4

    The tool emulates blk1 with 3 channels, 100 chains and blk2 with 4 channels, 200 chains, without reordering the scan chains and scan cells in each block.

- [-CHANNELs *int*] | [{-INPut_channels *int*} {-OUtput_channels int}]

  An optional switch and integer pair or set of switch and integer pairs that specifies the channels to use when emulating a chain:channel ratio for test data volume and fault coverage analysis. Channels must be specified when analyzing a design without compression logic inserted.

  If no channels are specified for a design with compression logic inserted, the following scenarios apply:

  - **Block-Level Architecture** — The sum of the channels defined for each block is used, and the design is emulated as a single compression block.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **Chip-Level Architecture** — The channels defined by the existing compression logic are used.

Use the -input_channels/-output channels arguments to specify an asymmetrical channel configuration. These arguments must be used in pairs, except when you specify -no_compactor, in which case you can specify -input_channels without -output_channels. You can also optionally use -no_compactor with -channels.

- {[-Broadcast_all_channels_to_identical_blocks [{*block1_name block2_name* ...}]]} ...

  An optional argument used to construct one or more groups of identical blocks that share input channels. The specified blocks must be pre-existing identical EDT blocks. The tool broadcasts the input channels to all EDT blocks that are in one specified identical block group.

  If this argument is specified but no block_name parameters are specified, the tool assumes that all existing EDT blocks are identical; in this case, only one channel broadcasting group is permitted.

- {[-SHARE_data_channels [{*block3_name block4_name* ...}] [-DATA_and_control_channels [*int*]]]} ...

  An optional argument used to construct one or more channel sharing groups such that the EDT blocks in each channel sharing group share data input channels with each block having individual control channels such that the total number of input channels used in this group equals the input channel number specified by -DATA_and_control_channels. The tool calculates the minimum number of input channels based on the channel sharing groups you define with this argument.

  The block_name parameters are the names of the blocks defined to be in the channel sharing group.

  The following rules apply to the creation of channel sharing groups:

  - If no block_name parameters are specified, the design must already have two or more EDT blocks; otherwise, the tool generates an error.
  - If no block_name parameters are specified, all blocks share the data channels; in this case, only one channel sharing group is permitted.
  - If block_name parameters are specified, each channel sharing group must have a minimum of two blocks defined; otherwise, the tool generates an error.
  - An EDT block can only be defined in one channel sharing group or one channel broadcasting group. Note, an EDT block does not have to be defined in any channel sharing or channel broadcasting group.
  - If a block is defined in a channel sharing group, the "set_edt_options -separate_control_data on" switch is automatically set. In this case, the tool calculates the number of control channels that are needed for the block by determining the number of control bits and the length of data channels. The total input channel count for each block, specified by either the analyze_compression

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

"-channels" or "-input_channels" switches, must be greater than or equal to the number of control channels plus 1. This means the block must have at least one data channel.

- The specified input pin count (N) for each channel sharing group must be greater than or equal to the sum of the #control channels on each block in the group plus 1.That is, at least one pin must be shared by the data channels of the EDT blocks in one channel sharing group.
- The number of input data pins available to be shared in the group cannot be greater than or equal to the sum of all data channels of all blocks in the group.
- The number of input data pins available to be shared in this group must be greater than or equal to the maximum number of data input channels of all EDT blocks in the group.

Note

The analyze_compression command supports channel sharing and channel broadcasting separately. You can use it to emulate a group of cores with channel sharing and a different group of identical cores with broadcasting, but there can not be any overlap of the cores between the groups.

As a workaround, use a representative core instance from each identical core group and perform channel sharing emulation. The results should be close to adding the other equivalent cores if each core is well isolated.

- [-EXEcute_after_atpg *tcl_command*]

  An optional switch and string pair that runs a Tcl command after the temporary test pattern generation phase of analyze_compression. You can use this switch to analyze the temporary test patterns. The <*tcl_command*> can be a permitted tool command or a Tcl proc. Not all tool commands are safe to execute in this temporary system state. Examples of permitted commands include compress_patterns and order_patterns.

Examples

Example 1

The following example analyzes a design and returns the upper range of chain:channel ratios:

analyze_compression

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// For stuck-at faults
//
// Chain:Channel Ratio     Predicted Fault Coverage Drop
//  -------------------    ------------------------------
//  153                    negligible fault coverage drop
//  154                    0.01 % - 0.05 % drop
//  160                    0.10 %
//  168                    0.15 %
//  171                    0.20 %
…
//  CPU time is 155 seconds.
```

Example 2

The following example emulates compression logic with 800 scan chains and 8 scan channels, generates test patterns, and displays test pattern statistics for a chip-level architecture:

set_fault_sampling 1
analyze_compression -chains 800 -channels 8

```
             Statistics Report
              Stuck-at Faults
-------------------------------------------
Fault Classes                     #faults
                                  (total)
---------------------------  --------------
  FU (full)                       59164
  -------------------------  --------------
  UO (unobserved)               167 ( 0.28%)
  DS (det_simulation)         47815 (80.82%)
  DI (det_implication)        10425 (17.62%)
  UU (unused)                   123 ( 0.21%)
  TI (tied)                     219 ( 0.37%)
  BL (blocked)                   50 ( 0.08%)
  RE (redundant)                142 ( 0.24%)
  AU (atpg_untestable)          223 ( 0.38%)
-------------------------------------------
Fault Sub-classes
  -------------------------
  AU (atpg_untestable)
    SEQ  (sequential_depth)      20 ( 0.03%)
    Unclassified                203 ( 0.34%)
  UC+UO
    AAB  (atpg_abort)             8 ( 0.01%)
    UNS  (unsuccess)            159 ( 0.27%)
-------------------------------------------
Coverage
  -------------------------
  test_coverage                      99.33%
  fault_coverage                     98.44%
  atpg_effectiveness                 99.72%
-------------------------------------------
#test_patterns                          348
#simulated_patterns                     416
CPU_time (secs)                       250.6
-------------------------------------------

Note: The reported statistics are based on a 1% fault sample.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  CPU time to analyze compression is 213 seconds.
//
//  ----------------------------------------------------
//  Scan volume report.
//  -------------------
//     channels     : 8
//     shift cycles : 164+4
//  ----------------------------------------------------
//  pattern       # test  # scan                  volume
//  type        patterns   loads  (cell loads or unloads)
//  ----------  --------  ------  -----------------------
//  chain_test       130     130                  174720
//  basic            348     348                  467712
//  ----------  --------  ------  -----------------------
//  total            478     478                  642432   (642.4K)
//
```

Example 3

The following example emulates the specified compression logic, generates test patterns, and displays test pattern statistics for a block-level architecture:

```
set_fault_sampling 5
analyze_compression -edt_block edt_bk1 -chains 400 -channels 2 \
                    -edt_block edt_bk2 -chains 2000 -channels 10
```

```
Statistics report
---------------------------------------------
                          #faults   #faults
fault class               (coll.)   (total)
-----------------------   -------   -------
FU (full)                  21712     31930
-----------------------   -------   -------
UO (unobserved                 14        26
DS (det_simulation          17203     27194
DI (det_implication)         4239      4374
PU (posdet_untestable)          6         7
UU (unused)                    29        51
TI (tied)                      22        22
RE (redundant)                112       116
AU (atpg_untestable)           87       140
-----------------------   -------   -------
test_coverage              99.52%   99.47%
fault_coverage             98.77%   98.88%
atpg_effectiveness         99.94%   99.92%
---------------------------------------------
#test_patterns                          700
#simulated_patterns                     704
Note: The reported statistics are based on a 5% fault sample.

CPU time to analyze compression is 70 seconds.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  ------------------------------------------------------------
//  Scan volume report.
//  -------------------
//     channels     : 1
//     shift cycles : 487+49
//  ------------------------------------------------------------
//  pattern             # test  # scan                  volume
//  type              patterns   loads  (cell loads or unloads)
//  ----------------  --------  ------  -----------------------
//  chain_test             265     265                   142040
//  basic                 6426    6426                  3444336
//  clock_po                10      10                     5360
//  clock_sequential      1763    1763                   944968
//  ----------------  --------  ------  -----------------------
//  total                 8464    8464                  4536704   (4.5M)
//
```

Example 4

In the following example, a fault model type is not specified. By default, the stuck-at fault model type is used.

analyze_compression -chain 100 -channel 2

```
// ************************************
// * Run ATPG for stuck-at faults.
// ************************************
```

Example 5

In the following example, the -Preserve_faults_from_analysis switch is used to update the fault detection status based on the ATPG results of the analyze_compression command.

analyze_compression -chain 100 -channel 2 -preserve_faults_from_analysis

```
// ************************************
// * Run ATPG for stuck-at faults.
// ************************************

Note: Fault detection status will be updated based on the ATPG results of
analyze_compression even though no patterns are stored.
```

Example 6

In the following example, the set_fault_type command is used to specify the fault model type to be used by the analyze_compression command.

set_fault_type transition
analyze_compression -chain 100 -channel 2

```
// ************************************
// * Run ATPG for transition faults.
// ************************************
```

Example 7

In the following example, the set_fault_type command is used to specify that a user-defined fault model type is to be used by the analyze_compression command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_fault _type UDFM
read_fault_sites ../fault/sampled.udfm
analyze_compression -chain 100 -channel 2

// ************************************
// * Run ATPG for UDFM faults.
// ************************************
```

Example 8

In the following example, the Xpress compactor type is specified for a single EDT block.

```
analyze_compression -compactor_type xpress -chain 100 -channel 2
```

Example 9

In the following example, both EDT blocks are emulated with the Xpress compactor.

```
analyze_compression -compactor_type xpress \
                    -edt_block blk1 -chain 100 -channel 2 \
                    -edt_block blk2 -chain 200 -channel 2
```

Example 10

In the following example, both EDT blocks are emulated with the Basic compactor.

```
analyze_compression \
                    -edt_block blk1 -compactor_type basic -chain 100 -channel 2 \
                    -edt_block blk2 -chain 200 -channel 2
```

Example 11

In the following example, only one EDT block is emulated with the low-power controller.

```
analyze_compression -enable_edt_power_controller \
                    -min_switching_threshold_percentage 20 \
                    -chain 100 -channel 2
```

Example 12

In the following example, the low-power controller is used for both EDT blocks. The switching_threshold_percentage switch is set to 20 percent.

```
analyze_compression -enable_edt_power_controller \
                    -min_switching_threshold_percentage 20 \
                    -edt_block blk1 -chain 100 -channel 2 \
                    -edt_block blk2 -chain 200 -channel 2
```

Example 13

In the following example, both EDT blocks are emulated with the same low-power controller.

```
analyze_compression -edt_block blk1 \
                    -enable_edt_power_controller \
                    -min_switching_threshold_percentage 20 -chain 100 -channel 2 \
                    -edt_block blk2 -chain 200 -channel 2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 14

In the following example, only one EDT block is emulated with the low-power controller. The switching_threshold_percentage switch is set to 15 percent by default.

```
analyze_compression -enable_edt_power_controller -chain 100 -channel 2
```

Example 15

In the following example, both EDT blocks are emulated with the low-power controller. The switching_threshold_percentage switch is set to 15 percent by default.

```
analyze_compression -enable_edt_power_controller \
                    -edt_block blk1 -chain 100 -channel 2 \
                    -edt_block blk2 -chain 200 -channel 2
```

Example 16

The following example enables N-detection ATPG during the analyze_compression.

```
set_multiple_detection -guaranteed_atpg_detections 3 -desired_atpg_detections 1
set_fault_sampling 2
analyze_compression -chain 100 -channel 2
```

```
//  ************************************
//  *   Run ATPG for stuck-at faults.
//  ************************************
......

  ----------------------
  test_coverage                                      78.79%
  fault_coverage                                     76.37%
  atpg_effectiveness                                 79.44%
------------------------------------------------------------
#test_patterns                                          617
#simulated_patterns                                    1280
CPU_time (secs)                                        61.0
------------------------------------------------------------
Multiple Detection Statistics
------------------------------------------------------------
  Detections                      DS Faults    Test Coverage
     (N)                   (Detection == N) (Detection >= N)
------------------------  ---------------- ----------------
      1                        4 (  0.29%)   1070 ( 78.79%)
      2                        9 (  0.64%)   1066 ( 78.50%)
      3                      206 ( 14.70%)   1057 ( 77.84%)
      4                       27 (  1.93%)    851 ( 62.67%)
      5+                     727 ( 51.89%)    824 ( 60.68%)
------------------------------------------------------------
bridge_coverage_estimate                           74.79%
------------------------------------------------------------

Note: The reported statistics are based on a 2% fault sample.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  CPU time to analyze compression is 30 seconds.
//
//  ----------------------------------------------------
//  Scan volume report.
//  -------------------
//     channels     : 4
//     shift cycles : 41+19
//  ----------------------------------------------------
//  pattern       # test  # scan                  volume
//  type        patterns   loads  (cell loads or unloads)
//  ----------  --------  ------  -----------------------
//  chain_test        36     136                   32640
//  basic            617     617                  148080
//  ----------  --------  ------  -----------------------
//  total            753     753                  180720   180.7K)
```

Example 17

The following examples demonstrate how the tool enforces the naming requirements for the emulation step.

Case 1

The original design has two EDT blocks named block1 and block2.

```
analyze_compression -edt_block block1 -chain10 -channel 1 \
                    -edt_block block2 -chain10 -channel 1
```

The tool maintains all cells in the original block1 and block2 because they have the same name as the original, but changes the number of scan chains and channels based on the command.

Case 2

The original design has two EDT blocks named block1 and block2.

```
analyze_compression -edt_block block3 -chain10 -channel 1 \
                    -edt_block block4 -chain10 -channel 1
```

The tool does not maintain all of the cells in the original blocks because the names changed. The tool reallocates cells to make all of the chains in the two blocks as balanced as possible.

Case 3

The original design has two EDT blocks named block1 and block2.

```
analyze_compression -edt_block block1 -chain10 -channel 1 \
                    -edt_block block4 -chain10 -channel 1
```

The tool does not permit the mixing of the original names and the newly specified names and generates an error message.

Case 4

The original design has two EDT blocks named block1 and block2.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
analyze_compression -edt_block block1 -chain10 -channel 1
```

The tool does not permit this case because it changes two blocks to one block, but still uses the same name as before.

Case 5

The original design has two EDT blocks named block1 and block2.

```
analyze_compression -edt_block block3 -chain10 -channel 1
```

The tool supports this case.

Case 6

The original design has one EDT block named block1.

```
analyze_compression -edt_block block1 -chain10 -channel 1 \
                    -edt_block block2 -chain10 -channel 1
```

The tool does not permit this and displays an error message.

Case 7

The original design has one EDT block named block1.

```
analyze_compression -edt_block block2 -chain10 -channel 1 \
                    -edt_block block3 -chain10 -channel 1
```

The tool supports this case.

Example 18

The following examples demonstrate how the tool supports channel sharing.

Case 1:

The design has one sharing group with 12 input channels shared between all blocks (block1, block2, block3 and block4).

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 4 \
                    -edt_block block3 -chain 300 -channel 6 \
                    -edt_block block4 -chain 200 -channel 2 \
                    -share_data_channels -data_and_control_channels 12
```

Case 2:

The design has two sharing groups. The first group has 6 input channels shared between block1 and block2. The second group has 8 input channels shared between block3 and block4.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 4 \
                    -edt_block block3 -chain 300 -channel 6 \
                    -edt_block block4 -chain 200 -channel 2 \
                    -share_data_channels block1 block2 -data_and_control_channels 6 \
                    -share_data_channels block3 block4 -data_and_control_channels 8
```

Case 3:

The design has one sharing group with 6 input channels shared between block1 and block2, but block3 does not share any inputs with other blocks.

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 4 \
                    -edt_block block3 -chain 300 -channel 6 \
                    -share_data_channels block1 block2 -data_and_control_channels 6
```

Case 4:

In this example, a block is included in multiple channel sharing groups:

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 4 \
                    -edt_block block3 -chain 300 \
                    -channel 6 -share_data_channels block1 block2 \
                    -data_and_control_channels 6 \
                    -share_data_channels block1 block3 -data_and_control_channels 8
```

The following error is generated:

```
// Error: Block 'block1' is included in multiple channel sharing groups.
```

Case 5:

In this example, a block is added to a channel sharing group but the block does not share input channels with any other blocks in that group.

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 4 \
                    -edt_block block3 -chain 300 -channel 6 \
                    -share_data_channels block1 -data_and_control_channels 6 \
                    -share_data_channels block2 block3 -data_and_control_channels 8
```

The following error is generated:

```
// Error: Block 'block1' is specified inside a channel sharing group, but
// does not share input channels with any other block.
```

Case 6:

In this example, a block is specified inside more than one channel sharing group.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 4 \
                    -edt_block block3 -chain 300 -channel 6 \
                    -share_data_channels block1 block1 -data_and_control_channels 6 \
                    -share_data_channels block2 block3 -data_and_control_channels 8
```

The following error is generated:

```
// Error: Block 'block1' is specified multiple times inside a channel
// sharing group.
```

Case 7:

This example assumes that each block needs two control channels.

```
analyze_compression -edt_block block1 -chain 100 -channel 3 \
                    -edt_block block2 -chain 200 -channel 4 \
                    -edt_block block3 -chain 300 -channel 4 \
                    -share_data_channels -data_and_control_channels 6
```

The following error is generated:

```
// Error: Channel sharing group with 'block1' (2 control channel),
// 'block2' (2 control channel), and 'block3' (2 control channels) needs
// at least 7 input channels for channel sharing.
```

Case 8:

This example assumes that block1 has one control and one data channel; block2 has one control channel and two data channels; and block3 has two control channels and two data channels. Because the command specifies to share nine channels, no channels are actually shared.

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 3 \
                    -edt_block block3 -chain 300 -channel 4 \
                    -share_data_channels -data_and_control_channels 9
```

The following error is generated:

```
// Error: Excessive input channels specified for channel sharing group
// which includes 'block1'. This group requires between 6 and 8 input
// channels.
```

Case 9:

This example assumes that block1 has one control and one data channel; block2 has one control channel and two data channels; and block3 has two control channels and two data channels. The command specifies to share five channels which is not enough.

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 3 \
                    -edt_block block3 -chain 300 -channel 4 \
                    -share_data_channels -data_and_control_channels 5
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following error is generated:

```
// Error: The available input channels (5) specified for the channel
// sharing group is not enough to drive the data input channels for block
// 'block2' and 'block3'. This group requires between 6 and 8 input
// channels. Alternatively, if you want to keep the shared channel count
// (5), you may reduce the number of input channels for the following
// blocks:
// 'block1' may have up to 2 channels.
// 'block2' may have up to 2 channels.
// 'block3' may have up to 3 channels.
```

Case 10:

In this example, channel sharing is specified but neither the total number of input pins nor any channel sharing groups are defined. In this case, the tool defines the channel sharing group to be all EDT blocks and calculates the required number of input channels for those three blocks.

```
analyze_compression -share_data_channels
```

The following message is generated:

```
// Sharing 8 input channels across block1 block2 block3.
```

Case 11:

In this example, channel sharing is specified and two channel sharing groups are defined. However, the total number of input pins is not specified. In this case, the tool calculates the required number of input channels for each of the two defined channel sharing groups.

```
analyze_compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 4 \
                    -edt_block block3 -chain 300 -channel 6 \
                    -edt_block block4 -chain 200 -channel 2 \
                    -share_data_channels block1 block2 \
                    -share_data_channels block3 block4
```

```
// Sharing 8 input channels across block1, block2.
// Sharing 10 input channels across block3, block4.
```

Example 19

The following examples demonstrate how the tool supports broadcasting channel input to blocks defined to be identical.

Case 1:

In this example, the original design is assumed to have three identical EDT blocks and each block has eight EDT input channels. Because no block names are listed, the tools assumes all EDT blocks are identical and broadcasts to all three blocks.

```
analyze_compression -broadcast_all_channels_to_identical_blocks
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Broadcasting 8 input channels to block1, block2, block3.
```

Case 2:

In this example, the tool broadcasts to two specified identical blocks and shares channels between two specified non-identical blocks.

```
analyze_compression -edt_block block1 -chain 100 -channel 4 \
                    -edt_block block2 -chain 100 -channel 4 \
                    -edt_block block3 -chain 300 -channel 6 \
                    -edt_block block4 -chain 200 -channel 2 \
                    -broadcast_all_channels_to_identical_blocks block1 block2\
                    -share_data_channels block3 block4 -data_and_control_channels 10
```

```
// Broadcasting 4 input channels to block1, block2.
// Sharing 10 input channels across block3, block4.
```

Case 3:

In this example, the tool broadcasts channel inputs to two different groups of identical blocks.

```
analyze_compression -edt_block block1 -chain 100 -channel 4 \
                    -edt_block block2 -chain 100 -channel 4 \
                    -edt_block block3 -chain 300 -channel 6 \
                    -edt_block block4 -chain 200 -channel 6 \
                    -broadcast_all_channels_to_identical_blocks block1 block2 \
                    -broadcast_all_channels_to_identical_blocks block3 block4
```

```
// Broadcasting 4 input channels to block1, block2.
// Broadcasting 6 input channels to block3, block4.
```

Case 4:

In this example, the tool attempts to broadcast channel inputs to two identical blocks; an error is generated because blocks that were specified to be identical are not identical because they have a different number of input channels.

```
analyze_compression -edt_block block1 -chain 100 -input_channels 3 -output_channels 2 \
                    -edt_block block2 -chain 100 -input_channels 4 -output_channels 2 \
                    -broadcast_all_channels_to_identical_blocks block1 block2
```

```
// Error: EDT block 'block1' and 'block2' are non-identical due to
//        different number of input channels.
//        '-broadcast_all_channels_to_identical_blocks' is not allowed.
```

Case 5:

In this example, the tool attempts to broadcast channel inputs to two identical blocks; an error is generated because blocks that were specified to be identical are not identical because they have a different number of output channels.

```
analyze_compression -edt_block block1 -chain 100 -input_channels 3 -output_channels 2 \
                    -edt_block block2 -chain 100 -input_channels 3 -output_channels 3 \
                    -broadcast_all_channels_to_identical_blocks block1 block2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Error: EDT block 'block1' and 'block2' are non-identical due to
//        different number of output channels.
//        '-broadcast_all_channels_to_identical_blocks' is not allowed.
```

Case 6:

In this example, the tool attempts to broadcast channel inputs to two identical blocks; an error is generated because blocks that were specified to be identical are not identical because they have a different number of scan chains.

```
analyze_compression -edt_block block1 -chain 100 -input_channels 3 -output_channels 2 \
                    -edt_block block2 -chain 120 -input_channels 3 -output_channels 2 \
                    -broadcast_all_channels_to_identical_blocks block1 block2
```

```
// Error: EDT block 'block1' and 'block2' are non-identical due to
//        different number of scan chains.
//        '-broadcast_all_channels_to_identical_blocks' is not allowed.
```

Case 7:

In this example, Chain1 in block1 has 400 scan cells and Chain1 in block2 has 500 scan cells. The tool attempts to broadcast channel inputs to these two identical blocks. An error is generated because blocks that were specified to be identical are not identical because Chain1 in block1 has a different number of scan cells than Chain1 in block2.

```
analyze_compression -edt_block block1 -chain 100 -input_channels 3 -output_channels 2 \
                    -edt_block block2 -chain 100 -input_channels 3 -output_channels 2 \
                    -broadcast_all_channels_to_identical_blocks block1 block2
```

```
// Error: EDT block 'block1' and 'block2' are non-identical due to
//        different number of scan cells on chain 'Chain1'.
//        '-broadcast_all_channels_to_identical_blocks' is not allowed.
```

Case 8:

In this example, the tool attempts to broadcast channel inputs to two different groups of identical blocks; an error is generated because one of the blocks is defined as a member of two different groups of identical blocks.

```
analyze_compression -edt_block block1 -chain 100 -channel 4 \
                    -edt_block block2 -chain 100 -channel 4 \
                    -edt_block block3 -chain 100 -channel 4 \
                    -broadcast_all_channels_to_identical_blocks block1 block2 \
                    -broadcast_all_channels_to_identical_blocks block2 block3
```

```
// Error: EDT block 'block2' show up in two groups."
```

Case 9:

In this example, the tool attempts to broadcast channel inputs to a block that is specified in both a group of identical blocks and a group of non-identical blocks.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
analyze_compression -edt_block block1 -chain 100 -channel 4 \
                    -edt_block block2 -chain 100 -channel 4 \
                    -edt_block block3 -chain 200 -channel 5 \
                    -broadcast_all_channels_to_identical_blocks block1 block2 \
                    -share_data_channels block2 block3 -data_and_control_channels 6
// Error: EDT block 'block2' show up in two groups."
```

Example 20

In this example, there are 100 chains, named “chain1, chain2, …, chain100” and the first four chains are set to uncompressed chains:

```
analyze_compression -uncompressed_chains chain1 chain2 chain3 chain4 -channels 4 -chain 50
```

Issuing this command specifies that chain1 to chain4 are uncompressed chains, while the scan cells in the remaining 96 chains are evenly distributed among 50 chains. These 50 chains are compressed with 4 EDT channels.

Example 21

In this example, there are 100 chains in an EDT design, and the first 4 chains are already uncompressed chains:

```
analyze_compression -edt_block blk1 -channels 4 -chain 50 -edt_block blk2 -channel 4 -chain 50
```

Issuing this command maintains the first 4 chains as uncompressed chains, while the scan cells in the remaining 96 chains are evenly distributed among 100 chains (50 chains in each of the two EDT block).

Example 22

In this example, there are 100 chains (chain1 through chain100) in an EDT design, and the first 4 chains are uncompressed chains. To add two more uncompressed chains (chain5 and chain6) during the emulation, you would issue the following command:

```
analyze_compression -uncompressed_chains chain5 chain6  -channels 4 -chain 50
```

This treats the first 6 chains as uncompressed chains, while the scan cells in the remaining 94 chains are evenly distributed among 50 chains, and these 50 chains are compressed with EDT of 4 channels.

Example 23

In this example, there are 100 chains (chain1 through chain100) in an EDT design, and the first 4 chains are uncompressed chains. To replace these uncompressed chains with new uncompressed chains (chain5 and chain6) during the emulation, you would issue the following command:

```
analyze_compression -no_keep_uncompressed_chains \
                    -uncompressed_chains chain5 chain6  -channels 4 -chain 50
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This treats chain5 and chain6 as uncompressed chains, while the scan cells in the remaining 98 chains are evenly distributed among 50 chains, and these 50 chains are compressed with EDT of 4 channels.

Example 24

The following example emulates compression logic with two channels. You can improve the accuracy of the basic coverage estimate by compressing the patterns and performing fault simulation on the undetected faults after temporary pattern generation with the following command:

```
analyze_compression -channels 2 -execute_after_atpg {compress_patterns -reset_au}
```

Related Topics

set_edt_instances

set_edt_options

set_fault_sampling

set_fault_type

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_control_signals

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Identifies and optionally defines the primary inputs of control signals.

Usage

analyze_control_signals [-Report_only | -Auto_fix] [-Verbose]

Description

The analyze_control_signals command identifies each control signal (clocks, set, reset, write-control, read-control, and so on) of every sequential element (for example, DFF, latch, RAM, ROM) and optionally defines its primary input as a control signal. The purpose of analyzing the control signals is to identify the primary inputs that need to be defined as a clock, read-control, or write-control. This analysis also considers pin constraints, but only traces through simple combinational gates.

If the -Verbose option is specified, the tool issues messages that indicate why certain control signals are not reported. At the end of the analysis, the tool displays statistical information such as the number of primary inputs identified as control signals, their types, and additional information.

If the -Auto_fix option is specified, all identified primary inputs of control signals are automatically defined. For example, when a clock is identified, an implicit add_clocks command is performed to define the primary input. By default, all control signals are only reported.

This command performs the flattening process automatically, if run prior to performing flattening.

Note

The tool defers to your expert knowledge when you use add_clocks to define a clock; the -Auto_fix option does not alter a clock definition specified by a preceding add_clocks command.

dft -scan Details

Because it is not possible to completely identify all control signals, you should only use the results of this command as a starting point in creating a dofile that defines the clock control signals, read and write controls in a design. You should use this dofile when the tool is being used for production scan and test logic insertion. Situations where the tool may not be able to identify particular control signals include the following:

- When the tool is unable to trace or simulate through complex logic, the tool cannot identify a top-level pin as a clock. These cases include clocks driven by PLLs or clock

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

gaters that require simulating a test setup procedure to obtain a sensitizable path to a top-level pin.

- The tool cannot accurately predict the off-state of a clock by observing the flip-flops it is driving and, therefore, requires the user to verify that the off-state of the clock/set/reset lines is correct.

Because of these types of issues, this command does not identify all clocks, their off-states, or controls signals. In this case, you should explicitly add the unidentified clocks using the add_clocks command.

Arguments

- -Report_only

  An optional literal that specifies to identify control signals only (does not define the primary inputs as control signals). This is the invocation default.

- -Auto_fix

  An optional literal that specifies to define the primary inputs of all identified control signals as control signals. For example, when a clock is identified, an implicit add_clocks command is performed to define the primary input.

- -Verbose

  An optional literal that specifies to display information on control signals (whether they are identified or not, and why) while the analysis is performed.

Examples

The following example analyzes the control signals, then only provides a verbose report on the control signals in the design. After examining the transcript, you can then perform another analysis of the control signals to add them.

analyze_control_signals -verbose

```
//  command: analyze_control_signals -reports_only -verbose
-----------------------------------------------------------------------
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Begin control signals identification analysis.
-----------------------------------------------------------------------
//  Warning: Clock line of '/cc01/tim_cc1/add1/post_latch_29/WRITEB_reg/r/
// (7352)' is uncontrolled at '/IT12 (4)'.
...
...
...
//  Identified 2 clock control primary inputs.
//     /IT23 (5) with off-state = 0.
//     /IT12 (4) with off-state = 0.
//  Identified 0 set control primary inputs.
//  Identified 1 reset control primary inputs.
//     /IRST (1) with off-state = 0.
//  Identified 0 read control primary inputs.
//  Identified 0 write control primary inputs.
-----------------------------------------------------------------------
//  Total number of internal lines is 105 (35 clocks, 35 sets , 35 resets,
//  0 reads, 0 writes).
//  Total number of controlled internal lines is 25 (17 clocks, 0 sets ,
//  8 resets, 0 reads, 0 writes).
//  Total number of uncontrolled internal lines is 80 (18 clocks, 35 sets,
//  27 resets, 0 reads, 0 writes).
//  Total number of added primary input controls 0 (0 clocks, 0 sets ,
//  0 resets, 0 reads, 0 writes).
-----------------------------------------------------------------------
```

analyze_control_signals -auto_fix

Related Topics

add_clocks

add_read_controls

add_write_controls

report_clocks

report_read_controls

report_write_controls

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_drc_violation

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -ijtag, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Generates a schematic in the schematic windows of the portion of the design involved with the specified rule violation number.

Usage

analyze_drc_violation ***rule_id-occurrence#*** [-append]

Description

When you issue the analyze_drc_violation command, the tool includes different simulation data in the netlist depending on the type of rule violation. The tool automatically opens the schematic windows of Tessent Visualizer if it is not already open, and sets the gate reporting to match the type of rule violation, just as if you had manually issued the set_gate_report command. The gate report setting is shown on the status bar. For some DRC violations (E5 for example), the gate level is automatically set to primitive.

Note that the following DRC errors do not have additional information that can be displayed with Tessent Visualizer:

- B2, B3, B4, B17
- E1, E12, E13, E14
- G rules
- K rules
- P rules
- T1, T8-T11, T13-T15, T18-T23
- W1-W19, W32, W33

For more information about displaying DRC information, refer to the report_drc_rules command. For information about each DRC, refer to "Design Rule Checking" on page 3167.

Arguments

- ***rule_id-occurrence#***

  A literal and integer argument pair that specifies the exact design rule violation (including the occurrence) that you want to analyze. The tool traces the violation back to the probable cause and then the schematic viewer displays all the gates in that trace. The argument must include the design rules violation ID (*rule_id*), the specific occurrence number of that violation, and the hyphen between them. For example, you can analyze the second occurrence of the C3 rule by specifying C3-2. The tool assigns the occurrences of the rules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

violations as it encounters them, and you cannot change either the rule identification number or the ordering of the specific violations.

- -append

  An optional switch that specifies to append the new schematic to the existing contents of the schematic windows instead of replacing the contents. If you do not specify the -append switch, the existing schematic is deleted and new data is shown in the schematic windows.

Examples

The following example defines the off-state of a clock incorrectly, causing a C2 rule violation. When a rule violation occurs, you can use the schematic windows to analyze the probable cause of the error.

With this example, the schematic windows display the sequential element associated with the clk input, along with a backward trace through the gates and nets to the associated primary input.

add_clocks 0 clk
set_system_mode analysis

```
...
//  ---------------------------------------------------------
//  Begin scan clock rules checking.
//  ---------------------------------------------------------
//  1 scan clock/set/reset lines have been identified.
//  All scan clocks successfully passed off-state check.
//  Error: Clock /CLK cannot capture data with other clocks off. (C2-1)
```

analyze_drc_violation c2-1

Related Topics

open_visualizer

report_drc_rules

set_drc_handling

set_visualizer_preferences

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_fault

Context: dft -edt, patterns -scan

Mode: analysis

Performs an analysis to identify why a fault is not detected and optionally displays the relevant circuitry in Tessent Visualizer.

Usage

Struck/Transition Usage

analyze_fault ***pin_pathname*** **{-Stuck_at {0 | 1} | -Rise | -Fall}**
[-Observe *gate_id#*] [-Boundary] [-Continue] [-Display [APpend]]

UDFM Usage

analyze_fault ***instance_pathname*** **-Udfm_type** ***name***
[{-Cell *name* | -Module *name*}] {-Fault *name* | -Bridge -Id *bridge_id*}
[-Observe *gate_id#*] [-BOundary] [-COntinue] [-Display [APpend]]

Path Delay Usage

analyze_fault ***path_name*** **{-Rise | -Fall}** [-Observe *gate_id#*] [-Auto] [-Display [APpend]]

Description

The analyze_fault command performs an analysis to identify why the fault that you specify was not detected. You can use the -Observe switch to specify the observe point for the sensitization analysis.

If you are using Tessent Visualizer, you can specify the -Display switch to graphically display the relevant circuitry for any fault detection. This may assist you in identifying either why a fault wasn't detected or how a fault was detected.

The fault analysis performed by the analyze_fault command consists of the following actions:

1. A message is given if the selected fault has been nofaulted.
2. A message is given to identify if the fault is in the current fault list. If the fault is in the current fault list and the fault is the representative member, its fault classification shows the fault class and the subclass information.
3. If the fault was not identified as included in the active fault list, the basic fault analysis is performed that determines if the fault can be classified as unused, tied, blocked, or detected by implication.
4. If the fault category is detected by simulation, detected by implication, or unused, a message is given and the analysis is terminated. You can override the termination by using the -Continue switch.
5. If the fault category is tied, all sources of the tied condition are identified and the analysis is terminated.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

6. If the fault category is blocked, all blockage points (100 maximum) are identified. For each blockage point, all sources of the tied conditions causing the blockage are identified. The analysis then terminates.
7. The states that result from all constrained pins and stable non-scan cells are calculated.
8. If the fault site is now prevented from attaining the necessary state, a message is given indicating the fault is tied by constrained logic, all sources of the tied condition are identified, and the analysis then terminates.
9. An analysis is made to identify all blockage points (100 maximum) and all potential detection points (25 maximum).
10. If there are no potential detection points, the blockage points are identified and for those points blocked by tied logic, all sources of the tied condition are identified. The analysis then terminates.
11. If there were potential detection points, the detection points are identified (25 maximum).
12. A controllability test generation is performed to determine if the fault site can be controlled. If successful, the test generation values are displayed using “set_gate_report parallel_pattern 0”. If unsuccessful, the analysis then terminates.
13. If an observe point is not selected, a complete test generation is attempted where the fault is sensitized from the fault site to any unblocked point. Potential problem points in any sensitization path are identified. The points include tri-state driver enable lines, transparent latch data lines, clock_pos, wire gates, latch and flip-flop set/reset/clock lines, RAM write/read/address/data lines, and ROM read/address lines. If the test generation is successful, the test generation values are displayed using “set_gate_report parallel_pattern 1”.
14. If an observe point is selected, the fault is sensitized from the fault site to the observe point. Potential problem points in the sensitization path are identified. If the sensitization is successful, the test generation values are displayed using “set_gate_report parallel_pattern 1”.

Note

Only UDFM, stuck-at, path delay, and transition fault types can be analyzed using the analyze_fault command.

When the fault type is path delay, the analyze_fault command performs an unsensitizable path analysis when the ATPG run is unable to create a test pattern. If the analysis finds that some segment of the path is unsensitizable, it attempts to find a minimum number of gates in the path that are required to prove the path unsensitizable.

For example:

```
analyze_fault path37 -s 0
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

produces the following report:

```
//  ---------------------------------------------------------
//  Path delay fault analysis for path37 slow to rise on
// launch point = 22032, capture point = 167521
//  ---------------------------------------------------------
//  Path delay test generation not successful for capture point = 167521
//  due to atpg_untestable.
//  Begin unsensitizable path analysis for path=path37
//  Cycle=0 path sensitization check failed status=redundant
//  Simultaneous sensitization of 2 path segments cannot be achieved
//  (status=redundant).
//  These are:
//      OR   145279 (I1) + ;
//      ...
//      AND  150252 (I1) + ;
//  Path is a static unsensitizable path. Robust detection will be
//  impossible.
```

The analyze_fault report shows that it is impossible to use both input 1 (the second input, inputs are numbered starting at 0) of the OR gate 145279 while at the same time using input 1 of the AND gate 150252. The ellipsis (...) indicates that there are other gates in the path between the reported gates that are not relevant to the unsensitizable path problem.

The "delete_fault_sites -Unsensitizable_paths" command lets you delete proven unsensitizable paths.

When the fault type is transition, the analyze_fault command performs the following analysis steps:

1. The controllability of logic value 0 and 1 at the fault site is analyzed.

   If the analysis succeeds, the fault site is controllable. You can access the analysis results by setting the gate report type to be parallel pattern 0(2) to set logic value 1(0) at the fault site for a slow-to-rise fault, or logic value 0(1) at the fault site for slow-to-fall fault.

   If the tool aborts this step or it otherwise fails, the fault cannot be detected due to a lack of controllability at the fault site. The tool then terminates the analysis unless you force it to continue.

2. The ability to launch a slow-to-rise (0 -> 1) or slow-to-fall (1 -> 0) transition at the fault site is analyzed.

   If the analysis succeeds, the transition you need can be created at the fault site. You can access the analysis results by setting the gate report type to be parallel pattern 3.

   If the tool aborts this step or it otherwise fails, the fault cannot be detected because a transition cannot be launched at the fault site. The tool terminates the analysis unless you force it to continue.

3. The tool attempts to generate a test pattern for the transition fault being analyzed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If successful, a test pattern exists to detect the fault. You can access the analysis results by setting the gate report type to be parallel pattern 1 and the analysis process terminates.

If the tool aborts this step or it otherwise fails, the tool proceeds to the next step.

4. The tool attempts to generate a test pattern for a stuck-at fault at the same fault site.

   If successful, there exists a sensitive path from the fault site to an observation point. The reason for aborting or failing to generate the test pattern for the original transition fault is that launching a transition at the fault site may block the fault propagation path. You can access the analysis results for the stuck-at fault by setting the gate report type to be parallel pattern 4.

Arguments

- ***pin_pathname***

  A required string that specifies the pin pathname of the fault where you want to perform stuck/transition analysis.

- ***path_name***

  A required string that specifies the pathname where you want to perform path delay analysis.

- ***instance_pathname***

  A required string that specifies the instance for which you want to perform UDFM fault analysis.

- -Stuck_at 0 | 1

  A switch and literal pair that specifies the stuck-at fault value that you want to analyze. The stuck-at values are as follows:

  0 — A literal that specifies that the tool analyze the *pin_pathname* for a “stuck-at-0” fault.

  1 — A literal that specifies that the tool analyze the *pin_pathname* for a “stuck-at-1” fault.

- **-Rise**

  A switch that specifies a slow-to-rise transition fault.

- **-Fall**

  A switch that specifies a slow-to-fall transition fault.

- **-Udfm_type *name***

  A required switch and value pair that identifies the type of UDFM fault to analyze.

- -Cell *name* | -Module *name*

  An optional switch and value pair that identifies the cell or module containing a UDFM fault.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Fault *name*

  An optional switch and value pair that identifies the UDFM fault to analyze.

- -Bridge -Id *bridge_id*

  An optional switch, and switch and value pair, that specifies an identifier for a UDFM bridge fault to analyze.

- -Observe *gate_id#*

  An optional switch and integer pair that specifies the observe point for the sensitization analysis.

  gate_id# — An integer that specifies a gate identification number whose location you want to use as the observe point for the sensitization analysis.

- -Boundary

  An optional switch that specifies to display the boundary faults when analyzing an ATPG untestable, tied, blocked, or redundant fault.

- -AUto

  An optional switch, applicable only to path delay faults, that automatically determines how far along the path the tool can successfully propagate a path delay fault.

- -Continue

  An optional switch that forces the tool to complete the analysis of faults that have already been detected by the pattern set. This lets you inspect the generated pattern by using the report_faults command.

  If you do not specify this switch and the tool detects the fault category by simulation, by implication, or as unused, then the tool displays a message and terminates the analysis.

- -Display [APpend]

  An optional switch and literal pair that causes the tool to create a gate list relevant to the fault being analyzed and to update the Tessent Visualizer Flat Schematic window with this information, If the Append switch is included, the information is added to the existing contents of the Flat Schematic window.

  The schematic view contains annotations for the following cases as required:

  - Successful ATPG — All gates that sensitize the fault effects to an observable point are added to the gate list. The pins on these gates are annotated with the simulated value that results from the pattern that was created.
  - Uncontrollable Fault Site due to Forbidden Conditions — All gates that prevent the fault site from attaining the required state are added to the gate list. The pins on these gates are annotated with the constrain_value data.
  - Blocked Fault Site due to Forbidden Conditions — All gates that sensitize the fault effects to the blockage point and all gates that prevent the blockage points from

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

attaining the necessary values are added to the gate list. The pins on these gates are annotated with the constrain_value data.

Examples

The following example performs test pattern generation, then performs an analysis to determine why the tool did not detect a stuck-at-1 fault at pin /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/Q:

set_system_mode analysis
create_patterns
report test data -class au

```
//  fault analysis summary of 17 faults
//  number faults tied by constraints = 7
//  number faults connected to nonscan_latch = 15
//  number faults connected to clock/set/reset = 8
//  number faults connected from clock = 6
//  number faults connected from nonscan_latch = 7
//  number faults unclassified = 0
```

report_faults -class au

```
type    code    pin_pathname
----   ------   ------------------------------------
  0    AU.TC    /tcb_tck
  1    AU.TC    /tcb_tck
  0    AU.TC    /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/SD
  1    AU.PC    /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/SD
  0    AU.TC    /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/TE
  1    AU.PC    /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/TE
  0    AU.PC    /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/D
  1    AU.PC    /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/D
  0    AU.TC    /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/TI
  1    AU.TC    /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/TI
  1    AU.TC    /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/Q
  0    AU.TC    /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/CP
  1    AU.TC    /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2/CP
  1    AU.TC    /ram_32_36/u_tcb/tcbreg_inst/SKEW/Q_reg/inst_2/Q
  0    AU.TC    /ram_32_36/u_tcb/tcbreg_inst/SKEW/Q_reg/inst_2/CP
  1    AU.TC    /ram_32_36/u_tcb/tcbreg_inst/SKEW/Q_reg/inst_2/CP
  1    AU.TC    /ram_32_36/u_tcb/tcbreg_inst/SKEW/Q_reg/inst_2/D
```

analyze_fault /ram_32_36/u_tc/tcbreg_inst/SHIFT_0/Q_reg/inst_2/Q -stuck_at 1

```
//  ---------------------------------------------------------------------------
//  Fault analysis for /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2
//  (29217) output Q stuck at 1
//  ---------------------------------------------------------------------------
//  Current fault classification = AU.TC
//  Fault site sequential depth: Control_0 = 255, Control_1 = 0, Observe = 255.
//      Fault site is tied to 1 due to constrained logic.
//      Tied gate source: 1  /ram_32_36/u_tcb/tcbreg_inst/SHIFT_0/Q_reg/inst_2
//      (29217)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following example analyzes a UDFM fault with multiple test alternatives:

UDFM Content:

```
UdfmType ("UDFM_sff_test") {
      Cell ("sff") {
         …
         Fault ("f6") {
            Test {Impact:100;StaticFault{"D":0;} Conditions{"D":0;}}
            Test {Impact:100;StaticFault{"D":0;} Conditions{"Q":0;}}
         }
   }
```

Example analyze_fault report:

analyze_fault U_2 -fault f6 -udfm_type UDFM_sff_test

```
//  ---------------------------------------------------------------------------
//  Fault analysis for UDFM fault UDFM_sff_test/sff/f6
//  ---------------------------------------------------------------------------
//  Current fault classification = AU (atpg_untestable)
//  Analyzing test alternative 1...
//  Fault location '/U_2/' (16) sequential depth: Control_0 = 0, Control_1 = 0,
//  Observe = 1.
//  Analyze if the fault location can be controlled without active conditions...
//      The fault location was set to 1 successfully.
//  Analyze if the condition D can be justified...
//  Test condition justification failed.
//  Search potential observation point...
//      Fault effect is blocked at /U_2 (20) due to unobservable scan cell.
//    The fault effect cannot propagate to any observation point.
//  Analyzing test alternative 2...
//  Fault location '/U_2/' (16) sequential depth: Control_0 = 0, Control_1 = 0,
//  Observe = 1.
//  Analyze if the fault location can be controlled without active conditions...
//      The fault location was set to 1 successfully.
//  Analyze if the condition Q can be justified...
//  Test condition justification passed.
//  Analyze if the fault location can be controlled with active conditions ...
//      The fault location can be controlled with active conditions.
//  Search potential observation point...
//      Fault effect is blocked at /U_2 (20) due to unobservable scan cell.
//    The fault effect cannot propagate to any observation point.
//
//  Fault analysis summary
//
//  UDFM fault: 'UDFM_sff_test/sff/f6'
//  Current fault classification: AU
//  Test 1: Status=unobservable
//     Fault location: State=stuck-0, '/U_2/' (16)
//     Condition D: State=0, Location='/U_4/Y' (14), Status=not justified
//  Test 2: Status=unobservable
//     Fault location: State=stuck-0, '/U_2/' (16)
//     Condition Q: State=0, Location='/U_2/Q' (11), Status=justified
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

delete_fault_sites

report_faults

report_testbench_simulation_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_graybox

Context: patterns -scan, patterns -ijtag

Mode: analysis

Identifies the instances and nets to be included in the graybox netlist by automatically setting their in_graybox attributes to true. Subsequently, a “write_design -graybox” command writes out the graybox netlist for the marked instances and nets.

Usage

Context: patterns -scan

analyze_graybox [-scan] [-preserve_instances *instance_objects*]
[-max_sequential_levels {*number* | unlimited}] [-collect_reporting_data]

Context: patterns -ijtag

analyze_graybox [-ijtag] [-preserve_instances *instance_objects*]

Description

Notes About the Scan Graybox Analysis

The Scan graybox analysis is only supported in the -no_rtl context. It is performed in analysis mode when the external scan mode successfully completed DRC.

All module names, except the top-level module, are uniquified by adding a prefix in the generated graybox netlist. You can change the default prefix by using the -edited_module_prefix option of the set_scan_insertion_options command.

Modules with the exclude_from_concatenated_netlist attribute set to true are not included inside the graybox netlist and their module names are not changed. This enables you to still bind the real simulation model of the hard macros such as memories or PLL models during simulation with the scan graybox views. Child physical region modules are uniquified and pruned into the parent block’s scan graybox if anything within them is part of the scan mode. A child physical region module is recognized with the is_physical_module attribute being set to true.

The scan graybox analysis performs the identification by tracing backward from all primary output pins, wrapper chain sequential cell input pins, and specified preserve_instance input pins. In general, the backward tracing used by the scan graybox analysis considers a path sensitization based on the state-stability values. These values are obtained by simulating the pin constraints and test-setup procedures provided for the external test mode. However, a pure structural tracing method is used when tracing backward from preserve_instance input pins.

Additional gates are included in the scan graybox to minimize the fault coverage differences in the fault list between when the graybox netlist is used and when the complete netlist is used. If a gate has multiple fanouts in the complete netlist, then it must also have at least two fanouts in the graybox netlist. The tool includes additional gates in the graybox netlist to maintain a fanout

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

of at least two for each of the gates that was originally identified during the feedback analysis. This is done to avoid inaccuracy in equivalent fault accounting between the two netlists.

Exclude the scan-out pins of the core chains from the backward tracing to prevent marking internal (core) logic for the scan graybox. To do this, set the ignore_for_graybox attribute of the scan-out pins to true using the set_attribute_value command.

Use the -preserve_instances option to include arbitrary instances (the instantiations of modules or the instantiations of library models) in the scan graybox view. The tool bypasses the graybox analysis within specified preserve module instances and automatically includes all of their sub-instances and nets in the scan graybox view. This enables you to overcome the limitations imposed by the sensitized path-based backward tracing, such as not permitting complex test_setup procedures that initialize non-scan cells in clocks paths and limit the sequential depth.

One typical -preserve_instances option usage is when the design includes a user-defined clock control module. You can specify the entire module as a preserve_instance if it cannot be identified by the scan graybox analysis.

To understand the reason why a given instance was preserved in the scan graybox, display the value of the in_graybox_reason attribute after you have called analyze_graybox in analysis mode.

The tool automatically adds certain categories of instances to the list of preserved instances during scan graybox generation.

- It automatically preserves instances of the EDT logic identified by EDT Finder and EDT logic in the channel pipeline stages, because EDT Finder is enabled by default. If you have disabled EDT Finder, use the -preserve_instances option to preserve these instances.
- It automatically preserves all instrument instances that you define using the add_core_instances command. For more information, refer to Tessent Core Description (TCD) in the *Tessent TestKompress User's Manual*.

  Note

  This only applies if you use EDT in the external mode so the tool defines EDT in the current mode when performing graybox analysis. If you do not use an EDT controller in the core for external test, it does not need to be preserved in the external mode.

- If you are using IJTAG and have an ICL description of the current design, then any ICL instance that is identified as part of the scan resource network and that can be mapped to a design instance is automatically preserved. An ICL module is identified as a Scan Resource Instrument when it has the ICL module attribute keep_active_during_scan_test not set to false. If you create the scan instrument using the Tessent tools (for example, an EDT controller), and insert the IJTAG network using

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

the create_dft_specification and process_dft_specification commands, the keep_active_during_scan_test attribute is already added by the tool. For more information, refer to IJTAG Network Insertion in the *Tessent IJTAG User's Manual*.

Note

The tool adds all instances marked with the preserve_in_graybox attribute to the list of instances specified with the -preserve_instances switch and handles them in the same way.

You can also set the preserve_in_graybox attribute on individual nets. This includes the specified net, but does not include the affected instances, and should be used with caution because you need to manually identify the instances that are connected to the net and make sure they are either preserved automatically, or specified with the -preserve_instances switch, or by also setting the attribute on the instances.

Currently, Tessent only supports Mux-DFF scan architecture with the graybox functionality.

For more information about scan graybox concepts and procedures, refer to Graybox Overview in the *Tessent Scan and ATPG User's Manual*.

The graybox netlist includes any combination feed-through paths that are in your design.

Notes About the IJTAG Graybox Analysis

The IJTAG graybox analysis works in netlist mode.

In netlist mode, the tool creates a single concatenated file with everything that needs to be preserved in the IJTAG graybox view. It instantiates child physical blocks but does not include their module definitions in the parent's IJTAG graybox. This differs from the scan graybox view, which may include the pruned child physical block module definitions if anything is needed in the given scan mode. It includes IJTAG instances located with child sub-blocks in the IJTAG graybox view of their parent physical block. Refer to the set_design_level command for a definition of physical block versus sub-block.

You can analyze and create the IJTAG graybox view using the extract_icl command by specifying the "-create_ijtag_graybox on" switch in RTL mode. The extract_icl command preserves any logic between the fully preserved instances and the primary inputs and outputs, and between any groups of fully preserved instances. The pruned IJTAG graybox is constructed from the quick-synthesized view and includes the instantiation of the RTL cells, where needed. If you want to visualize the content of the IJTAG graybox, use the "read_design [get_current_design] -view ijtag_graybox -force" command.

Note

Similarly to scan graybox analysis, you can use the -preserve_instances switch to add any existing logic to the ijtag_graybox.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In RTL mode, the IJTAG graybox comprises the pruned logic interconnecting the IJTAG, the user-specified preserve_instances between them, and the primary inputs and outputs. The tool writes the pruned modules into a single file per language or format combination, and those files include only the instantiation of IJTAG and user-specified full preserve_instances. The tool also creates a design load dictionary with read entries for each created concatenated file, and a read entry for the original module definitions of the IJTAG and user-specified full preserved instances. The child physical block and child sub-block are instantiated in the pruned modules, but their module definitions are not referenced within the design load dictionary. Use "read_design -view full | ijtag_graybox | interface" on the child blocks to load the view you need.

Note

As mentioned earlier, the graybox includes combinational feed-throughs.

To understand the reason why a given instance was preserved in the IJTAG graybox, display the value of the in_graybox_reason attribute after you have called "analyze_graybox -ijtag" in analysis mode but before you call "write_design -graybox".

IJTAG Graybox Limitations in RTL Mode

Certain limitations can prevent creating an IJTAG graybox in RTL mode. These revolve around particular cases for VHDL and System Verilog modules. The situations involving these limitations are uncommon, so creating an RTL IJTAG graybox should be possible for most designs.

- Limitations for all VHDL modules

  You must use quick synthesis to synthesize VHDL modules that must be pruned and kept in the IJTAG graybox. If you do not synthesize them, the tool reports an error and lists them. To force synthesis, set the synthesize_before_analysis module-level attribute before calling the "extract_icl -create_ijtag_graybox on" command.

  Tip

  You can use the "set_attribute_value [get_modules -filter language==vhdl] -name synthesize_before_analysis" command if you want to ensure the quick synthesis of all the VHDL modules up front.

- Limitations for top modules of physical and sub-blocks written in VHDL

  These modules can only have simple ports such that the port names do not change between the RTL and the quick-synthesized views. Generics and parameters are also not permitted in these top-level VHDL modules. This limitation is only for the top VHDL module of a physical or sub-block and does not apply to lower VHDL modules.

- Limitations for top module of physical and sub block written in System Verilog

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

These modules may not contain ports with unpacked or enum types defined in the $unit space. They may have such ports if their types are declared within packages.

Note

The IJTAG graybox should not be used for ATPG and fault simulation. It does not include any additional logic to ensure that the fault accounting in the graybox matches the full netlist. This means you will not find any instances in the graybox where the in_graybox_reason attribute reports stem_preservation or stem_preservation_clock_path.

Arguments

- -ijtag

  An optional switch that specifies to generate an IJTAG graybox. This switch is inferred when you invoke the analyze_graybox command within the "patterns -ijtag" context.

  The tool automatically does the IJTAG graybox analysis within the extract_icl and "write_design -tsdb" commands. When you run extract_icl after the process_dft_specification command in the no_rtl mode, the tool cannot automatically extract the IJTAG graybox because the newly inserted IJTAG instruments are not yet synthesized, and thus could not be included into the concatenated IJTAG graybox netlist. When you invoke extract_icl with the -create_ijtag_graybox switch in the above situation, the tool generates the following warning:

```
//  Warning: The IJTAG graybox was not generated because module
definitions
//           matched to ICL instruments are not loaded in memory.
//           In order to generate the IJTAG graybox, issue the
following commands in setup mode:
//
//             run_synthesis
//             set_context patterns -ijtag
//             read_design chip_top -force
//             set_current_design chip_top
//             set_system_mode analysis
//             analyze_graybox -ijtag
//             write_design -tsdb -graybox
//
//           Here is the list ICL modules for which actual modules
are not in memory:
//             chip_top_gate_tessent_bscan_interface
//             chip_top_gate_tessent_sib_pb1
//             chip_top_gate_tessent_sib_pb2
//             chip_top_gate_tessent_sib_pb3
//             chip_top_gate_tessent_sib_sri
//             chip_top_gate_tessent_tap_main
```

- -scan

  An optional switch that specifies to generate a scan graybox. This switch is inferred when you invoke the analyze_graybox command within the "patterns -scan" context.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -preserve_instances *instance_objects*

  An optional switch that specifies a collection of one or more instances or a well formatted Tcl list of one or more instance names. The tool forces the specified instances to be preserved into the graybox view. It also automatically preserves the combinational logic between the preserved instances, and between the preserved instances and the primary inputs and outputs.

- -max_sequential_levels {*number* | unlimited}

  An optional switch and integer or literal pair that is only relevant to the scan graybox analysis. It specifies the number of levels of non-scan cells when tracing back from the POs or from the data input pins of the wrapper cells before reaching the PIs or other wrapper cells. If the specified sequential levels limit is exceeded during backtracing, all the logic so far traced from the origin pin is dropped and not included in the graybox netlist. The limit specified by this switch does not apply to the backtracing from the clock, set, reset and scan enable pins of the wrapper cells. Sequential cells initialized to constant values (Tie-0 or Tie-1) with an active set or reset pin are not counted as sequential levels during backtracing. The "number" argument can accept a value of 0 or greater, where 1 is the default value when this switch is not specified with the command. The "unlimited" argument specifies to not limit the number of sequential levels during backtracing.

- -collect_reporting_data

  An optional switch that that is only relevant to the scan graybox analysis. It specifies to count the number of leaf instances (library model instances at design level) that are marked for each backward tracing origin, such as a PO, a wrapper cell input pin, or a "preserve_instance" input pin (all input pins combined). This switch is required if you want to use the report_graybox_statistics command to report the collected data. Collecting reporting data requires additional netlist tracing and could impact analysis execution performance.

Examples

Example 1

The following command performs graybox analysis on the current design. The module instance "tap_ctrl_ins" and all sub-instances within are marked for graybox logic without going through the graybox analysis.

```
analyze_graybox -preserve_instances tap_ctrl_ins -collect_reporting_data
report_graybox_statistics -top 10
```

Example 2

The following example shows a summary from the analyze_graybox command. The report includes details on the number of graybox instances and where they are located.

```
sub-command: analyze_graybox
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// The 'in_graybox' attribute has been set for 2340 design
// instances (8.08% of 28946 total design instances).
// 909 (3.14%) instances were identified in ICL scan resource instruments.
// 3 (0.01%) instances were identified in the logic from the preserved
//           instances to other preserved instances, wrapper chains, and
//           design boundary.
// 1428 (4.93%) instances were identified in the wrapper chains and the
//              logic from the wrapper chains to the design boundary.
```

Related Topics

report_graybox_statistics

set_attribute_value

write_design

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_icl_suspect

Context: patterns -silicon_insight -ijtag

Mode: setup

Analyzes the suspects for a configured scan path of interest.

Usage

analyze_icl_suspect {***suspect_signature*** | **-verify_scan_path**} [-pattern_type {stil | svf}] [-report]

Description

You can analyze the ICL network by either specifying a suspect signature or the -verify_path_switch.

Use this command when manually debugging IJTAG networks as described in "Diagnosing and Debugging IJTAG Networks" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

When you specify a suspect signature, the tool isolates the suspect node based on the signature, performs analysis with setup and flush patterns, and returns a diagnosis result that indicates what type of defect could be causing the observed behavior. The possible results are:

Table 3-17. Suspect Signature Diagnosis Results

Result	Description
?	Indeterminable state. Refer to "Manually Debugging an IJTAG Network" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.
good	Functions as designed.
register_stuck@0	One or more flops in the register suspect are stuck at 0.
register_stuck@1	One or more flops in the register suspect are stuck at 1.
register_inconsistent_length @*icl_length*@*dut_length*	Inconsistent length specified in the ICL and observed from the DUT. For example: `register_inconsistent_length@16@17` This message indicates that the length specified in ICL is 16 while the length observed from the DUT is 17.
sib_stuck@0	The SIB is stuck at 0 (client mode).
sib_stuck@*index*	The SIB is stuck at host mode for the specified index, where the index is 1 or higher.
mux_stuck@*index*	The scan mux is stuck at input at the specified index.

When you specify the -verify_path_switch, the tool assumes that you have already configured a scan path of interest from TDI to TDO and performs verification of the scan path.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***suspect_signature*** | **-verify_scan_path**

  A required choice between specifying a suspect signature or the -verify_scan_path switch.

  suspect_signature — Specify the suspect signature in the format *suspect_name*[*@input_index*], where the suspect name can be the hierarchical name of a scan register, scan mux, or SIB as returned in the diagnosis results suspects list. The tool assumes a 0 state.

  For SIBs, the input index is the host input with 0 reserved for client mode. For scan muxes, the input index is the programmed selected input.

  -verify_scan_path — This switch returns a pass or fail result for the scan path of interest.

- -pattern_type {stil | svf}

  An optional switch and literal pair that specifies the pattern type—STIL or SVF—for the generated temporary (setup and flush) patterns that are run to analyze the scan path. The default is SVF.

- -report

  An optional switch that prints the analysis results to the transcript.

Examples

analyze_icl_suspect block1_l1.sib1@1 -report

analyze_icl_suspect block1_l1.tdr1.R[7:0]

analyze_icl_suspect -verify_scan_path -pattern_type stil

Related Topics

assert_icl_sib

create_icl_flush_pattern

create_icl_setup_pattern

deassert_icl_sib

get_icl_network

get_icl_scan_mux

isolate_icl_node

report_icl_network

set_icl_network

set_icl_scan_mux

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_layout_hierarchy

Context: patterns -scan_diagnosis

Mode: setup

Analyzes the layout LEF/DEF hierarchy and stores the analysis results in a layout hierarchy database.

Usage

analyze_layout_hierarchy ***layout_hierarchy_db_dir*** [-lef *lef_filename* … | -leflist *leflist*] [-def *def_filename* ... | -deflist *deflist*] [-replace] [-threads *N*]

Description

This command generates a layout hierarchy database that stores the layout hierarchy of the LEF/DEF designs. By default, the tool performs multi-threaded analysis. Specify this command as a one-time setup step before loading the flat model and pattern files, and specifying the create_layout command.

After analyzing the layout hierarchy, the tool generates a layout hierarchy database that contains the results of the analysis. From this database, you can:

- Report estimates of the resources required to generate an LDB.
- Ensure that only one top DEF module exists.
- Optimize chip-mapped core-level LDB creation times when you are working with hierarchical designs.

Refer to "Estimation of Resources Required for Generating LDBs" in the *Tessent Diagnosis User's Manual* for more information.

Arguments

- ***layout_hierarchy_db_dir***

  A required string that specifies the name of the layout hierarchy database that contains the results of the analysis, as follows:

  - *layoutHierarchy.hierdb*—Database the describes the layout hierarchy.
  - *def_hierarchy_info.gds*—GDSII file of the DEF file structures.
  - *def_hierarchy_info.gds.layerprops*—Layer properties file for layout hierarchy file def_hierarchy_info.gds.

- -lef *lef_filename* …

  An optional switch and string that specifies space-separated name(s) of the LEF file(s).

- -leflist *leflist*

  An optional switch and string pair that specifies a Tcl list of the LEF file(s).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -def *def_filename* ...

  An optional switch and string that specifies space-separated name(s) of the DEF file(s).

- -deflist *deflist*

  An optional switch and string pair that specifies a Tcl list of the DEF file(s).

- -replace

  An optional switch that overwrites the existing layout hierarchy database with the newly generated layout hierarchy database.

- -threads *N*

  An optional switch and integer pair that specifies the number of threads to run in parallel during the analysis. The default value is 0, which indicates to use all the available CPUs on the current machine, if needed. A value less than 0 uses the number of available CPUs less *N*. A value greater than 0 uses *N* CPUs. The number of threads is always capped by the number of DEF files currently being processed.

Examples

The following examples illustrate how the -threads option functions, where:

- nthreads: number of threads.
- ncpus: number of CPUs.
- ndef: number of specified DEF files.

Example 1

Suppose the maximum number of CPUs is 32 and you specified four DEF files.

Table 3-18. Example 1 -threads Option

Threads	Behavior
Undefined or 0	Use nthreads = ncpus Build the minimum (nthreads,ndef) = 4
-threads 2	Use nthreads = minimum(threads,ncpus) = 2 Build the minimum (nthreads,ndef) = 2
-threads 200	Use nthreads = minimum(threads,ncpus) = 32 Build the minimum (nthreads,ndef) = 4
-threads -2	Use nthreads = ncpus-2 = 30 (If negative, then set to 1.) Build the minimum (nthreads,ndef) = 4
-threads -200	Use nthreads = ncpus-200 = -168 = 1 (If negative, then set to 1.) Build the minimum (nthreads,ndef) = 1
-threads -29	Use nthreads = ncpus-29 = 3 (If negative, then set to 1.) Build the minimum (nthreads,ndef) = 3

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

Suppose the maximum number of CPUs is 32 and you specified 400 DEF files.

Table 3-19. Example 2 -threads Option

Threads	Behavior
Undefined or 0	Use nthreads = ncpus Build the minimum (nthreads,ndef) = 32
-threads 2	Use nthreads = minimum(threads,ncpus) = 2 Build the minimum (nthreads,ndef) = 2
-threads 200	Use nthreads = minimum(threads,ncpus) = 32 Build the minimum (nthreads,ndef) = 32
-threads -2	Use nthreads = ncpus-2 = 30 (If negative, then set to 1.) Build the minimum (nthreads,ndef) = 30
-threads -200	Use nthreads = ncpus-200 = -168 = 1 (If negative, then set to 1.) Build the minimum (nthreads,ndef) = 1
-threads -29	Use nthreads = ncpus-29 = 3 (If negative, then set to 1.) Build the minimum (nthreads,ndef) = 3

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_restrictions

Context: dft -edt, patterns -scan

Mode: analysis

Performs an analysis to automatically determine sources of the problem from a failed ATPG run.

Usage

All Fault Types Except IDDQ Faults Usage

analyze_restrictions [-Auto] [-COnflict_source_analysis]

IDDQ Faults Usage

analyze_restrictions [-INT_float_auto_except]

Description

The analyze_restrictions command reports the ATPG restrictions that caused a failed ATPG run. If you issue the analyze_restrictions command without an argument, the default is -Auto. The analysis may be lengthy depending on the number of restrictions. You can stop the analysis at any time by pressing Ctrl-C.

Arguments

- -Auto

  An optional switch that automatically performs the following step-by-step analysis:

  - **Step 1** — The tool identifies the type of constraints that failed the analysis.
  - **Step 2** — The tool analyzes the problem constraint types and finds the minimum constraints that prevent the tool from passing the analysis.
  - **Step 3** — Optional. The tool performs a detailed analysis of each constraint identified in Step 2. You perform this detailed analysis by using the -COnflict_source_analysis switch in conjunction with the -Auto switch.

  The set of constraints that conflict with the constraint are reported. The tool analyzes the constraints in the following order:

  - Dynamic ATPG constraints
  - Dynamic cell constraints
  - Static ATPG constraints
  - Static cell constraints
  - Named capture procedure
  - Bus floating checking
  - Bus contention (with the specified driver restriction)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

  - Port contention

- -COnflict_source_analysis

  An optional switch that performs a detailed analysis (Step 3) of each constraint identified in Step 2 in the -Auto switch description. This option can increase the tool's runtime under certain circumstances.

- -INT_float_auto_except

  An optional switch that creates Iddq exceptions automatically for the "-int_float" Iddq check failure sites found during the analysis. Iddq exceptions are defined for failures that are the result of "-int_float" Iddq checks previously specified with the set_iddq_checks command. You can report the automatically added exceptions with the report_iddq_exceptions command or use the delete_iddq_exceptions command to remove unwanted exceptions.

Examples

Example 1

The following example analyzes a failed ATPG run in which a user-specified ATPG constraint is conflicting with a cell constraint and causes the ATPG to fail test generation.

create_patterns

```
//  Error: Current restrictions cannot be simultaneously satisfied with
//  ANY abort limit. Status=atpg_untestable.
//  Halting run.  Suggest analyze_restrictions.
//  You can specify the -NOAnalyze option in command "create_patterns"
//  to force the run, but coverage will probably be low.
```

analyze_restrictions

```
//   -----------------------------------
//  | Step-1  Global constraint analysis |
//   -----------------------------------
//    Order-1: dynamic ATPG constraints, analysis=to_be_analyzed
//   ----------------------------------------------------
//  | Step-2  Identifying minimum conflicting constraint |
//   ----------------------------------------------------
//  Following atpg constraints conflict with the rest of constraints:
//    atpg constraint: 1 dynamic  /FF3/Q (10)
//  You can use -conflict_source_analysis switch to analyze which
//  constraints conflict with the above constraints.
```

To perform detailed analysis of the conflict, then issue the analyze_restrictions command with the optional -conflict_source_analysis switch:

analyze_restrictions -conflict_source_analysis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//   -----------------------------------
//  | Step-1  Global constraint analysis |
//   -----------------------------------
//    Order-1: dynamic ATPG constraints, analysis=to_be_analyzed
//   ---------------------------------------------------
//  | Step-2  Identifying minimum conflicting constraint |
//   ---------------------------------------------------
//  Following atpg constraints conflict with the rest of constraints:
//    atpg constraint: 1 dynamic  /FF3/Q (10)
//   ------------------------------------------------------------------
//  | Step-3  Identifying conflict source of first problem constraint |
//   ------------------------------------------------------------------
//  Following constraint conflicts with problem constraint #1:
//    atpg constraint: 1 dynamic  /FF1/Q (8)
```

Example 2

The following IDDQ example analyzes a failed ATPG run, displays the results of the analysis, and automatically adds an Iddq exception for each ZVAL gate that failed in an "-int_float" Iddq check. Only part of the report, about Iddq failures, is shown.

analyze_restrictions -int_float_auto_except

```
...
//  With Iddq constraints and Iddq checks removed, PASSES ability to
//  satisfy restrictions.
//  Problem is with Iddq specific restrictions. Analyzing further to find
//  problems.
//  With Iddq constraints removed, and all Set Iddq Check options removed
//  except -intfloat, FAILS.
//  Must fix -intfloat option problems to generate ANY iddq test given
//  current restrictions & circuit.
//  Will attempt to locate and report specific -int_float violators.
//  No single bus is constrained to Z (or already Z).  Will analyze
//  further.
//  The following ZVAL gates cause -int_float to fail by themselves.
//    ZVAL gate /cpu/alu/U2392/ (54417).  ABORTed.
//    ZVAL gate /cpu/alu/U4177/ (63712).  ABORTed.
//  Will add_iddq_exceptions for -int_float failure sites.
...
```

Related Topics

add_iddq_exceptions

analyze_atpg_constraints

delete_iddq_exceptions

report_iddq_exceptions

set_contention_check

set_iddq_checks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_scan_chains

Context: dft -scan

Mode: analysis

Distributes scan elements into new scan chains.

Usage

analyze_scan_chains [-force]

Description

This command distributes scan elements into new scan chains; scan elements identified as non-scannable are not considered for distribution. The scan modes and chain families defined using the add_scan_mode and create_scan_chain_family commands respectively control how the allocation of new chains is done. If not explicitly specified, the tool automatically defines a default unwrapped scan mode.

When allocating new scan chains, this command considers various scan element attributes, such as “is_non_scannable”, “clock_domain”, “clock_edge”, “class”, “cluster_name”, and “power_domain_name” along with implicit/explicit distribution constraints imposed by the set_scan_insertion_options, add_scan_mode, or create_scan_chain_family commands.

The resulting scan chain objects are themselves scan elements that can be introspected; they do not have their “exists” attribute set as they are not implemented in the netlist yet. These scan chains contain the distributed scan elements, but not the final required pipelining, retiming, or mode-enabling logic — that gets added when the insert_test_logic is invoked, once the distribution model is acceptable to the user.

Ensure that you specify the following commands in the following order:

1. Test point identification and analysis: analyze_test_points
2. X-bounding: analyze_xbounding
3. Wrapper analysis (as needed): analyze_wrapper_cells
4. Scan insertion: analyze_scan_chains

Arguments

- -force

  When no scan modes have been defined, this optional switch deletes the results of previous wrapper cell analysis and infers a default unwrapped scan mode.

Examples

analyze_scan_chains

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Chain allocation of 'int_test' mode completed:
//        14 distributed chains of sizes ranging from 12 to 13
//  Chain allocation of 'ext_test' mode completed:
//         5 distributed chains of sizes ranging from 10 to 13
...
```

report_scan_chains

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_simulation_mismatches

Context: patterns -scan

Mode: analysis

Analyzes simulation mismatches that occur between test patterns and a parallel testbench simulation performed by an external timing simulator.

Usage

analyze_simulation_mismatches {
-AUto
| {**-FAilure_file** ***failure_file*** [**-VCD_file** ***vcd_filename***]}
| {**-MISmatches** ***mismatch_bit***...}
| ***Verilog_testbench_pathname*** }
{ [-SImulation_script *simulator_invoke_script*] | [-MAnual] }
[-Internal_patterns | -External_patterns]
[-SCan_test | -CHain_test | -ALl_test]
[-HOST *server_name*] [-Lib *lib_name*]
[-TEstbench_for_vcd *new_filename* [-REplace]]
[-SHift_check {ON | OFF}] [-VErbose]

Description

By default, the tool analyzes the simulation VCD file, test patterns, and design data to identify the mismatch sources. Once the analysis is complete, Tessent Visualizer can graphically display the overlapping data to help you pinpoint the source of each mismatch.

Note

The analyze_simulation_mismatches command supports parallel testbenches and fail files generated by parallel testbenches only. It does not support serial testbenches and files from them.

Refer to "Potential Causes of Simulation Mismatches" in the *Tessent Scan and ATPG User's Manual* for complete information.

Note

All generated files are placed in the *work_dft_debug* directory inside your working directory. This directory is created if it does not already exist.

When using the analyze_simulation_mismatches command for a pattern with a P53 violation, the tool reports an error and stops the DRC checking:

```
// Error: P53 violation may produce incorrect "analyze simulation
// mismatches" result.
// Suggest to fix the test procedure timeplate to avoid P53.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The P53 error is due to an undefined clock waveform in the timeplate. Refer to "P53" on page 3490 for additional information on resolving this issue.

Note

The timestamp where the simulation actually fails does not match what this command reports, because the time analyze_simulation_mismatches command reports includes test setup to the beginning of the failing cycle.

Arguments

- **-AUto**

  Required switch that automatically runs the entire simulation mismatch analysis on the currently loaded test patterns and design. This is the default behavior.

  The top-level design and Verilog library must be compiled in your work directory prior to starting the simulation mismatch analysis. The tool searches for a "work" directory within the current working directory for these compiled files. Use the -Lib switch to specify a different directory.

  The tool uses Questa Advanced Simulator as the Verilog simulator unless you specify a different simulator with the -Simulation_script switch.

- **-FAilure_file *failure_file***

  Required switch and string pair that specifies the pathname to an existing failure file to use for the simulation mismatch analysis. Using this switch bypasses Stages 1 and 2 of the analysis.

  -VCD_file ***vcd_filename***

  Optional switch and string that specifies the pathname of an existing simulation results VCD file for the mismatch analysis. This switch can only be used with the -FAilure_file switch. Using this switch bypasses the debug testbench simulation performed in Stage 4.

- **-MISmatches *mismatch_bit...***

  Required switch and repeatable string that specifies a list of mismatches to analyze instead of using a failure file.

  A *mismatch_bit* should be specified in one of the following formats:

  fail_pattern_index chain_name cell_index

  For example: "1 chain5 20" specifies the mismatch is on scan pattern 1, scan chain5, scan cell index 20.

  fail_pattern_index PO_name

  For example: "2 xyz" specifies the mismatch is on pattern 2 at PO signal xyz.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ***Verilog_testbench_pathname***

  A required string that specifies the pathname of an existing Verilog testbench for the invocation of the old mismatch analysis. The tool performs an interactive communication with QuestaSim instead of dumping VCD data. This option only supports the Questa Advanced Simulator.

- -SImulation_script *simulator_invoke_script*

  An optional switch and string pair that specifies a simulation invocation script for the mismatch analysis. The script must invoke an external simulator and compile and simulate the testbench specified by the -PAtterns switch. By default, the analysis uses the simulator specified by the set_external_simulator command.

- -MAnual

  An optional switch to specify the "analyze simulation mismatch" functionality to stop at the step when the external Verilog simulator is needed. This option is useful when the Verilog simulation environment is complex and the simulation script cannot be written directly. When specifying the step, the tool stops after saving Verilog testbench and gives an instruction for what to do next to complete the mismatch debugging flow. See Example 6.

  Note

  Note that the -MAnual and -SImulation_script switches are mutual exclusive.

- -Internal_patterns

  An optional switch that specifies the current internal pattern set for the mismatch analysis. This is the default.

- -External_patterns

  An optional switch that specifies the currently loaded external pattern set for the mismatch analysis.

- -SCan_test

  An optional switch that specifies that the tool only analyzes scan patterns. This is the default.

- -CHain_test

  An optional switch that specifies that the tool only analyzes chain patterns.

- -ALl_test

  An optional switch that specifies that the tool analyzes both scan and chain patterns.

- -HOST *server_name*

  An optional switch and string that specifies the name of a remote host on which to run the simulation portions of the mismatch analysis. The specified server must be accessible via rsh. For more information on rsh requirements, see "Multiprocessing Requirements" in the *Tessent Scan and ATPG User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Lib *lib_name*

  An optional switch and string pair that specifies the pathname of the working simulation library. The working simulation library contains the compiled top-level design and any required parts libraries. By default, the tool uses the library named *work* in the same directory as the pattern testbench. For QuestaSim applications, you must use this switch to specify a different library whether or not you enter the vlib command.

- -TEstbench_for_vcd *new_filename*

  An optional switch and string that specifies a filename for the debug testbench created in Stage 3 (default name *tessent_default.v_vcdtb.v*). The name of the simulation results VCD file is derived from the name specified by this switch. The default VCD dump filename created from the debug testbench is *tessent_default.v_debug.vcd*

- -REplace

  An optional switch that overwrites the *debug_testbench_filename* file if it already exists.

- -SHift_check {ON| OFF}

  An optional switch and literal that enables or disables checking for simulation mismatches that occur during the scan shift cycles. By default, the tool performs shift checking, but you might want to disable this feature for performance reasons after you are sure that shift operation is working properly.

  ON — Checks and reports if the scan shift cycle contains a mismatch between the Verilog simulation value and the ATPG expected value.

  OFF — Analyzes mismatches only on the capture cycles.

- -VErbose

  An optional switch that displays additional information about the simulation commands run in the background and the detailed mismatch data.

Examples

Example 1

The following example assumes the pathname of the QuestaSim bin directory is defined by the PATH variable and that patterns are generated but not yet saved. This example demonstrates the automatic simulation mismatch analysis flow—see "Automatically Analyzing Simulation Mismatches" in the *Tessent Scan and ATPG User's Manual*.

The example first specifies the invocation command for the external simulator, then saves the generated pattern set. Next, it creates a Verilog library named *my_mismatch_work,* compiles the source for the top-level design and supporting simulation parts library into the library, and analyzes the simulation mismatches. Tessent Visualizer is invoked and displays the simulation, test pattern, and design data with the mismatch sources highlighted in red.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 3-29. analyze_simulation_mismatches Example

```
QuestaSim setting   =
    vsim -c +nowarnTSCALE -voptargs="+acc" +notimingcheck +nospecify
//  command: set_external_simulator vsim +nospecify +notimingcheck
//  command: vlib results/work
//  command: vlog -work results/work data/circle_mscan.v -v data/verilog_lib.v
QuestaSim-64 vlog 2020.4 Compiler 2020.10 Oct 13 2020
Start time: 15:26:59 on Apr 12,2021
vlog -work results/work data/circle_mscan.v -v data/verilog_lib.v
-- Compiling module DFFS_scan
-- Compiling module circle
-- Scanning library file 'data/verilog_lib.v'
-- Compiling module xor3
-- Compiling module oai21
-- Compiling module mux21
-- Compiling module lat
-- Compiling module or3
-- Compiling module xor2
-- Compiling module or4
-- Compiling module dfscrs
-- Compiling module aoi32
-- Compiling module or2

Top level modules:
    circle
End time: 15:27:00 on Apr 12,2021, Elapsed time: 0:00:01
Errors: 0, Warnings: 0
//  command: read_procfile data/new.testproc
//  command: write_patterns results/pat_p.v -verilog -replace
//  command: write_patterns results/pat_s.v -verilog -serial -replace
//  command: write_patterns results/pat.ascii -rep
//  command: analyze_simulation_mismatches -lib results/work
//  ------------------------------------------------------------------------------------
//  Step-1  Creating Verilog testbench work_dft_debug/tessent_default.v with failure file dump.
//  ------------------------------------------------------------------------------------
//  Step-2  Running Verilog testbench work_dft_debug/tessent_default.v to create failure file.
//  ------------------------------------------------------------------------------------
//  Step-3  Loading failure file work_dft_debug/tessent_default.v.fail.
//  ------------------------------------------------------------------------------------
//  Step-4  Creating Verilog testbench work_dft_debug/tessent_default.v_vcdtb.v with VCD dump.
//  ------------------------------------------------------------------------------------
//  Step-5  Running Verilog testbench work_dft_debug/tessent_default.v_vcdtb.v to create VCD file.
//          Running QuestaSim to create VCD file work_dft_debug/tessent_default.v_debug.vcd
//  ------------------------------------------------------------------------------------
//  Step-6  Loading VCD file work_dft_debug/tessent_default.v_debug.vcd.
//  ------------------------------------------------------------------------------------
//  Step-7  Analyzing 20 internal scan patterns in parallel format.
//  ----  ------------------  ----------  ---------  -------------  --------  -------  ------------------------------
//    ID  instance (gate_id)  pattern-ix  time (ns)  capture_cycle  ATPG_val  VCD_val  mismatch_reason
//  ----  ------------------  ----------  ---------  -------------  --------  -------  ------------------------------
//     1  '/ix1286/Y' (337)   scan-6           1900        0            1        0     incorrect library cell "and4"
//     2  '/ix1278/Y' (238)   scan-15         12900        0            1        0     incorrect library cell "and4"
//  command: exit -d
```

Example 2

This example demonstrates using the -failure_file switch when you have previously done Verilog simulation. This example assumes you have run an initial simulation to verify test patterns, generated a failure file, and created a script to set up and run an external simulator on a separate server from the ATPG session. The following example runs a simulation mismatch analysis on the specified failure file using internal test patterns on a separate server.

analyze_simulation_mismatches -failure_file pat.fs.v.fail -simulation_script runsim

```
// Running script "runsim pat.fs.v.fail_vcdtb.v" from local host to create VCD file pat.fs.v.fail_debug.vcd.
// Debug 24 internal patterns, CPU time = 0.02 sec
...
//  Step-5  Analyzing 24 internal scan patterns in parallel format.
//  ----  ------------------  ----------  ---------  ---------  --------  -------  ------------------------------
//    ID  instance (gate_id)  pattern-ix  time (ns)  seq_depth  ATPG_val  VCD_val  mismatch_reason
//  ----  ------------------  ----------  ---------  ---------  --------  -------  ------------------------------
//     1  '/ix1286/Y' (364)   scan-1           1900      0          1        0     incorrect library cell "and4"
//     2  '/ix1278/Y' (265)   scan-15         17300      0          1        0     incorrect library cell "and4"
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For information on the output of this command, see the report_mismatch_sources command.

Example 3

This example shows the results of using the -failure_file and -vcd_file switches and shows a simulation script “runsim” for QuestaSim:

```
#! /bin/csh -f
#Expect netlist has been compiled to work directory, next line commented
#vlog ~design/netlist.v -v ~/library/lib.v
vlog ${1}
vsim ${2} -c -do "run -all" -sdfmax /${2}/design_inst=~/design/good.sdf
```

Note that in the previous example, ${1} is the Verilog testbench name. The simulation script is used twice, first for running the testbench with the initial pattern set to get the failing bits, and then a second run for the reduced testbench to get the VCD file. Therefore, the vlog script needs to take a parameter instead of fixed testbench name. The ${2} is the Verilog testbench top instance name, which may change when the testbench name changes, so the name needs to be a parameter rather than a fixed name.

The following example uses the “runsim” script for automatic simulation mismatch analysis. Assume that the design and Verilog library have been compiled in advance in the work directory. After creating patterns, run the following command:

analyze_simulation_mismatches -simulation_script runsim

```
//  Running script "runsim work_dft_debug/tessent_default.v_vcdtb.v" from local host to create VCD file
work_dft_debug/tessent_default.v_debug.vcd.
...
//  ------------------------------------------------------------------------------------------
//  Step-5  Analyzing 24 internal scan patterns in parallel format.
//  ----  ------------------  ----------  ---------  ---------  --------  -------  ----------------------------
//    ID  instance (gate_id)  pattern-ix  time (ns)  seq_depth  ATPG_val  VCD_val  mismatch_reason
//  ----  ------------------  ----------  ---------  ---------  --------  -------  ----------------------------
//     1  '/ix1286/Y' (364)   scan-1           1900      0          1        0     incorrect library cell "and4"
//     2  '/ix1278/Y' (265)   scan-15         17300      0          1        0     incorrect library cell "and4"
```

Example 4

The following example shows a simulation script for NCVerilog:

```
#! /bin/csh -f
ncsdfc ../data/good.sdf
#Expect netlist to be compiled, next line commented out
#ncvlog ~/design/netlist.v -v ~/library/lib.v
ncvlog ${1}

#The following two commands create sdfcmdfile for ncelab to load SDF file
echo 'COMPILED_SDF_FILE = "~/design/top.sdf.X",' >! sdfcmdfile
echo "SCOPE = ${2}.design_inst;" >> sdfcmdfile
ncelab worklib.${2}:module -sdf_cmd_file sdfcmdfile
ncsim worklib.${2}:module
```

Example 5

The following example shows how to run a simulation analysis on a remote host.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_simulation_mismatches -auto -host flea01 -simulation_script runsim

```
//  Simulation mismatch source is deleted.
//  Running script "runsim tessent_default.v_vcdtb.v" from host01 to create VCD file tessent_default.v_debug.vcd.
//  Debug 25 internal patterns, CPU time = 0.02 sec
...
//  ----  ------------------  ----------  ---------  ---------  --------  -------  ----------------------------
//    ID  instance (gate_id)  pattern-ix  time (ns)  seq_depth  ATPG_val  VCD_val  mismatch_reason
//  ----  ------------------  ----------  ---------  ---------  --------  -------  ----------------------------
//     1  '/ix1286/Y' (364)   scan-1           1900      0          1        0     incorrect library cell "and4"
//     2  '/ix1278/Y' (265)   scan-15         17300      0          1        0     incorrect library cell "and4"
```

Example 6

The following example shows using the -MAnual switch:

analyze_simulation_mismatches

```
// ----------------------------------------------------------------------
// Step-1  Creating Verilog testbench work_dft_debug/tessent_default.v
// with failure file dump.
// ----------------------------------------------------------------------
// Manual flow is specified by user. Please simulate Verilog testbench
// work_dft_debug/tessent_default.v to create failure file
// work_dft_debug/tessent_default.v.fail.
// Then, "analyze simulation mismatch -fail
// work_dft_debug/tessent_default.v.fail" to generate reduced testbench
// with VCD dump.
```

Upon completion of this step, take the Verilog testbench *work_dft_debug/tessent_default.v* and perform the Verilog simulation to product the failure file *tessent_default.v.fail*.

Then, you run the following command:

analyze_simulation_mismatches -fail work_dft_debug/tessent_default.v.fail -manual

```
// ----------------------------------------------------------------------
// Step-1  Loading failure file work_dft_debug/tessent_default.v.fail.
// ----------------------------------------------------------------------
// Step-2  Creating Verilog testbench
// work_dft_debug/tessent_default.v.fail_vcdtb.v with VCD dump.
// ----------------------------------------------------------------------
// Manual flow is specified by user. Please simulate Verilog testbench
// work_dft_debug/tessent_default.v.fail_vcdtb.v to create VCD file
// work_dft_debug/tessent_default.v.fail_debug.vcd.
// Then, "analyze simulation mismatch -fail
// work_dft_debug/tessent_default.v.fail -vcd
// work_dft_debug/tessent_default.v.fail_debug.vcd" to complete the
// analysis.
```

After this step, take the Verilog testbench *work_dft_debug/tessent_default.v.fail_vcdtb.v* and perform the Verilog simulation to product the VCD dump file *work_dft_debug/ tessent_default.v.fail_debug.vcd*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Finally, you use the following command to complete the task:

```
// ------------------------------------------------------------------------------------
// Step-1  Loading failure file work_dft_debug/tessent_default.v.fail.
// ------------------------------------------------------------------------------------
// Step-2  Loading VCD file work_dft_debug/tessent_default.v_debug.vcd.
// ------------------------------------------------------------------------------------
// Step-3  Analyzing 2 internal scan patterns in parallel format.
//  ----  ------------------  ----------  ---------  ---------  --------  -------  ------------------------------
//    ID  instance (gate_id)  pattern-ix  time (ns)  seq_depth  ATPG_val  VCD_val  mismatch_reason
//  ----  ------------------  ----------  ---------  ---------  --------  -------  ------------------------------
//     1  */u102/*    (124)   scan-10          2100      0          1        0     incorrect loading during the main shifts
```

Related Topics

report_external_simulator

report_measure_cycles

report_mismatch_sources

write_patterns

set_external_simulator

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_test_points

Context: dft -test_points

Mode: analysis

Performs test point analysis and generates a list of test points for the tool to insert.

Usage

analyze_test_points [-coverage_estimation_effort {high | low}]

Description

Test point analysis can help you in the following ways:

- Improve LBIST random test coverage
- Improve ATPG test coverage
- Reduce ATPG pattern counts

For more information, see Test Point Analysis and Insertion in the *Tessent Scan and ATPG User's Manual*.

You choose the type of test point analysis performed using the set_test_point_types command, which defaults to a hybrid of LBIST coverage and EDT pattern count. Use the set_test_point_analysis_options command to specify the analysis performed by analyze_test_points.

You can use the test point analysis capability to identify LBIST test points that improve the fault coverage of pseudo-random patterns. By default, the tool bases this analysis on the entire population of stuck-at faults within the design. Alternatively, the LBIST test point analysis algorithm can identify test point locations that may improve fault coverage of deterministic patterns. For example, you could load the list of faults left undetected (AU, UC, UO, PT or PU) after performing ATPG. Targeting these fault locations identifies test points that might enable the tool to create ATPG patterns that successfully target these faults. For more information, see "Test Points for Improving the Test Coverage of Deterministic Patterns" in the *Tessent Scan and ATPG User's Manual*.

Note

The tool does not consider all gates in the design for inserting test points. It does not insert test points on gates in the clock tree or in the scan path, gates in protected blocks, or gates where you explicitly specified that no test points should be inserted. If there are many locations where test points cannot be inserted, the tool may not be able to find the best locations for test points. This may increase the number of test points needed to reach the fault coverage target. Therefore, if test points cannot be inserted at more than 20 percent of the gate-pins, the tool generates a warning. The warning specifies the number of these locations and the main reasons why test points cannot be inserted at these locations.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

To manually add test points before performing the test point analysis, use the add_control_points and the add_observe_points commands. You must do this prior to running the analyze_test_points command. Use the delete_test_points command to remove these manually added test points if you do not want the tool to consider these test points during analysis.

Note

After you have generated a list of test points with the analyze_test_points command, you must delete these test points before you can modify the locations where test points are permitted. Delete any existing test points before running the add_notest_points and delete_notest_points commands.

You must run the following commands in the following order:

1. Test point identification and analysis: analyze_test_points
2. X-bounding: analyze_xbounding
3. Wrapper analysis (as needed): analyze_wrapper_cells
4. Scan insertion: analyze_scan_chains

Arguments

- -coverage_estimation_effort {high | low}

  An optional switch and literal pair that specifies the type of algorithms and the amount of effort the tool applies to performing test point analysis. This option is only valid for test point types “lbist_test_coverage” or the hybrid “lbist_test_coverage edt_pattern_count”.

  high— the tool emulates the test points and performs fault simulation to determine the achievable test coverage. You must provide a named capture procedure (NCP) to the ATPG tool to get accurate test coverage estimates. You can use distributed processing to reduce the runtime. For more information, see “Multiprocessing for ATPG and Simulation” in the *Tessent Scan and ATPG User’s Manual*.

  low— the tool estimates test coverage based on structural analysis of the design. This is the default.

Examples

Example 1

The following example specifies that a maximum number of 50 test points are to be generated for a design that is tested with a set of 30000 random patterns. Then it performs the test points analysis based on these specifications. After the test points are reported to the screen they are inserted into the netlist:

```
set_system_mode analysis
set_test_point_analysis_options -total_number 50 -pattern_count_target 30000
analyze_test_points
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_test_points

insert_test_logic

Example 2

The following example generates 50 test points and performs the test point analysis. The test points are reported to the screen but are then deleted without being inserted as they were only generated for reporting purposes. It then sets the attribute no_control_point to true for block1 and again performs test point analysis. After the test points are reported to the screen they are inserted into the netlist.

```
set_system_mode analysis
set_test_point_analysis_options -total_number 50 -pattern_count_target 30000
analyze_test_points
report_test_points
delete_test_points -all
set_attribute_value block1 -name no_control_point -value true
analyze_test_points
report_test_points
insert_test_logic
```

Example 3

The following example generates ATPG test points to reduce deterministic pattern counts:

```
SETUP> set_test_point_type  edt_pattern_count
SETUP> set_system_mode analysis
ANALYSIS> set_test_point_analysis_options -total_number 1000
ANALYSIS> analyze_test_points
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//
//  Incremental Test Point Analysis
//  ----------------------------------------
//  TPs  100 =    0 (CP) +  100 (OP),
//  TPs  200 =    0 (CP) +  200 (OP),
//  TPs  300 =    0 (CP) +  300 (OP),
//  TPs  400 =    0 (CP) +  400 (OP),
//  TPs  500 =    0 (CP) +  500 (OP),
//  TPs  600 =    100 (CP) +  500 (OP),
//  TPs  700 =    200 (CP) +  500 (OP),
//  TPs  800 =    300 (CP) +  500 (OP),
//  TPs  900 =    400 (CP) +  500 (OP),
//  TPs 1000 =    500 (CP) +  500 (OP),
//  TPs 1000 =    500 (CP) +  500 (OP),
//
//  Performing test point optimization...
//
//  Test point analysis completed: specified number of test points has
//  been identified.
//  Total number of test points        1000
//    Control Points                   500
//    Observe Points                   500
//    Maximum control point per path   5
//    CPU_time (secs)                  51.6
//
```

Example 4

The following example generates LBIST test points to improve test coverage:

```
SETUP> set_test_point_type lbist_test_coverage
SETUP>set_system_mode analysis

ANALYSIS> set_test_point_analysis_options -total_number 1000
ANALYSIS> analyze_test_points
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Identifying locations of X-bounding muxes prior to test point
//  analysis.
//
//  Test Coverage Report before Test Point Analysis
//  ----------------------------------------------
//  Target number of random patterns           100000
//
//  Total Number of Faults                     379416
//    Testable Faults                          339781  (  89.55%)
//      Logic Bist Testable                    319101  (  84.10%)
//      Blocked by xbounding                    11597  (   3.06%)
//      Uncontrollable/Unobservable              9083  (   2.39%)
//
//  Estimated Maximum Test Coverage                        93.91%
//  Estimated Test Coverage (pre test points)              82.67%
//  Estimated Relevant Test Coverage (pre test points)     88.03%
//
//
//  Incremental Relevant Test Coverage Report
//  -----------------------------------------
//  TPs   50 =   49 (CP) +    1 (OP), TC 91.24
//  TPs  100 =   93 (CP) +    7 (OP), TC 93.71
//  TPs  150 =  141 (CP) +    9 (OP), TC 94.88
//  TPs  200 =  172 (CP) +   28 (OP), TC 95.18
//  TPs  250 =  212 (CP) +   38 (OP), TC 95.59
//  TPs  300 =  262 (CP) +   38 (OP), TC 95.99
//  TPs  350 =  308 (CP) +   42 (OP), TC 96.42
//  TPs  400 =  352 (CP) +   48 (OP), TC 96.99
//  TPs  450 =  392 (CP) +   58 (OP), TC 97.49
//  TPs  500 =  440 (CP) +   60 (OP), TC 97.97
//  TPs  550 =  475 (CP) +   75 (OP), TC 98.55
//  TPs  600 =  500 (CP) +  100 (OP), TC 98.84
//
//  Test point analysis completed: target estimated test coverage has been
//  achieved.
//  Inserted Test Points                           629
//    Control Points                               510
//    Observe Points                               119
//
//
//  Test Coverage Report after Test Point Analysis
//  ----------------------------------------------
//  Target number of random patterns           100000
//
//  Total Number of Faults                     380350
//    Testable Faults                          340715  (  89.58%)
//      Logic Bist Testable                    320035  (  84.14%)
//      Blocked by xbounding                    11597  (   3.05%)
//      Uncontrollable/Unobservable              9083  (   2.39%)
//
//  Estimated Test Coverage (post test points)             92.99%
//  Estimated Relevant Test Coverage (post test points)    99.00%
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 5

The following example uses the "-coverage_estimation_effort high" switch for an accurate estimate of test coverage:

ANALYSIS> read_procfile capture.proc
ANALYSIS> add_processors localhost:1
ANALYSIS> analyze_test_points –coverage_estimation_effort high

```
// Test Coverage Report after Test Point Analysis
    (coverage_estimation_effort high)
// ---------------------------------------------
// Target number of random patterns            10000
//
// Total Number of Faults                    4412294
//    Testable Faults                         4392962  (  99.56%)
//      Logic Bist Testable                   4321504  (  97.94%)
//      Blocked by xbounding                    68743  (   1.56%)
//      Uncontrollable/Unobservable              2715  (   0.06%)
//
// Start simulating 10000 random patterns, emulating 3000 test points
      to get more accurate Test Coverage estimation...
//
// Estimated Test Coverage by emulating random patterns         91.29%
// Estimated Relevant Test Coverage by emulating random patterns  91.29%
//
// Test point analysis completed: specified number of test points has been
     identified.
// Total requested test points              3000
// Total inserted test points               3000
//    Control Points                          920
//    Observe Points                         2080
//    Maximum control point per path            5
//    CPU_time (secs)                      1476.8
//
```

The "analyze_test_points –coverage_estimation_effort high" command provided an estimated test coverage of 91.29% by emulating random patterns. This estimate is close to the 91.16% test coverage reported after simulating 10K random patterns in "pattern -scan" context on this testcase.

Related Topics

add_observe_points

delete_test_points

insert_test_logic

report_test_points

set_test_point_analysis_options

set_test_point_insertion_options

set_test_point_types

write_scan_setup

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_test_point_dofile

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_wrapper_cells

Context: dft -scan

Mode: analysis

Identifies shared and dedicated wrapper cells for the primary I/O ports.

Usage

analyze_wrapper_cells [-force | -reset] [-verbose]

Description

This command launches the identification of the shared wrapper cells for the primary I/O ports that can be controlled with the set_wrapper_analysis_options command. The user-specified clocks and scan-related I/O ports are automatically excluded from identification. The wrapper cells identified for the primary I/O ports are stitched into the wrapper chains. If the set_wrapper_analysis_options command was not run, the analyze_wrapper_cells command performs the default wrapper cells identification similar to running the set_wrapper_analysis_options command with no arguments.

When the analyze_wrapper_cells command is used in conjunction with the set_dedicated_wrapper_cell_options command, the wrapper cells analysis results in registration of the specified primary I/O ports or the primary I/O ports that have failed the shared wrapper cells identification with the dedicated wrapper cells to guarantee better test coverage of the circuit. This means that new scan cells are added to all specified and failed primary I/O ports during scan insertion, subject to certain design considerations. The dedicated wrapper cells added to the primary I/Os are stitched into the wrapper chains. Note that if there are pad cells on primary ports, the tool inserts dedicated wrapper cells at the core side of the pad cells.

When you use the set_wrapper_analysis_options command, with or without the set_dedicated_wrapper_cell_options command, the identification includes the identification of existing sequential elements accessible via combinational logic from primary inputs or primary outputs as input/output registration cells (that is, shared wrapper cells). Thus, a shared wrapper cell serves a dual purpose of both functional mode, as well as the control and observe function during test. The advantage of shared wrapper cells is that less logic must be added and the external test mode (ExTest) can test the timing of the actual path.

Analyze_wrapper_cells may invalidate data from other analysis stages. Use this command before any scan chain analysis. If you have already completed scan chain analysis when you enter the analyze_wrapper_cells command, it results in an error. You can use the -force switch to override it, in which case the results of scan chain analysis are reset, and you must repeat it to recreate the scan modes and scan chain families you want.

The report_wrapper_cells command reports information about the identified wrapper cells for each primary I/O port subjected to wrapper cell identification.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Ensure that you specify the following commands in the following order:

1. Test point identification and analysis: analyze_test_points
2. X-bounding: analyze_xbounding
3. Wrapper analysis (as needed): analyze_wrapper_cells
4. Scan insertion: analyze_scan_chains

Arguments

- -force

  An optional switch that directs the tool to forcefully run the analyze_wrapper_cells command. To avoid using invalidated data, the tool deletes all the scan modes and scan chain families. You must recreate the scan modes and scan chain families as needed after you run the analyze_wrapper_cells command.

- -reset

  An optional switch that deletes the results of wrapper analysis, resets the wrapper_type attributes and removes information about dedicated wrapper cells

- -verbose

  An optional switch that specifies to print additional information on the progress of wrapper cell analysis.

Examples

Example 1

The following example illustrates the use of the analyze_wrapper_cells command:

set_dedicated_wrapper_cell_options on -ports out3
analyze_wrapper_cells

```
Port information and user constraints:
--------------------------------------
Input ports:      4 total,     1 ignored,     0 excluded,     0 off,     0 on,     3 auto
Output ports:     3 total,     0 ignored,     0 excluded,     0 off,     1 on,     2 auto

Wrapper analysis summary:
   -------------------------
      1 output port required a dedicated wrapper cell.
      1 input port required a dedicated wrapper cell.
      1 flip-flop was converted into an output shared wrapper cell.
      3 flip-flops were converted into input shared wrapper cells.
      Use report_wrapper_cells for more details.
```

report_wrapper_cells -verbose

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
-----------------------------------------------------------------------------------------
Primary I/O  Max Logic  #Shared Wrapper #Internal  Wrapper     Clock   Inferring  Inferring
Port         Level      Cells            Feedback  Chain Type          Dedicated  Reason
                        Direct/Internal  Gates                         Wrapper
                        -Feedback)                                     Cell
                        Identified
                        [32/32]
-----------------------------------------------------------------------------------------
'out1' (O)   0            1   'flop4'               Output     'clk'    No   --
'out2' (O)   0            0   /dow_10sffp1_i (new)  Output     'clk'    Yes  Max Logic Level
'out3' (O)   0            0   /dow_8sffp1_i (new)   Output     'clk'    Yes  --
'in2'  (I)   0            1/0 'flop3'               Input      'clk'    No   --
'in1'  (I)   0            2/0 'flop1'               Input      'clk'    No   --
                              'flop2'               Input      'clk'
'in3'  (I)   0             0   /diw_18sffp1_i (new) Input      'clk'    Yes Max Logic Level
-----------  ---------  -------------------------  ----------  --------------------  -----
The wrapper cells identification has failed for the following PIs and POs.
These PIs and POs will be I/O Registered unless excluded from the registration.
-----------------------------------------------------------------------------------------
'in3'        Input     Exceeded the limit, 1, on the number of combinational logic levels
                       between this PI and the first level of sequential cells'out2'
             Output    Exceeded the limit, 1, on the number of combinational logic levels
                       between this PO and the first level of sequential cells
-----------------------------------------------------------------------------------------
These PIs and POs will be excluded from the I/O Registration unless they were explicitly
included.
-----------------------------------------------------------------------------------------
'clk'                            Input                            Clock port
-----------------------------------------------------------------------------------------
```

Example 2

The following is an example of the wrapper analysis output generated when you use the -verbose switch:

analyze_wrapper_cells -verbose

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//
//     Port information and user constraints:
//     --------------------------------------
//         Input ports:   8 total,   2 ignored,   0 excluded,     0 off,     0 on,     6 auto
//        Output ports:   3 total,   0 ignored,   0 excluded,     0 off,     0 on,     3 auto
//// Identify potential output shared wrapper cells
// Tracing fan-in of output port OUT_1.
// Tracing fan-in of output port OUT_2.
...
// Tracing set/reset drivers of potential shared wrapper cells for port OUT_1
// Tracing set/reset drivers of potential shared wrapper cells for port OUT_2
..
// Process scan segments and multi-bit cells
// Identify SET/RESET ports
//
// Identify potential input shared wrapper cells
// Tracing fan-out of input port IN_1.
// Tracing fan-out of input port IN_2.
...
// Tracing feedback wrapper cells for port IN_1
// Tracing feedback wrapper cells for port IN_2
...
// Process scan segments and multi-bit cells
// Identify feedback cells for user-specified wrapper cells
// Processing clock gaters
// Wrapper analysis results post-processing
//
//     Wrapper analysis summary:
//     -------------------------
//        x flip-flops were converted into output shared wrapper cells.
//        Use report_wrapper_cells for more details.
```

Related Topics

report_wrapper_cells

set_dedicated_wrapper_cell_options

set_wrapper_analysis_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_xbounding

Context: dft -scan

Mode: analysis

Performs X-bounding analysis.

Usage

analyze_xbounding [-force]

Description

This command is used to identify memory elements (current scan cells or cells that will be converted to scan-in a subsequent run) that might capture an unknown value during Logic BIST. In addition, the command identifies where to place bounding muxes that prevent these X-sources from reaching the memory elements and corrupting the signature during Logic BIST.

The X-bounding analysis uses the E5 DRC to identify and bound all X sources from reaching scan cells and corrupting the MISR. All primary inputs and internal X source pins (for example, non scan cells, blackbox outputs pins) are x-bounded.

The X-bounding analysis uses the E9 DRC to identify wire gates that may be driving a net to opposing values (contention). The tool does not add hardware to avoid the contention, however it adds X-bounding logic to prevent observation of the wire gates with the E9 violation.

The X-bounding analysis uses the E10 DRC to detect tristate bus contention. The tool does not add any hardware to avoid the bus contention, however it adds X-bounding logic to prevent observation of the bus with the E10 violation.

The X-bounding analysis uses the E11 DRC to detect a bus that can attain a Z-state. The tool does not add any hardware to prevent the bus from reaching the Z state, however it adds X-bounding logic to prevent observation of the floating bus.

The X-bounding process depends on pin constraints on the primary input pins to indicate them as X sources. These X-constraints are implicitly enforced by this command, but can also come from existing input constraints. When a pin is explicitly constrained to either 0 or 1, it is not treated as an X source.

The mux and scan cell library models used for X-bounding can be specified using the cell_type attribute in the Tessent Cell library or by the add_cell_models command. If any of the required library cell models are not specified, an error message is reported.

The analyze_xbounding command also marks as notest points the previously observable gate pins that are on the input side of the X-bounding muxes. In addition, an X-source that does not feed into any scannable flops (for example, it simply connects to a primary output pin) is not bounded. Any previously observable gate-pins that are driven by that X-source are also marked as notest points.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

You must issue this command to perform X-bounding analysis. The tool does not automatically perform the analysis when you issue the insert_test_logic command.

Analyze_xbounding may invalidate data from other analysis stages. This command should be used before any wrapper analysis or and scan chain analysis. If you have already completed any of these steps when you issue the analyze_xbounding command, it results in an error. You can use the -force switch to override it, in which case the results of wrapper analysis and scan chain analysis are reset and you must repeat them.

The X-bounding analysis also bounds false and multi-cycle paths. This capability is triggered by reading an SDC file prior to issuing the command. The tool identifies any scannable flops that could capture an X due to a user-defined false or multi-cycle path. The tool adds X-bounding logic that blocks the path at the D-input of the destination of the flop. These flops "capture" the inverted value of the flop output so they can still trigger transitions at the output during launch-of-capture transition fault testing.

Ensure that you specify the following commands in the following order:

1. Test point identification and analysis: analyze_test_points
2. X-bounding: analyze_xbounding
3. Wrapper analysis (as needed): analyze_wrapper_cells
4. Scan insertion: analyze_scan_chains

Arguments

- -force

  An optional switch that directs the tool to forcefully issue the analyze_xbounding command. In order to avoid using invalidated data the tool resets the results of wrapper analysis and deletes all the scan modes and scan chain families before doing X-bounding. It is recommended that you re-run wrapper analysis and set the scan modes and scan chain families as needed before you issue the analyze_xbounding command.

Examples

This example transitions from setup to analysis mode, triggers the X-bounding analysis, and then reports the locations of the X-bounding muxes:

```
set_system_mode analysis
analyze_xbounding
report_xbounding -verbose
```

Related Topics

insert_test_logic

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_sdc

report_notest_points

report_xbounding

set_xbounding_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

annotate_diagnosis

Context: patterns -scan_diagnosis

Mode: analysis

Adds DFM and RCD data, if available, to existing diagnosis reports.

Usage

annotate_diagnosis ***directory_path*** [-force] [-verbose] [-report] [-regexpr *regular_expression*]

Description

You can add RCD constants to existing diagnosis reports using the annotate_diagnosis command. To annotate a diagnosis report, you must load the same pattern set that you used to run diagnosis and create RCD constants.

If there is already a DFM rule of the same name and type in the report, the tool replaces it. If there are already existing RCD feature statistics in the report, the tool replaces them.

The tool also checks the design/patterns MD5s in the ALL_FAILURE_INFO section of each diagnosis report to match the currently loaded design and patterns. Annotation fails if these do not match.

Note

The set_diagnosis_options -include_dfm_rules and -include_rcd_constants switches do not affect the annotation process. Information is always preserved or added in the annotated diagnosis reports based on the content of the VDB or LDB that is open.

Arguments

- ***directory_path***

  A required string that specifies the directory path of a diagnosis report.

- -force

  An optional switch that specifies to bypass any issues with MD5 compatibility that arise.

- -verbose

  An optional switch that specifies to report all MD5 mismatch warnings during the annotate_diagnosis run.

- -report

  An optional switch that specifies to report a cumulative summary report of MD5 mismatch warnings during the annotate_diagnosis run.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -regexpr *regular_expression*

  An optional switch that specifies to match the diagnosis reports to the specified regular_expression. If this switch is not specified, the following extensions are automatically skipped:

  swp .adb .ldb .csv .lay .json .tcl .pl .sh .csh .do .db .verify

Examples

Example 1

The following example adds the RCD constants to the failure_file_directory diagnosis report.

read_flat_model flat_model.gz

read_patterns pattern_file.gz

open_layout ldb_directory

annotate_diagnosis failure_file_directory

Example 2

The following example shows how to add RCD constants to existing diagnosis reports. Use the read_patterns command to specify the applicable pattern set.

```
# Specify the same pattern set that was used to run diagnosis
# and create RCD constants
```

read_patterns /src/patterns.ascii

```
# Open the LDB
```

open layout design.dft.ldbannotate_diagnosis small_diags_rcd_gz

Related Topics

delete_dfm

import_dfm

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

append_to_collection

Context: unspecified, all contexts

Mode: all modes

Adds one or more objects to a collection referenced by the specified ***var_name***.

Usage

append_to_collection ***var_name obj_spec*** [-unique]

Description

The append_to_collection command appends all elements in obj_spec to the collection referenced by the variable ***var_name***. If the variable does not exist, the variable is created and made to point to a collection with elements from ***obj_spec*** as its value. An error is generated if the variable exists and does not reference a collection.

The result of the append_to_collection command is either a modification of the collection that was initially referenced by ***var_name*** or a new collection if the variable did not exist.

The append_to_collection command provides the same semantics as the add_to_collection command but with significant performance improvement. If you are building up a collection in a loop, using the append_to_collection command is more efficient as shown in "Example 2" on page 489.

Arguments

- ***var_name***

  A required value that specifies a Tcl variable name. The objects matching ***obj_spec*** are added to the collection referenced by this variable.

- ***obj_spec***

  A required value that specifies a Tcl list of one or more object names or a collection of one or more objects to append to the collection referenced by ***var_name***. The ***obj_spec*** can be an empty collection.

- -unique

  An optional switch that prevents duplicate objects from being appended to the collection. By default, duplicate objects are not removed.

Examples

Example 1

The following example appends a collection referenced by variable *b* to collection referenced by variable *a*.

```
set a [get_ports a*]
set b [get_ports b*]
append_to_collection a $b
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example creates a collection of all nets connected to pins of design instances matching u1/myPrefix*. Note that using the -unique option automatically removes duplicate nets. This example is the same as the example shown for the add_to_collection command. Using the append_to_collection command for such an application is recommended and is much more efficient.

```
foreach_in_collection inst [get_instances u1/myPrefix* -of_type design] {
        set inputs [get_pins -of_instance $inst -direction input]
        set outputs [get_pins -of_instance $inst -direction output]
        append_to_collection sum [get_fanins $inputs -stop_on net] -unique
        append_to_collection sum [get_fanouts $outputs -stop_on net] -unique
}
```

Related Topics

add_to_collection

compare_collections

copy_collection

filter_collection

foreach_in_collection

index_collection

is_collection

range_collection

remove_from_collection

sizeof_collection

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

apply_specification_defaults

Context: dft, patterns

Mode: all modes

Automatically applies all defaults specified in the DefaultsSpecification/DftSpecification and DefaultsSpecification/PatternsSpecification wrappers to wrappers that were created manually.

Usage

apply_specification_defaults [*id*] [-design_name *design_name*]
[-config_object *config_object*]

Description

You must use this command to apply the defaults under the following circumstances:

- When a DftSpecification or PatternsSpecification has been created manually.
- When a DftSpecification or PatternsSpecification has been modified after the specification has been created and the new defaults should be applied.

You can also use this command when creating a DftSpecification wrapper in the Configuration Data window. You do not need this command when using the create_dft_specification command because the create_dft_specification command automatically applies the specified defaults while creating the DftSpecification wrapper.

Similarly, you can also use this command when creating a PatternsSpecification wrapper in the Configuration Data window. You do not need this command when using the create_patterns_specification command because the create_patterns_specification command automatically applies the specified defaults while creating the PatternsSpecification wrapper.

Note

Note that you may need to use the apply_specification_defaults command to re-apply the defaults when the specification or the “defaults” have been modified *after* the wrappers have been created.

Arguments

- *id*

  An optional string that specifies the specification wrapper that is to be processed. As described in the DftSpecification and PatternsSpecification wrapper syntax, the wrapper is identified by the two strings: *design_name* and *id*. If you only have one DftSpecification or PatternsSpecification wrapper present in memory that matches *specification*(design_name,*), you can omit the *id* from the process_dft_specification or process_patterns_specification command. However, if several DftSpecification or PatternsSpecification wrappers matching *specification*(design_name,*) exist in memory, you must specify the appropriate *id* when executing the process_dft_specification or

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

process_patterns_specification command. Refer to the -design_name option to understand how the design_name value is obtained.

- -design_name *design_name*

  An optional switch and string pair that specifies the *design_name* value used to identify the DftSpecification or PatternsSpecification wrapper. The -design_name value defaults to the current design if it has been set with the set_current_design command. If the current design has not been set, it is extracted from the DftSpecification or PatternsSpecification wrapper existing in memory.

  When the *id* is specified, if exactly one wrapper matches "*specification* (*,*id*)" in memory, the *design_name* is extracted from the design_name parameter of the DftSpecification or PatternsSpecification wrapper; if more than one wrapper matches, the -design_name must be specified.

  When the *id* is not specified, if exactly one wrapper matches "*specification* (*,*)" in memory, the design_name is extracted from the design_name parameter of the DftSpecification or PatternsSpecification wrapper; if more than one wrapper matches, the -design_name must be specified.

- -config_object *config_object*

  An optional switch and string pair that applies the defaults to only one specific DftSpecification or PatternsSpecification wrapper. By default all the specifications in memory that correspond to the selected *id* and "-design_name *design_name*" are processed.

Examples

Example 1

The following example calls the apply_specification_defaults command on a DftSpecification wrapper that was created manually to get the specified defaults found in the DefaultsSpecification/DftSpecification wrapper applied to the DftSpecification wrapper.

apply_specification_defaults

Example 2

The following example uses the apply_specification_defaults to reapply defaults to the DftSpecification after the specification has been created.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_context dft -rtl
set_tsdb_out_dir tsdb_outdir_test_read
read_verilog chip.vb
set_current_design chip
set_design_level sub_block
set_system_mode analysis
set_defaults_value \
   DftSpecification/IjtagNetwork/HostScanInterface/Chip/tck chip_tck
set_defaults_value \
   DftSpecification/IjtagNetwork/HostScanInterface/Block/tck block_tck

read_config_data -from_string {
  DftSpecification(chip,rtl) {
    IjtagNetwork {
      HostScanInterface(ijtag) {
        Tap(TAP1) {
        }
      }
    }
  }
}

apply_specification_defaults

set top [get_config_element DftSpecification]
report_config_data $top
process_dft_specification
extract_icl
```

Related Topics

create_dft_specification

create_patterns_specification

process_dft_specification

process_patterns_specification

set_defaults_value

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

assert_icl_sib

Context: patterns -silicon_insight -ijtag

Mode: setup

Configures the specified SIB to pick one of the host inputs.

Usage

assert_icl_sib ***sib_name*** [-host_index *num_host_interfaces*]

Description

Use this command when you are configuring a scan path in the IJTAG network. The specified SIB is active in the scan path.

Use this command when manually debugging IJTAG networks as described in "Diagnosing and Debugging IJTAG Networks" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

If the specified SIB has more than one host interface, you must specify the -host_index switch.

Arguments

- ***sib_name***

  A required variable the specifies the name of the SIB that you want to assert to a 1 state.

- -host_index *num_host_interfaces*

  An optional switch and variable pair that specifies the number of host interfaces the specified SIB has. The host interfaces correspond to input indices starting at the number 1. 0 is reserved for the client interface.

Examples

assert_icl_sib sib_l4

assert_icl_sib sib_l4 -host_index 0

Related Topics

analyze_icl_suspect

create_icl_flush_pattern

create_icl_setup_pattern

deassert_icl_sib

get_icl_network

get_icl_scan_mux

isolate_icl_node

report_icl_network

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_icl_network

set_icl_scan_mux

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

catch_output

Context: unspecified, all contexts

Mode: all modes

Runs the specified tool command line, and may be used to prevent command errors from aborting the enclosing dofile or Tcl proc.

Usage

catch_output ***cmdline*** [-output *output_variable*] [-result *result_variable*] [-tee]

Description

This command returns 0 if the embedded command succeeds, and 1 if it fails.

You can use the -output switch to specify a variable to contain all of the output of the embedded command. You can use the -result switch to specify a variable to contain the Tcl result of the embedded command if the command returns one.

Arguments

- ***cmdline***

  A required string enclosed in braces "{ }" that specifies the command and associated arguments whose output you want to capture. The command is typically a report_* command. A return value of 1 indicates the command failed; a return value of 0 indicates the command succeeded.

- -output *output_variable*

  An optional switch and variable name that is set to the value of the entire command output.

- -result *result_variable*

  An optional switch and variable name that specifies to write the Tcl result returned by the embedded command to the *result_variable* variable.

- -tee

  An optional switch that specifies to write the command output to the transcript and to a logfile.

Examples

Example 1

The following example writes the result of the get_gate_pins command to the variable oVar; the result can be an error message if the command fails. If get_gate_pins fails, catch_output returns 1, and the "if" wrapper commands are run.

The example also assigns the Tcl result returned by the get_gate_pins command to the rVar variable; the result is a collection if the command succeeds. If get_gate_pins succeeds, the catch_output command returns 0, and the "else" wrapper commands are run.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The dofile continues even if the get_gate_pins command fails. Nothing is reported to the transcript.

```
if {[catch_output {get_gate_pins foo*} -output oVar -result rVar]} {
  # commands when get_gate_pins fails
} else {
  # commands when get_gate_pins succeeds
}
```

Example 2

The following example uses the catch_output command to capture the output of a tool command in a variable.

```
set retVar [catch_output {report_gate /gate3} -output resultA -tee]
```

```
// /gate3  nor02
//      A0        I  /gate2/Y
//      A1        I  /ff2/Q
//       Y        O  /ff5/D  ff4/D  ff3/D  gate6b/E
```

```
If { $retVar == 0 }  {
  puts $resultA
}
```

```
// /gate3  nor02
//      A0        I  /gate2/Y
//      A1        I  /ff2/Q
//       Y        O  /ff5/D  ff4/D  ff3/D  gate6b/E
```

Related Topics

set_transcript_style

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

check_design_rules

Context: dft, patterns

Mode: setup

Transitions the tool from setup mode to analysis mode.

Usage

check_design_rules

Description

Causes the tool to transition from setup mode to analysis mode.

In RTL context, the design is synthesized in parts before creating the flat model. The DRC rules specific to the current context are then run. If no DRC with a severity of error fails, the tool enters analysis mode.

If any DRC with a severity of error fails, the tool remains in setup mode, which enables you to analyze the DRC violation and either add more definitions (such as clocks) or make edits to your circuit.

Arguments

None

Examples

The following example shows the results of calling the check_design_rules command, which generates a DRC violation. The violation is fixed by defining a missing clock. The check_design_rules command is then re-issued and successfully transitions from system mode to analysis mode.

```
SETUP> check_design_rules
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  ---------------------------------------------------------------------
//  Begin RTL synthesis.
//  --------------------
//  Module 'sub_block1' synthesized.
//  Module ’block1’ synthesized.
//  RTL synthesis completed, synthesized modules=2, Time=1.88 secs.
//  ---------------------------------------------------------------------
//  ---------------------------------------------------------------------
//  Begin RTL synthesis.
//  --------------------
//  Module 'my_clock_gate' synthesized.
//  Module 'my_inv' synthesized.
//  RTL synthesis completed, synthesized modules=2, Time=1.94 secs.
//  ---------------------------------------------------------------------
//  ---------------------------------------------------------------------
//  Begin RTL synthesis.
//  --------------------
//  Module 'my_clock_mux' synthesized.
//  RTL synthesis completed, synthesized modules=1, Time=1.84 secs.
//  ---------------------------------------------------------------------
//  Warning: Rule FP2 violation occurs 1 times
//  Flattening process completed, cell instances=82, gates=843, PIs=5,
//  POs=0, CPU time=0.01 sec.
//  ---------------------------------------------------------------------
//  Begin circuit learning analyses.
//  --------------------------------
//  Learning completed, CPU time=0.00 sec.
//  Error: The memory clock pin 'sub_block1_i1/mem1/CLK' (67.40) source
//  tracing stopped at pin 'pll/vco' (9.0).
//  Correct the blocking conditions or use the 'add_clocks -period' if
//  this is an embedded oscillator or
//  add_clocks -reference' if this is an active PLL or clock divider.
//  (DFT_C1-1)
//  Error: There was 1 DFT_C1 violation (Memory clock not properly sourced
//  by a declared clock).
```

SETUP> add_clocks pll/vco -reference pll/ref -freq_mult 10 -freq_divider 2

```
//  Warning: Flat model deleted.
```

SETUP> check_design_rules

```
//  Warning: Rule FP2 violation occurs 1 times
//  Flattening process completed, cell instances=82, gates=844, PIs=6,
//  POs=0, CPU time=0.01 sec.
//  ---------------------------------------------------------------------
//  Begin circuit learning analyses.
//  --------------------------------
//  Learning completed, CPU time=0.00 sec.
```

ANALYSIS>

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

check_synthesis

Context: all contexts

Mode: all modes

Checks the status of synthesis that was previously launched by the run_synthesis command.

Usage

check_synthesis [-design_name *design_name*] [-design_id *design_id*] [-report_status | -get_status_dictionary | -wait | -kill] [-synthesis_output_directory_list *directory_list*]

Description

Checks the status of synthesis run that was previously launched by the run_synthesis command.

Based on the specified option, you can generate a report, get a status dictionary, or get the pass or fail status.

Arguments

- -design_name *design_name*

  An optional switch and value pair that specifies the design name to which run_synthesis is associated with. By default, the name of the *current_design* is used when it is set. If you run the command immediately after running the process_dft_specification command, you do not need to specify this option.

- -design_id *design_id*

  An optional switch and value pair that specifies the *design_id* to which run_synthesis is associated with. By default, the name of the *current_design_id* is used when it is set. If you run the command immediately after running the process_dft_specification command, you do not need to specify this option.

- -report_status

  An optional switch that is mutually exclusive with the -get_status_dictionary and -wait options. When none of the these options is specified, -wait is assumed. By default, -report_status includes all the *design_id*s found in the synthesis directories specified by -synthesis_output_directory_list.

- -get_status_dictionary

  An optional switch that is mutually exclusive with -report_status and -wait. When none of these options is specified,-wait is assumed.

  The Tcl dictionary returned with -get_status_dictionary contains all the information that was gathered to produce the report generated when -report_status is used instead.

- -wait

  An optional switch that waits for synthesis task to be completed before returning with a completion message. When running in interactive mode, the command reports a status line

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

that is updated every second showing at what stage synthesis is current operating, which can be unscheduled, queued, running, pass, or fail, as shown in Example 1.

- -kill

  An optional switch that kills the selected running synthesis. The -kill switch is mutually exclusive with-report_status, -get_status_dictionary and -wait. If no switch is specified, -wait is assumed.

- -synthesis_output_directory_list *directory_list*

  An optional switch and value pair that specifies a list of directories that contain the *design_name_design_id.synthesis_synthesis_tool* directories. The default value is *./synthesis_outdir*. You can configure the default value using the "set_run_synthesis_options -synthesis_output_directory" command and option

Examples

The following examples show the results of using check_synthesis with the available options.

Example 1

SETUP> check_synthesis

```
unscheduled 0 queued 0 running 0 pass 1 fail 0
// The 17 synthesized files listed in
// './synthesis_outdir/blockB_rtl.synthesis_dc_shell/file_list' were
// copied to the current tsdb output directory.
```

Example 2

SETUP> check_synthesis -get_status_dictionary

```
// ./synthesis_outdir {blockB_rtl {status pass date {Mon Aug 24 13:02:38
// PDT 2015} sub_dir blockB_rtl.synthesis_dc_shell log_file
// dc_shell.synthesis_log errors_count 0 errors {}}}
```

Example 3

**SETUP> check_synthesis -synthesis_output_directory_list \
synthesis_outdir**

```
unscheduled 0 queued 0 running 0 pass 1 fail 0
// The 17 synthesized files listed in
// 'synthesis_outdir/blockB_rtl.synthesis_dc_shell/file_list' were copied
// to the current tsdb output directory.
```

Example 4

The following example shows check_synthesis usage to report the synthesis status for all *design_id*s in the synthesis output directories named "check_out_ut1" and "check_out_ut2" for design "chip":

**SETUP>check_synthesis -design_name chip -report \
-synthesis_output_directory_list [list check_out_ut2 check_out_ut1]**

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Synthesis status for check_out_ut2/chip_rtl.synthesis_dc_shell
// ==============================================================
// --------  ------  ------  ----------------------------
// Name      Status  Errors  Date
// --------  ------  ------  ----------------------------
// chip_rtl  pass    0       Mon Feb 08 09:05:08 PST 2016
//

// Synthesis status for:
//   (1) check_out_ut1/chip_rtl.synthesis_dc_shell
//   (2) check_out_ut1/chip_rtl2.synthesis_dc_shell
// ==============================================
// -------------  ------  ------  ----------------------------
// Name           Status  Errors  Date
// -------------  ------  ------  ----------------------------
// (1) chip_rtl   fail    3       Thu Feb 04 11:25:49 PST 2016
// (2) chip_rtl2  pass    0       Mon Feb 08 09:05:08 PST 2016
//
```

Related Topics

get_run_synthesis_options

report_run_synthesis_options

run_synthesis

set_run_synthesis_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

check_testbench_simulations

Context: dft, patterns

Mode: all

Checks the status of simulations that were previously launched by the run_testbench_simulations command.

Usage

check_testbench_simulations
[-design_name *design_name*]
[-design_id *design_id*]
[-pattern_id *pattern_id*]
[-select *pattern_list*]
[-exclude *pattern_list*]
[-report_status | -get_status_dictionary
| -kill | -wait [-keep_simulation_data {on | off | on_failure}]]
[-simulation_output_directory_list *simulation_output_directory_list*]
[-simulator {questa | vcs | incisive}]

Description

Checks the status of simulations that were previously launched by run_testbench_simulations command.

Based on the specified option, you can generate a report, get a status dictionary, or get the pass or fail status. You can also kill pending simulations or delete patterns simulation directories.

Arguments

- -design_name *design_name*

  An optional switch and value pair that specifies the design_name that the patterns to check are associated with. By default, the name of the *current_design* is used when it is set. The design_name is part of the *design_name_design_id.simulation_pattern_id* directory name that contains the simulation results.

- -design_id *design_id*

  An optional switch and value pair that specifies the *design_id* to that the patterns to check are associated with. By default, the *design_id* value set using the *design_name_design_id.simulation_pattern_id* directory name that contains the simulation results.

  A *design_id* must only use a combination of the following characters:

  - Alphabetic, both uppercase and lowercase. For example: “abc”, “ABC”, “AbC”, and so on.
  - Numeric. For example: “123”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

  - Underscore. For example: “A_1”.

- -pattern_id *pattern_id*

  An optional switch and value pair that specifies the *pattern_id* that the patterns to simulate are associated with. By default, the *pattern_id* used by the previously run process_patterns_specification command is reused automatically; otherwise, the default value of “signoff” is used.

  The *pattern_id* is part of the *design_name_design_id.simulation_pattern_id* directory name that contains the simulation results.

- -select *pattern_list*

  An optional switch and value pair that restricts the action on a list of *pattern_name.simulation[_simulation_id]* directories that match at least one of the entries in the *pattern_list*. The name patterns are simple glob patterns with * as the wild card. Refer to Example 2.

- -exclude *pattern_list*

  An optional switch and value pair that excludes a list of *pattern_name.simulation[_simulation_id]* directories that match at least one of the entries in the *pattern_list* from the action of the command. The name patterns are simple glob patterns with * as the wild card. Refer to Example 3.

- -report_status

  An optional switch that is mutually exclusive with the -get_status_dictionary, -kill, and -wait options. When none of these options are specified, -wait is assumed. When you use the -report_status switch, the command creates a report listing the simulation status of each pattern found inside the *design_name_design_id.simulation_pattern_id* directories located inside one of the *simulation_output_directory_list* directories that was simulated, is still simulating, or is waiting to be simulated. You can restrict the report can to a subset of the *pattern_name.simulation[_simulation_id]* directories by using the -select or the -exclude options.

  The format of the report is illustrated here. The date entry shows the time the current status was achieved. The missing miscompares column is for negative testing where the run_testbench_simulations -expected_miscompare_count option was used or a *simulator.simulation_log_reference* file was present in the directory.

```
// Simulation status for <simulation_output_directory>
// ==================================================
// --------------                ------        ---------       -------  ----
// Pattern Name                  Status        Unexpected      Missing  Date
// --------------                ------        ---------       -------  ----
// <pattern_name> (<sim_id>)     unscheduled
// <pattern_name> (<sim_id>)     queued
// <pattern_name> (<sim_id>)     running
// <pattern_name> (<sim_id>)     pass              0           0         Wed Oct 08
//                                                                     23:38:32 GMT 2014
// <pattern_name> (<sim_id>)     fail              2           0         Wed Oct 08
//                                                                     23:39:40 GMT 2014
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

After issuing the run_testbench_simulations command, the "check_testbench_simulation -report_status" command and switch reports the simulation status for the tool operations in the *<pattern_name>.simulation* sub-directory.

For a given simulation, the status normally changes as follows:

- unscheduled — During the compilation phase of the design, all the simulation log files under the *<pattern_name>.simulation* sub-directory are created with an empty content but with the name *<simulator>.simulation_log.unscheduled*. The tool does not queue the *<pattern_name>.simulation_script* at this point because the library of the design is not available until the compilation is completed.
- queue — The tool completes compilation and all simulation log files are renamed from *.unscheduled* to *<simulator>.simulation_log.queued*.
- running — As the simulations start running, the log file is renamed from *.queued* to *<simulator>.simulation_log.running*.

When the simulation completes, the name of the file is renamed from *.running* to *<simulator>.simulation_log*.

- -get_status_dictionary

  An optional switch that is mutually exclusive with the -report_status, -kill and -wait options. When none of those options are specified, -wait is assumed. When you use the -get_status_dictionary switch, the command returns a Tcl dictionary listing the simulation status of each pattern simulated, simulating, or waiting to be simulated and found inside the *design_name_design_id.simulationpattern_id* directories located inside one of the simulation_output_directory_list directories. You can restrict the dictionary to a subset of

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

the <pattern_name>.simulation[_simulation_id>] directories by using the -select or the -exclude options. The format of the returned dictionary is as follows:

```
<simulation_output_directory> {
  <pattern_name>[.<simulation_id>] {
    status unscheduled | queued | running | pass | fail
    date   <date>
    unexpected_miscompare_count <int>
    unexpected_miscompares {
      miscompare<int> {
        object_name     <object_name>
        step_name       <step_name>
        expected_value  <value>
        simulated_value <value>
      }
    }
    missing_expected_miscompare_count <int>
    missing_expected_miscompares {
      miscompare<int> {
        object_name     <object_name>
        step_name       <step_name>
        expected_value  <value>
        simulated_value <value>
      }
    }
  }
}
```

- -kill

  An optional switch that is mutually exclusive with the -report_status, -get_status_dictionary, and -wait options. When none of these options are specified, -wait is assumed. When you use the -kill switch, the selected running simulation will be stopped.

- -wait [-keep_simulation_data {on | off | on_failure}]

  An optional switch that is mutually exclusive with the -report_status, -kill, and -get_status_dictionary options. When none of these options are specified, -wait is assumed. When you use the -wait switch, the command waits for all the simulations to be completed before returning with an error message when any of the simulations failed.

  You can also use the optional -keep_simulation_data switch with the -wait option to choose how you deal with intermediate simulation files.

  on — All simulation intermediate files are retained.

  Note

  You can set the TESSENT_KEEP_SIMULATION_DATA environment variable to 1 to keep intermediate simulation data.

  off — All simulation data is removed, irrespective of whether the simulation passes or fails.

  on_failure — All passing simulation data is removed. Failing simulation data is retained. This is the default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When running in interactive mode, the command reports a status line that is updated every second, showing how many simulations are in the unscheduled, queued, running, pass, or fail state as shown in the following.

```
check_testbench_simulations
unscheduled <int> queued <int> running <int> pass <int> fail <int>
```

The patterns that are waited on are those simulating or waiting to be simulated, and found inside the *design_name_design_id.simulation_pattern_id* directories located inside one of the simulation_output_directory_list directories. The list can be restricted to a subset of the *pattern_name.simulation[_simulation_id]* directories by using the -select or the -exclude options. If you ran the run_testbench_simulations command using the -generate_scripts_only option or if you issued the check_testbench_simulation -kill option without rerunning the run_testbench_simulations command, you may have simulation directories that have no simulation log file and are not scheduled to run. Those should be deleted using the -delete option.

Press Ctrl-C to stop the waiting and return to the interactive prompt.

- -simulation_output_directory_list *simulation_output_directory_list*

  An optional switch and value pair that specifies a list of directories that contain the *design_name_design_id.simulation_pattern_id* directories. The default value is *./simulation_outdir*.

- -simulator questa | vcs | incisive

  An optional switch and value pair to check the status of the simulator used for the simulation that was specified with the run_testbench_simulations command.

Examples

Example 1

The following example runs the run_testbench_simulations command to launch the simulations. The check_testbench_simulation command is then run to monitor the simulations and return an error when the simulations finish and any of them failed. If an error code is trapped by the catch command, the report_status is dumped to the transcript and the tool is terminated with an error code. If no simulation failed, the string “All sims passed” is echoed in the transcript.

```
process_patterns_specification
run_testbench_simulations
if {[catch check_testbench_simulations]} {
  check_testbench_simulations -report_status
  exit -force 1
 } else {
   puts “All sims passed”
 }
```

```
// sub-command: check_testbench_simulations
// Error: 2 out of 4 simulations failed:
//        MemoryBisr_BisrChainAccess with 3 unexpected miscompares
//        MemoryBist_P2.FI with 2 missing miscompares
// sub-command: check_testbench_simulations -report_status
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
process_patterns_specification
run_testbench_simulations
if {[catch check_testbench_simulations]} {
   check_testbench_simulations -report_status
   exit -force 1
 } else {
   puts "All sims passed"
 }
```

```
// sub-command: check_testbench_simulations
// Error: 2 out of 4 simulations failed:
//        MemoryBisr_BisrChainAccess with 3 unexpected miscompares
//        MemoryBist_P2.FI with 2 missing miscompares
// sub-command: check_testbench_simulations -report_status

// Simulation status for ./simulation_outdir
// ========================================
// ------------  -------   ----------- -----------   -------------------------
// Pattern Name Status    Unexpected  Missing       Date
//                                          Miscompares Miscompares
// ------------------------------  -------   ----------- -----------   ---------------------
// MemoryBisr_BisrChainAccess fail         3        0         Wed Oct 08 23:38:32 GMT 2014
// MemoryBist_P1              pass         0        0         Wed Oct 08 23:38:35 GMT 2014
// MemoryBist_P2              pass         0        0         Wed Oct 08 23:38:39 GMT 2014
// MemoryBist_P2 (FI)         fail         0        2         Wed Oct 08 23:38:44 GMT 2014
```

Example 2

The following example demonstrates using the -select switch:

```
> check_testbench_simulations -kill -select [list MemoryBisr_BisrChainAccess MemoryBist_P1]
```

Example 3

The following example demonstrates using the -exclude and -select switch:

```
> check_testbench_simulations -kill -exclude [list *Bisr*] -select [list MemoryBis*]
```

Example 4

The following example demonstrates how to check the status of multiple simulations based on a Tcl list:

```
# Start several simulations in several different output directories:
run_testbench_simulations -no_wait -design_name mod1a -simulation_out_dir sim_outdir_1
run_testbench_simulations -no_wait -design_name mod1b -simulation_out_dir sim_outdir_1
run_testbench_simulations -no_wait -design_name mod2 -simulation_out_dir sim_outdir_2

# Check and wait on all the simulations at the same time:
check_testbench_simulations -design_name * -design_id * -sim_out_dir_list \
      [list sim_outdir_1 sim_outdir_2]

# On completion, report all status (note how a list is used for design_name):
check_testbench_simulations -design_name [list mod1* mod2] -design_id * \
      -sim_out_dir_list [list sim_outdir_1 sim_outdir_2]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

report_testbench_simulation_options

run_testbench_simulations

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

close_layout

Context: patterns -scan, patterns -scan_diagnosis

Mode: analysis

Closes the LDB. Any subsequent diagnosis you perform is non-layout aware.

Usage

close_layout

Description

Closes the LDB. Any subsequent diagnosis you perform is non-layout aware.

Arguments

None

Related Topics

create_layout

open_layout

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

close_pattern_set

Context: patterns -ijtag

Mode: analysis

Finalizes and closes the currently open pattern set.

Usage

close_pattern_set [-network_end_state {keep | reset | *initial*}]

Description

Must be run after the open_pattern_set command. If any iRead or iWrite commands remain that have not been processed by an iApply statement, close_pattern_set prints a warning, and then iApply processes them automatically.

By default, the end state of the IJTAG network is the one generated by the last iApply statement. When you specify the value reset or initial for the -network_end_state switch, the network is written with additional data to restore the values only in scan registers controlling the select lines of ScanMultiplexers according to switch value. When you specify the value keep for the -network_end_state switch, the result is the same as not using network_end_state, and the end state of the IJTAG network is the same as using only the close_pattern_set command.

The state of the network after running the close_pattern_set command will be the state at the beginning of the next pattern set opened by the open_pattern_set command with the -no_initial_ireset switch.

Arguments

- -network_end_state

  keep — This value is the default and it specifies to not take special actions when closing the pattern set. The state of the ICL network remains in the condition reached at the end of the last iApply command.

  reset — Value used when it is necessary to restore the reset state of the IJTAG network at the end of the pattern set. The reset state is restored using iWrite commands to the state control registers. State control registers control the select lines of ScanMultiplexers. The sib bits inside SIBs are an example of state control registers. These registers are written to their reset values specified in the ICL. Other scan registers are not affected. The reset state is often restored without extra scan operations, but depending on the IJTAG network configuration some network topologies may require additional scan loads.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The tool does not reset those scan registers in the scan path configuration when they are the target of an iTake command, or they receive their last written value from an explicit iWrite provided through PDL input. The tool resets scan registers in the scan path configuration that the IJTAG solver sets to fulfill an iWrite to another scan register. To determine which scan registers are not set to their reset value, use the command "set_ijtag_retargeting_options -network_end_state_warnings on" prior to closing the pattern set.

You can use this option when you need to run multiple patterns immediately after this one without doing an iReset between them. Because the reset state is restored at the end of the pattern sets, you can run them in arbitrary orders.

initial— Value used when it is necessary to restore the initial state of the IJTAG network at the end of the pattern set to what it was at the beginning of the pattern set. Restoration of the initial state is performed with iWrite commands to the state control registers. State control registers control the select lines of ScanMultiplexers. The sib bits inside SIBs are an example of state control registers. Other scan registers are not affected. The initial state is often restored without extra scan operations, but depending on the IJTAG network configuration some network topologies may require additional scan loads.

You can use this option when you need to run the same pattern set multiple times in a loop and the pattern set does not perform an iReset at the beginning. In this case, it is essential that the state of the IJTAG network is identical at the beginning of the first run as it is at the beginning of all subsequent runs. Performing a polling operation is an application in which a pattern set is run repeatedly.

You can also use this option when you need to run multiple patterns immediately after this one without doing an iReset between them and restoring a custom network state different than the state achieved by "-network_end_state reset". In other words, use this option when you want to have a custom initial state at the beginning of each pattern, without doing iReset which resets the entire IJTAG network, and you want to restore the scan path to a custom state without affecting the other registers.

Note

To specify a network_end_state for a test_setup procedure, use the "-test_setup_network_end_state" option of the set_ijtag_retargeting_options command.

Return Values

None

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

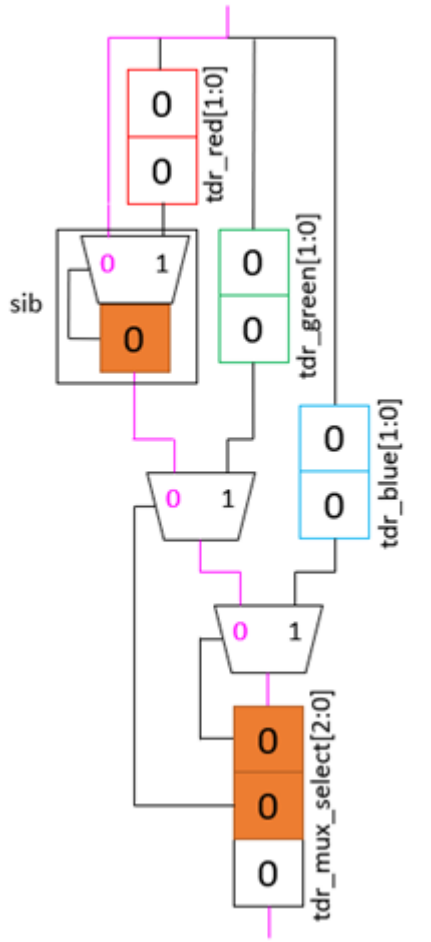
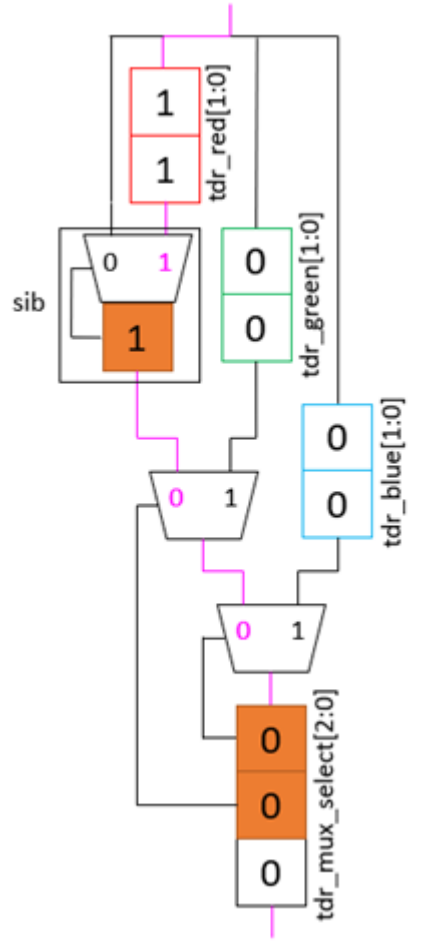
Examples

Example 1

The IJTAG network consists of four TDRs, two ScanMultiplexers and one SIB. Registers filled with color control select lines of the ScanMultiplexers (including the ScanMultiplexer inside the SIB). The active scan path is marked using magenta color.

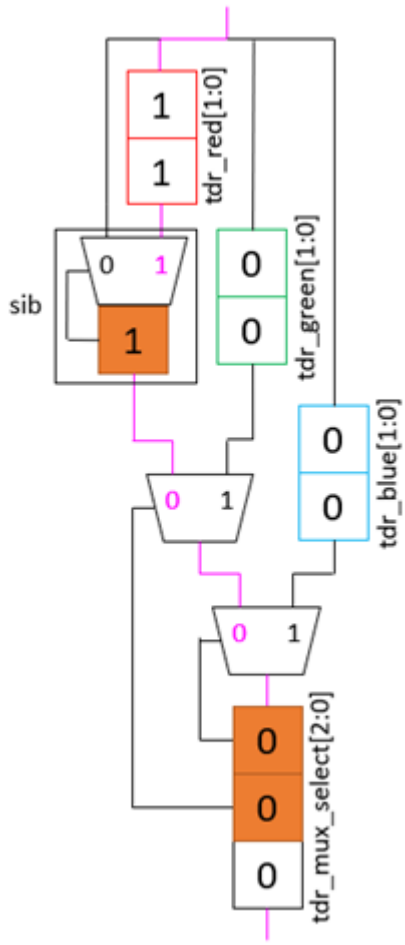
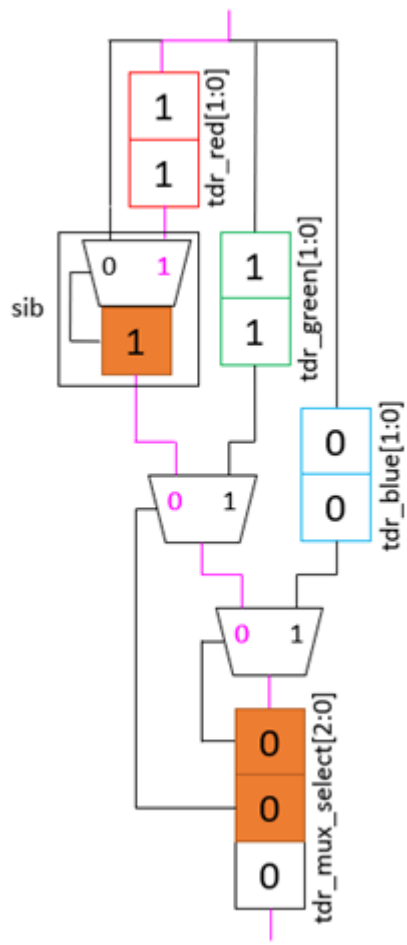
The following tables illustrate the effect of the various pattern set controls on the network state at the start of the pattern set and through the end of the pattern set.

Comments and Code Examples	Network state at open_pattern_set	Network state at close_pattern_set

Pattern: initial_pattern
Network End State: keep
 	
The IJTAG network of the example is in a reset state after an iReset command. The example code writes a value of 1 to the sib bit to change the scan path to write to the tdr_red scan register. The code keeps the network end state when it closes the initial pattern.
 	
Tessent Shell:
`open_pattern_set initial_pattern`
`  iWrite tdr_red.tdr 0b11`
`  iApply`
`close_pattern_set -network_end_state keep`
 	
Output PDL:
`iProc initial_pattern {} {`
`  iWrite sib.sib 0b1`
`  iRead  sib.sib 0b0`
`  iWrite tdr_mux_select.tdr[2:0] 0b000`
`  iRead  tdr_mux_select.tdr[2:0] 0b000`
`  iApply`
`  iWrite tdr_red.tdr[1:0] 0b11`
`  iRead  tdr_red.tdr[1:0] 0b00`
`  iWrite sib.sib 0b1`
`  iRead  sib.sib 0b0`
`  iWrite tdr_mux_select.tdr[2:0] 0b000`
`  iRead  tdr_mux_select.tdr[2:0] 0b000`
`  iApply`
`}`
		

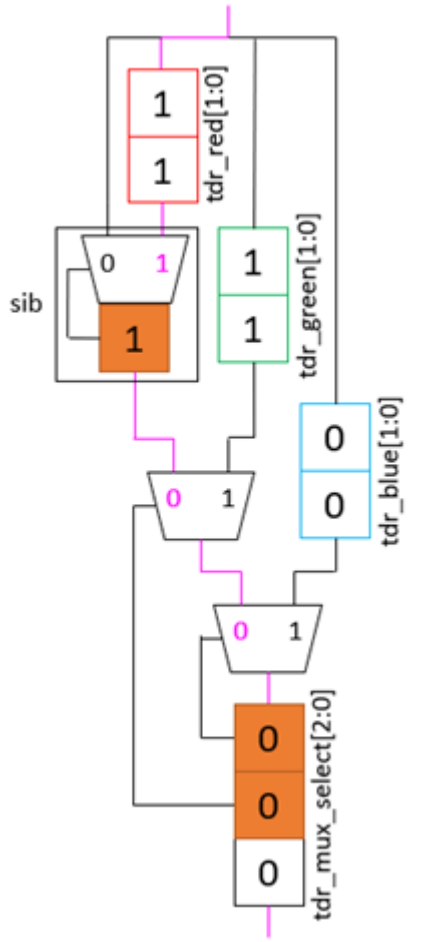
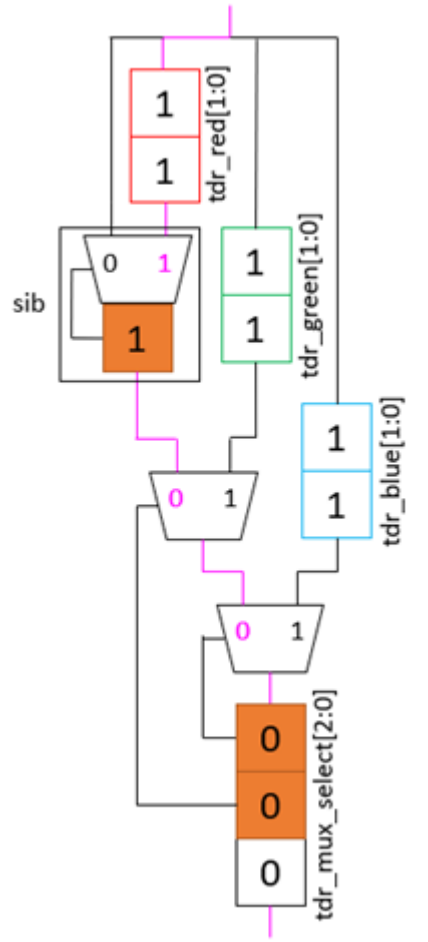
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Comments and Code Examples	Network state at open_pattern_set	Network state at close_pattern_set

Pattern: pattern1
Open Switch: -no_initial_ireset
Network End State: initial
 	
Pattern1 keeps the end state of the previous pattern when it opens by using the no_initial_ireset switch. At the end of pattern1, the example code restores the scan path to the initial state found at the beginning of the pattern set. Restoring the initial network state affects only the bits driving ScanMultiplexers. The pattern may run in a loop because the beginning and the end state of the active scan path are the same.
 	
Tessent Shell:
`open_pattern_set pattern1`
`-no_initial_ireset`
`  iWrite tdr_green.tdr 0b11`
`  iApply`
`close_pattern_set -network_end_state`
`initial`
 	
Output PDL:
`iProc pattern1 {} {`
`  iWrite tdr_red.tdr[1:0] 0b11`
`  iRead  tdr_red.tdr[1:0] 0b00`
`  iWrite sib.sib 0b1`
`  iRead  sib.sib 0b0`
`  iWrite tdr_mux_select.tdr[2:0] 0b010`
`  iRead  tdr_mux_select.tdr[2:0] 0b000`
`  iApply`
`  iWrite tdr_green.tdr[1:0] 0b11`
`  iRead  tdr_green.tdr[1:0] 0b00`
`  iWrite tdr_mux_select.tdr[2:0] 0b000`
`  iRead  tdr_mux_select.tdr[2:0] 0b000`
`  iApply`
`}`
		

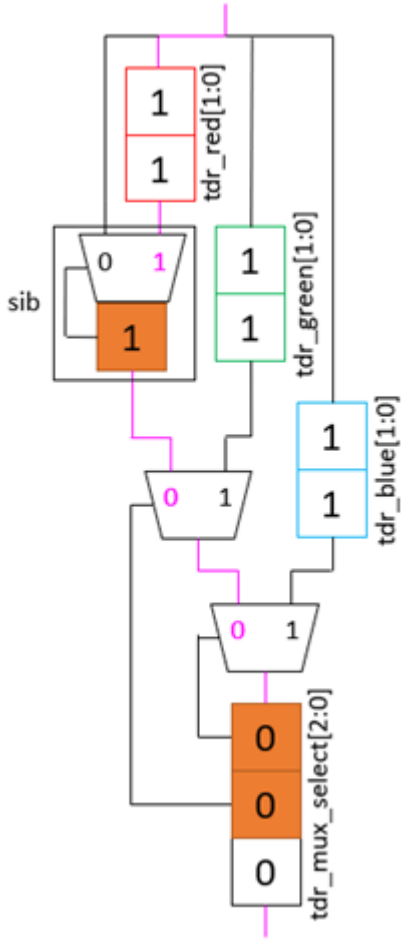
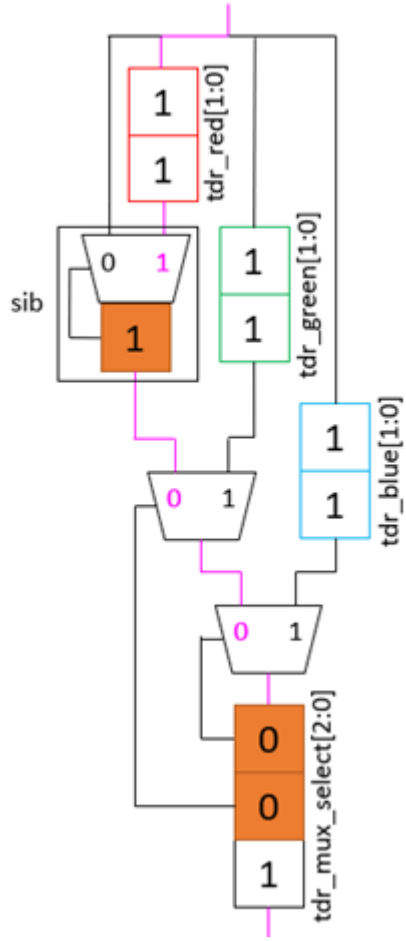
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Comments and Code Examples	Network state at open_pattern_set	Network state at close_pattern_set

Pattern: pattern2
Open Switch: -no_initial_ireset
Network End State: initial
 	
Pattern2 starts and ends with the same IJTAG network state as pattern1 above. By the end of pattern2, the example code restores the scan path to the initial state found at the beginning of the pattern set
 	
Tessent Shell:
`open_pattern_set pattern2`
`-no_initial_ireset`
`  iWrite tdr_blue.tdr 0b11`
`  iApply`
`close_pattern_set -network_end_state`
`initial`
 	
Output PDL:
`iProc pattern2 {} {`
`  iWrite tdr_red.tdr[1:0] 0b11`
`  iRead  tdr_red.tdr[1:0] 0b00`
`  iWrite sib.sib 0b1`
`  iRead  sib.sib 0b0`
`  iWrite tdr_mux_select.tdr[2:0] 0b001`
`  iRead  tdr_mux_select.tdr[2:0] 0b000`
`  iApply`
`  iWrite tdr_blue.tdr[1:0] 0b11`
`  iRead  tdr_blue.tdr[1:0] 0b00`
`  iWrite tdr_mux_select.tdr[2:0] 0b000`
`  iRead  tdr_mux_select.tdr[2:0] 0b000`
`  iApply`
`}`
		

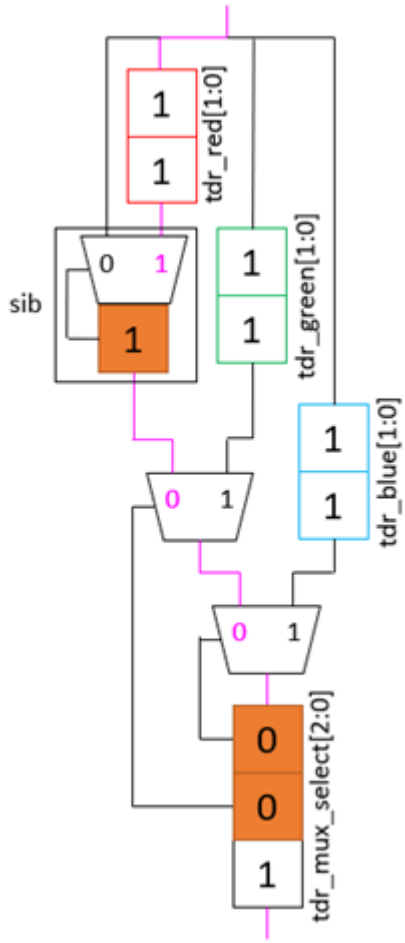
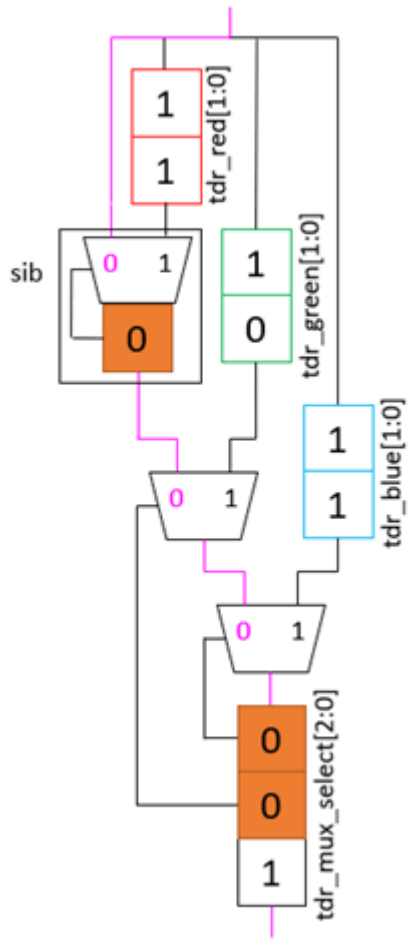
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Comments and Code Examples	Network state at open_pattern_set	Network state at close_pattern_set

Pattern: pattern3
Open Switch: -no_initial_ireset
Network End State: initial
 	
Pattern3 opens with explicit iWrites to tdr_mux_select to drive ScanMultiplexer select lines. By the end of pattern3, the example code restores the state of the scan path to the initial state found at the beginning of the pattern set. The code overwrites the values of the tdr_mux_select register bits 0 and 1 that control ScanMultiplexer select lines to restore the initial state.
 	
Tessent Shell:
`open_pattern_set pattern3 -no_initial_ireset`
`  iWrite tdr_mux_select.tdr 0b111`
`  iApply`
`close_pattern_set -network_end_state initial`
 	
Output PDL:
`iProc pattern3 {} {`
`  iWrite tdr_red.tdr[1:0] 0b11`
`  iRead  tdr_red.tdr[1:0] 0b00`
`  iWrite sib.sib 0b1`
`  iRead  sib.sib 0b0`
`  iWrite tdr_mux_select.tdr[2:0] 0b111`
`  iRead  tdr_mux_select.tdr[2:0] 0b000`
`  iApply`
`  iWrite tdr_blue.tdr[1:0] 0b11`
`  iRead  tdr_blue.tdr[1:0] 0b00`
`  iWrite tdr_mux_select.tdr[2:0] 0b100`
`  iRead  tdr_mux_select.tdr[2:0] 0b000`
`  iApply`
`}`
		

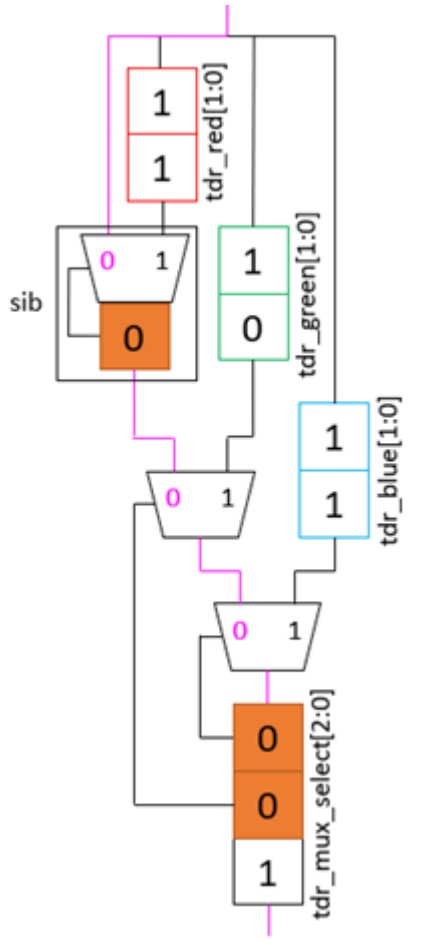
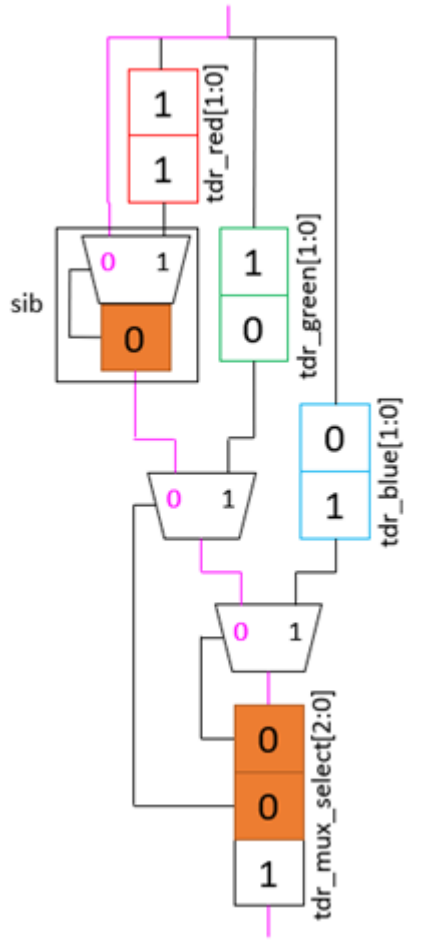
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Comments and Code Examples	Network state at open_pattern_set	Network state at close_pattern_set

Pattern: end_pattern1
Open Switch: -no_initial_ireset
Network End State: reset
 	
By the end of end_pattern1, the example code restores the values in scan registers controlling select lines of ScanMultiplexers to values declared in the ICL ResetValue property.
 	
Tessent Shell:
`open_pattern_set end_pattern1 -no_initial_ireset`
`  iWrite tdr_green.tdr 0b10`
`  iApply`
`close_pattern_set -network_end_state reset`
 	
Output PDL:
`iProc end_pattern1 {} {`
`  iWrite tdr_red.tdr[1:0] 0b11`
`  iRead  tdr_red.tdr[1:0] 0b00`
`  iWrite sib.sib 0b0`
`  iRead  sib.sib 0b0`
`  iWrite tdr_mux_select.tdr[2:0] 0b110`
`  iRead  tdr_mux_select.tdr[2:0] 0b000`
`  iApply`
`  iWrite tdr_green.tdr[1:0] 0b10`
`  iRead  tdr_green.tdr[1:0] 0b00`
`  iWrite tdr_mux_select.tdr[2:0] 0b100`
`  iRead  tdr_mux_select.tdr[2:0] 0b000`
`  iApply`
`}`
		

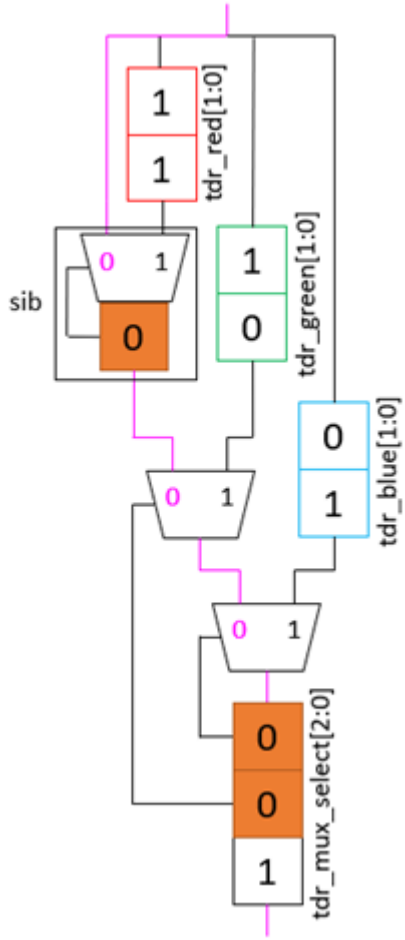
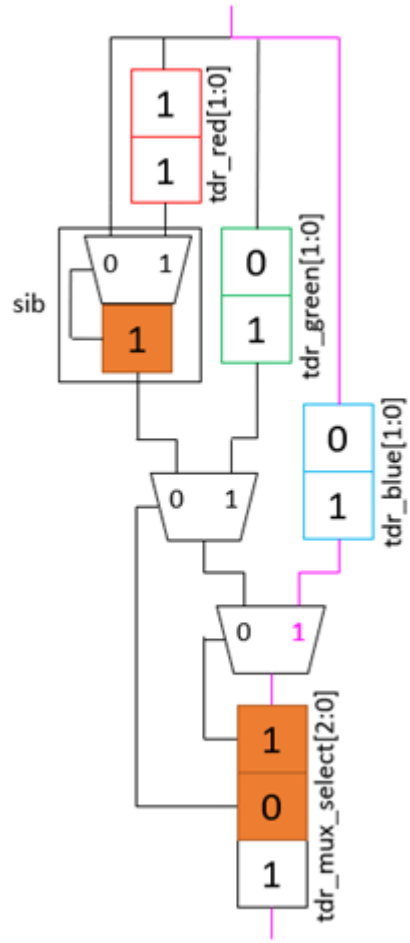
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Comments and Code Examples	Network state at open_pattern_set	Network state at close_pattern_set

Pattern: end_pattern2
Open Switch: -no_initial_ireset
Network End State: reset
 	
By the end of end_pattern2, the example code restores the values in scan registers controlling select lines of ScanMultiplexers to values declared in the ICL ResetValue property.
 	
Tessent Shell:
`open_pattern_set end_pattern2 -no_initial_ireset`
`  iWrite tdr_blue.tdr 0b01`
`  iApply`
`close_pattern_set -network_end_state reset`
 	
Output PDL:
`iProc end_pattern2 {} {`
`  iWrite sib.sib 0b0`
`  iRead  sib.sib 0b0`
`  iWrite tdr_mux_select.tdr[2:0] 0b101`
`  iRead  tdr_mux_select.tdr[2:0] 0b000`
`  iApply`
`  iWrite tdr_blue.tdr[1:0] 0b01`
`  iRead  tdr_blue.tdr[1:0] 0b00`
`  iWrite tdr_mux_select.tdr[2:0] 0b100`
`  iRead  tdr_mux_select.tdr[2:0] 0b000`
`  iApply`
`}`
		

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Comments and Code Examples	Network state at open_pattern_set	Network state at close_pattern_set

Pattern: end_pattern3
Open Switch: -no_initial_ireset
Network End State: reset
 	
By the end of end_pattern3, the example code restores the values in scan registers controlling select lines of ScanMultiplexers to values declared in the ICL ResetValue property. The SIB bit and tdr_mux_select register bit 2 keep their values because an iWrite specified through PDL writes to them in the last iApply of the pattern set.
 	
Tessent Shell:
`open_pattern_set end_pattern3 -no_initial_ireset`
`iWrite tdr_green.tdr 0b01`
`iApply`
`iWrite sib.sib 0b1`
`iWrite tdr_mux_select.tdr[2] 0b1`
`iApply`
`close_pattern_set -network_end_state reset`
 	
Output PDL:
`iProc end_pattern3 {} {`
`iWrite sib.sib 0b0`
`iRead sib.sib 0b0`
`iWrite tdr_mux_select.tdr[2:0] 0b110`
`iRead tdr_mux_select.tdr[2:0] 0b000`
`iApply`
`iWrite tdr_green.tdr[1:0] 0b01`
`iRead tdr_green.tdr[1:0] 0b00`
`iWrite tdr_mux_select.tdr[2:0] 0b100`
`iRead tdr_mux_select.tdr[2:0] 0b000`
`iApply`
`iWrite sib.sib 0b1`
`iRead sib.sib 0b0`
`iWrite tdr_mux_select.tdr[2:0] 0b101`
`iRead tdr_mux_select.tdr[2:0] 0b000`
`iApply`
`}`
		

Example 2

Code examples below present sequences of patterns from Example 1 which may be executed without any additional scan loads between subsequent iCalls.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Patterns from Example1 which begin and end with the same active scan path may be run in a loop.

```
open_pattern_set sequence1
 iCall initial_pattern
 //any sequence of pattern1, pattern2 or pattern3
 iCall pattern1
 iCall pattern2
 iCall pattern2
 iCall pattern3
 //end_pattern1 starts from initial state
 iCall end_pattern1
 //end_pattern2 starts from reset state
 iCall end_pattern2
 //end_pattern3 starts from reset state
 iCall end_pattern3
close_pattern_set

open_pattern_set sequence2
 //end_pattern2 starts from reset state
 iCall end_pattern2
 iCall end_pattern2
 //initial_pattern starts from reset state
 iCall initial_pattern
 iCall pattern3
 iCall pattern2
 iCall pattern1
 //end_pattern1 starts from initial state
 iCall end_pattern1
 //end_pattern3 starts from reset state
 iCall end_pattern3
close_pattern_set

open_pattern_set sequence3
 iCall initial_pattern
 //any sequence of pattern1, pattern2 or pattern3
 iCall pattern1
 iCall pattern1
 iCall_pattern3
 //end_pattern1 starts from initial state
 iCall end_pattern1
 //initial_pattern starts from reset state
 iCall initial_pattern
 iCall pattern2
 iCall pattern2
 //end_pattern1 starts from initial state
 iCall end_pattern1
 //end_pattern2 starts from reset state
 iCall end_pattern2
 iCall end_pattern2
 iCall end_pattern2
 iCall end_pattern3
close_pattern_set
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

get_open_pattern_set

get_pattern_set_list

index_collection

open_pattern_set

report_pattern_sets

reset_open_pattern_set

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

close_tsdb

Context: all contexts

Mode: all modes

Makes the contents of a previously opened TSDB directory invisible to the tool.

Usage

close_tsdb {***directory_path_list*** | **-all**} [-silent]

Description

The specified TSDB output directory is automatically opened and cannot be closed with this command.

Arguments

- ***directory_path_list***

  A required string that specifies an absolute or relative path to a TSDB directory, or a Tcl list containing one or many absolute or relative paths to TSDB directories. You must specify either the ***directory_path_list*** or **-all** option, but you cannot specify both.

- **-all**

  A required option that specifies that all previously opened TSDB directories are to be closed. Note that the specified TSDB output directory is automatically opened and remains opened. You must specify either the ***directory_path_list*** or **-all** option, but you cannot specify both.

- -silent

  An optional switch that suppresses the warning that is normally generated if the specified ***directory_path_list*** is not currently opened or if there are no opened TSDB directories when the **-all** option is used.

Examples

This example closes two TSDB directories with one command invocation.

get_tsdb_list

```
/home/projects/chipa/tsdbs/corea.tsdb
/home/projects/chipa/tsdbs/corec.tsdb
```

close_tsdb [list ${tsdb_home}/corea.tsdb ${tsdb_home}/coreb.tsdb]
get_tsdb_list

```
/home/projects/chipa/tsdbs/corec.tsdb
```

Related Topics

get_tsdb_list

get_tsdb_output_directory

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

open_tsdb

set_tsdb_output_directory

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

close_visualizer

Context: unspecified, all contexts

Mode: all modes

Saves display data and closes Tessent Visualizer.

Usage

close_visualizer [-server]

Description

Saves display data and closes the Tessent Visualizer client application. Selecting “X” from the window control buttons performs the same function as close_visualizer.

Note

The appearance of the window control buttons may differ from the above, depending on the particular desktop environment used.

Display data is automatically saved and remains persistent for the current tool session unless the -server option is used.

Arguments

- -server

  An optional switch that closes a Tessent Visualizer server. When the server is closed, all Visualizer-specific data such as schematic contents, opened Text/HDL Viewer and diagnosis report files, and filters are reset.

Examples

The following example opens Tessent Visualizer, displays the Flat Schematic window, and creates and displays a portion of the design, then closes the viewing session:

```
open_visualizer -display flat_schematic
add_schematic_objects /alu_inst/lui/z_reg[1]
close_visualizer
```

The following example opens and then closes a Tessent Visualizer server:

```
open_visualizer -server_only
close_visualizer -server
```

Related Topics

open_visualizer

set_visualizer_preferences

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

collect_cdp_test_failure_logs

Context: patterns -silicon_insight

Mode: all modes

Saves failure logs for a CDP test to a directory.

Usage

collect_cdp_test_failure_logs ***test_name_list*** [-flog_directory *flog_directory*] [-force]

Description

The collect_cdp_test_failure_logs command runs the main pattern and all diagnostic ATPG patterns for the specified test or tests and saves the failure logs in the specified directory.

Arguments

- ***test_name_list***

  A required string or Tcl list that identifies one or more CDP tests to collect failure logs from.

- -flog_directory *flog_directory*

  An optional switch and string that specifies the directory in which to create directories that contain the failure files for each specified CDP test. The default directory is the current working directory.

- -force

  An optional switch that collects failure logs even if the main pattern passes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

compare_collections

Context: unspecified, all contexts

Mode: all modes

Compares the contents of two collections.

Usage

compare_collections ***collection1 collection2*** [-order_dependent]

Description

If the same objects are in both collections, the result is 0 (zero). If the objects in the collections are different, the result is a non-zero value. The order of the objects can optionally be considered by using the -order_dependent switch.

If two empty collections are compared, the compare_collections command considers them identical and returns a result of 0 (zero).

Arguments

- ***collection1***

  A required string that specifies the base collection for the comparison.

- ***collection2***

  A required string that specifies the collection to compare with ***collection1***.

- -order_dependent

  An optional switch that causes the order of the objects to be considered when comparing the collections. The collections are considered to be different if the objects are ordered differently, and a non-zero value is returned.

Examples

The following example shows a variety of comparisons.

set a [get_ports a*]

```
{a[3] a[2] a[1] a[0]}
```

set b [get_ports b*]

```
{b[3] b[2] b[1] b[0]}
```

set c [get_ports c*]

```
{}
```

#Same objects, same order
compare_collections $a $a

```
0
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

#Same objects, same order
compare_collections $a $a -order_dependent

```
0
```

#Different objects
compare_collections $a $b

```
1
```

#Different objects
compare_collections $a $b -order_dependent

```
1
```

#Two null collections
compare_collections $c $c

```
0
```

#Same objects, different order
compare_collections [add_to_collection $a $b] [add_to_collection $b $a]

```
0
```

#Same objects, different order
compare_collections [add_to_collection $a $b] [add_to_collection $b $a] -order_dependent

```
2
```

Related Topics

add_to_collection

append_to_collection

copy_collection

filter_collection

foreach_in_collection

index_collection

is_collection

range_collection

remove_from_collection

sizeof_collection

sort_collection

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

compress_layout

Context: patterns -scan_diagnosis

Mode: analysis

Compresses a LDB.

Usage

LDBs created after v2013.3: compress_layout

LDBs created with v2013.3 and before: compress_layout ***layout_database_name***

Description

You must open the LDB using the open_layout command before performing the compression operation.

Refer to "Layout Database Compression and Decompression" in the *Tessent Diagnosis User's Manual* for complete information.

Arguments

- ***layout_database_name***

  A required string that specifies the location to and the name of the LDB to be created during the compression operation.

Related Topics

create_layout

open_layout

uncompress_layout

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

compress_patterns

Context: dft -edt, patterns -scan

Mode: analysis

Compresses patterns in the current test pattern set.

Usage

compress_patterns [*passes_integer*] [-Reset_au] [-MAx_useless_passes *integer*]
[-MIn_elim_per_pass *integer*] [-MUltiple_detection [-Order]]
[-MIXed {Essential | Order [-Delta *percentage*] [-Limit *integer*]}]

Description

Note

You should use create_patterns for the smallest pattern set. The compress_patterns command is for specialized processes.

The compress_patterns command performs static pattern compression on the current test pattern set by repeating fault simulation for the patterns in either reverse or random order and selecting only those patterns required for detection. The *passes_integer* argument specifies the number of pattern compression passes. The first pattern compression pass runs in reverse order and then alternates between random and reverse for additional passes. If you do not specify a *passes_integer* argument, the tool performs only one compression pass.

This is a multiprocessing command for fault simulation. This command supports IDDQ fault models. For more information about multiprocessing, refer to "Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual*.

If you specify the -Reset_au option, then when the tool performs pattern compression, it selects the AU faults for later fault simulation. If during pattern compression these AU faults simulate as possible-detected, the tool labels them as PU (possible-detected—ATPG_untestable), and they receive test coverage credit as possible-detected faults. If the number of pattern compression passes is greater than 1, the tool only performs the resetting of the AU faults for the first pass.

The compress_patterns command has a residual memory effect. The initial mode (reverse/random) is not fixed. Instead, it toggles back and forth, starting with the mode last used in the same run. The tool always starts with the reverse order at invocation.

If you include the -Multiple_detection switch, the static compression performed by this command does not simulate faults in the posdet (PD) fault class (PT and PU subclasses) and as a result, the tool may discard patterns that detect PD faults. Therefore, the test coverage reported after using this command switch may not be accurate if there are PD faults. To ensure you get

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

the most accurate report of test coverage in this case, re-simulate the patterns using one of the following approaches:

- Enter the "set_system_mode analysis" command followed by the simulate_patterns command.
- Enter the order_patterns command with the -Simulate option.

Arguments

- *passes_integer*

  An optional integer that specifies the number of pattern compression passes. The default is 1 (performs only one compression pass).

- -Reset_au

  An optional switch that specifies fault simulation of AU faults.

- -MAx_useless_passes *integer*

  An optional switch and integer pair that specifies the maximum number of consecutive, useless (no eliminated patterns) passes the tool permits before terminating the pattern compression process. This command option has no effect on the pattern set if the number of passes (*passes_integer* argument) is smaller than the *integer* value of this switch. The default is the *passes_integer* value.

- -MIn_elim_per_pass *integer*

  An optional switch and integer pair that specifies the minimum number of eliminated patterns required in a single pass to continue the pattern compression process. If you specify this switch, you must enter a value greater than 0.

- -MUltiple_detection

  An optional switch that specifies for the tool to use a multiple detection based algorithm to carry out static pattern compaction. When you use this switch, the tool runs only one pass of compaction and discards patterns that detect only posdet (PD) faults.

- -Order

  An optional switch used with the -Multiple_detection switch to specify pattern ordering at the same time as static pattern compaction.

- -MIXed Essential | Order

  An optional switch and string pair that specifies to use a mixed compaction technique to compact the current internal pattern set. If you use the "Essential" option, the tool considers essential faults, those faults detected only once in the pattern set, as candidates for dynamic compaction. This option ensures that the patterns used to detect the essential faults are not dropped during pattern compression to keep the reached coverage.

  If you use the "Order" option, the tool selects fault candidates for dynamic compaction based on pattern set ordering.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Using the "-Mixed Order" switch turns off pattern classification until you issue a "set_pattern_classification On" command.

Mixed compaction can reduce the size of the current internal pattern set in most situations, but Siemens EDA cannot guarantee that in every case the size of the pattern set after the tool performs mixed compaction is less than the size of the pattern set before mixed compaction. Therefore, be sure to save the current internal pattern set before carrying out mixed compaction.

- -Limit *integer*

  An optional switch and integer pair that modifies the behavior of the -Mixed option. The -Limit switch specifies the maximum number of faults that the tool attempts unsuccessfully to merge with the target fault pattern when dynamic compaction is involved. Without the -Limit switch, -Mixed causes the tool to use an adaptive algorithm to determine this number.

- -Delta *percentage*

  An optional switch and real number pair (between 0 and 100) that modifies the behavior of the -Mixed Order option. Normally, -Mixed Order selects fault candidates for dynamic compaction based on pattern set ordering. The -Delta switch causes the tool to consider as candidates for dynamic compaction, the specified percentage of the total number of faults instead of using pattern set ordering. The default percentage is 2.

Examples

The following example compresses the generated test pattern set with two passes; the first pass is by reverse order and the second pass is by random order:

```
set_system_mode analysis
create_patterns
compress_patterns 2
```

Related Topics

add_processors

create_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

convert_design_path_format

Context: all contexts

Mode: all modes

Converts a list of design paths to the specified dot or slash separator.

Usage

convert_design_path_format ***design_path_list*** {-to_dot_separator | -to_slash_separator}

Description

The convert_design_path_format command accepts a list of design instance names and performs a conversion of the hierarchy separators according to the selected option. The command returns the converted names in a list of the same size as specified with the command.

Arguments

- ***design_path_list***

  A required list that specifies the design instance hierarchical paths to be converted. In cases where the *design_path_list* contains escaped identifiers, ensure correct formatting of the input list is maintained. The Tcl "list of lists" construct can be used to avoid breaking the hierarchical name into parts at the white spaces terminating the escaped identifiers.

- -to_dot_separator

  A switch that specifies the hierarchical paths are converted to "." separators in the path name. This option is mutually exclusive with the -to_slash_separator option.

- -to_slash_separator

  A switch that specifies the hierarchical paths are converted to "/" separators in the path name. This option is mutually exclusive with the -to_dot_separator option.

Examples

The following example demonstrates the conversion of two hierarchical design instance paths from slash separators to dot separators:

```
SETUP>convert_design_path_format {core/modA/blockB core/modB/blockB} \
        -to_dot_separator
core.modA.blockB core.modB.blockB
SETUP>
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

copy_collection

Context: unspecified, all contexts

Mode: all modes

Duplicates the contents of an existing collection, resulting in a new collection. The base collection remains unchanged.

Usage

copy_collection ***collection***

Description

The only time you need to copy a collection is when you want to append to one copy while leaving the original copy unchanged. Copying a collection is equivalent to using add_to_collection with an empty collection as the second collection.

Arguments

- ***collection***

  A required string that specifies the collection to be copied.

Examples

The following example shows the results of copying a collection and then appending to the copy. Note how *a2* is modified by append_to_collection, but *a1* remains unaffected. The second reference to *a2* called *a3* sees the updated collection because it is simply a second reference to the same collection referenced by *a2*.

set a1 [get_ports a*]

```
{a[3] a[2] a[1] a[0]}
```

set a2 [copy_collection $a1]

```
{a[3] a[2] a[1] a[0]}
```

set a3 $a2

append_to_collection a2 [get_ports b[0]]

```
{a[3] a[2] a[1] a[0] b[0]}
```

puts [get_name_list $a1]

```
{a[3]} {a[2]} {a[1]} {a[0]}
```

puts [get_name_list $a2]

```
{a[3]} {a[2]} {a[1]} {a[0]} {b[0]}
```

puts [get_name_list $a3]

```
{a[3]} {a[2]} {a[1]} {a[0]}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

add_to_collection

append_to_collection

compare_collections

filter_collection

foreach_in_collection

index_collection

is_collection

range_collection

remove_from_collection

sizeof_collection

sort_collection

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

copy_module

Context: all contexts

Mode: insertion

Creates an exact copy of a design module and assigns it a new name so that it can then be used as part of a create_instance and replace_instances.

Usage

copy_module ***obj_spec*** **-new_name** ***module_name*** [-silent]

Description

Creates an exact copy of a design module and assigns it a new name so that it can then be used as part of a create_instance and replace_instances.

The copy_module command returns a collection with one module object corresponding to the newly created module.

Arguments

- ***obj_spec***

  A required value that specifies a module name or a collection containing a single module object as returned by the get_modules command.

- **-new_name** ***module_name***

  A required switch and value pair that specify the name of the new module once it has been copied. If *module_name* already exists, the module is copied and "_integer" is appended to the end of the *module_name* where integer increments as each new module of the same name is added.

- -silent

  An optional switch that suppresses the error that is normally generated if ***obj_spec*** does not point to a single module that exists. A null collection is returned when an error was suppressed. This option also suppresses the error that is generated if the specified *module_name* already exists.

Examples

Example 1

The following example creates a copy of module ModA to ModB. ModB is identical to ModA and can be instantiated into the design.

copy_module ModA -new_name ModB

```
{ModB}
```

Example 2

The following example introspects the module name of an instance to create a copy.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

copy_module [get_att_value u1/u2 -name module_name] -new_name ModB

```
{ModB}
```

Related Topics

create_instance

create_module

get_common_parent_instance

get_icl_ports

get_icl_pins

get_nets

replace_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

copy_simulation_context

Context: all contexts

Mode: setup, analysis

Prerequisite: A flat model must currently exist.

Copies the simulation values and forces from one simulation context to another.

Usage

copy_simulation_context [-from *source_context_name*] [-to *target_context_name*]

Description

The copy_simulation_context command copies the simulation values and forces from any predefined or user-defined simulation context to any user-defined simulation context.

Note that you must specify either the -to or the -from switch, or specify both switches.

Arguments

- -from *source_context_name*

  An optional switch and string pair that specifies the simulation context from which to copy the simulation values and forces. The source_context_name can be predefined (listed in Table 8-6 on page 2434) or user-defined. If you do not use this switch and you do supply the -to switch, the command uses the current simulation context as the source_context.

- -to *target_context_name*

  An optional switch and string pair that specifies the simulation context to which to copy the simulation values and forces. The target_context_name can be any user-defined context name that already exists and cannot be a predefined context name (listed in Table 8-6 on page 2434). If you do not use this switch and you do use the -from switch, the command uses the current simulation context as the target_context.

Examples

The following examples show two ways to copy the simulation values of the predefined simulation context, stable_load_unload, to the user-defined simulation context, myContext:

```
SETUP> set_current_simulation_context myContext
SETUP> copy_simulation_context -from stable_load_unload
```

You can accomplish the same as the above with a single command:

```
SETUP> copy_simulation_context -from stable_load_unload -to myContext
```

Related Topics

add_simulation_context

add_simulation_forces

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_simulation_contexts

delete_simulation_forces

get_simulation_context_list

get_simulation_value_list

report_simulation_contexts

report_simulation_forces

simulate_clock_pulses

simulate_forces

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_bisr_segment_order_file

Context: all contexts

Mode: analysis

Creates the BisrSegmentOrderSpecification configuration data wrapper and extracts the list and ordering of BISR chain segments.

Usage

create_bisr_segment_order_file [*design_id*] [-replace]

Description

This command extracts the list of repairable memory instances by inspecting the ICL file and performing design introspection of all memories that have the set_memory_instance_options -use_in_memory_bisr_dft_specification attribute set. The ScanInPort pins of BISR chains found on child instances within included ICL modules are also extracted.

When a DEF file is available, the command extracts the BISR chain and optimizes the order based on the coordinates of the memory instances. If the DEF file specifies a block instance, the command optimizes the ordering based on the block's BISR_SI pin coordinates. If a block's BISR_SI pin does not have the coordinates specified, but the parent block does, the command uses those coordinates for ordering optimization. If DEF information is not available, the command alphabetically sorts the BISR chain by memory instance names.

The BISR Segment Order File can be edited and the BISR chain can be re-ordered during the insertion phase of the design flow. Any BISR registers that do not appear in the BISR Segment Order File does not have a BISR module generated and has its repair ports untouched. For further information on assigning and controlling BISR chains, refer to the "Inserting BISR Chains in a Block" topic in the *Tessent MemoryBIST User's Manual For Use With Tessent Shell* manual.

The format of the BISR Segment Order File containing the BisrSegmentOrderSpecification wrapper is shown in Figure 13-72. The file containing the BisrSegmentOrderSpecification wrapper is referenced by the DftSpecification/MemoryBisr bisr_segment_order_file : *filename* property.

Arguments

- *design_id*

  An optional string that identifies the design. The default is the current design.

- -replace

  An optional switch that specifies the existing wrapper is to be replaced. If the command is run and finds a BISR Segment Order File already exists for the specified *design_id*, an error is generated informing the user to use -replace.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_capture_procedures

Context: dft -edt, patterns -scan

Mode: analysis

Creates named capture procedures based on clock sequences either derived from patterns or specified at the command line.

Usage

create_capture_procedures [-Name *capture_procedure_name*]
{**-PAtterns** [Internal | External] [*pattern_index*...] |
{{**-Clock_sequence** ***clock_sequence*** | **-SAME_clocks_pulse**
{**SIngle_clock_per_procedure** | **GRoup_noninteracting_clocks_per_procedure**}
[*pulse_times*]} [-FORce_pi [*cycle_number*...]] [-MAsk_po]
[-SLow_cycles {***cycle_number***...}] [-Load_cycles {***cycle_number***...}]
[[-COndition *pinpathname value* [*cycle_number*...]]...]
[[-TImeplate [Fast | Slow] *timeplate_name*]...]
[{-EXT_Clock_sequence *clock_sequence* | -EXT_Same_clock_pulse *external_clock_name pulse_times*} [-EXT_Timeplate *timeplate_name*]]} [{>|>>} *file_pathname*]

Description

Use the create_capture_procedures command before or after pattern generation. This command creates the procedures in the internal memory only. Save the procedure to a test procedure file using the write_procfile command.

For information on using named capture procedures, see "At-Speed Test With Named Capture Procedures" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- -Name *capture_procedure_name*

  An optional switch and string pair that specifies the name of the capture procedure to create. If there is already a procedure with this name, a suffix is added to the name specified. If this command creates multiple named capture procedures, it uses the specified *capture_procedure_name* as the prefix of each name and adds a suffix. If you do not specify -Name, the default name is "user_capture_proc<*n*>," where <*n*> is an integer suffix.

- **-PAtterns** [Internal | External] [*pattern_index*...]

  A switch that specifies which patterns to extract clock sequences from for the named capture procedures. The following options are available:

  [Internal | External] — An optional literal that extracts clock sequences from internal or external patterns. The default is internal if not specified.

  [*pattern_index*...] — An optional, repeatable string that extracts clock sequences from the patterns with the specified index. The pattern index starts at 1. By default, the tool extracts clock sequences from all the patterns of the internal or external pattern set if a pattern index is not specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-Clock_sequence *clock_sequence***

  A switch and string pair that specifies the clock sequence for the named capture procedure. *clock_sequence* is a sequence of cycles. A cycle is denoted by one or more clocks enclosed by a pair of parentheses. Adjacent clocks within a cycle must be separated by either white space or a comma. The clock is designated by the gate identification number or pin pathname. For example, (c1,c2) (c1) (c2,c3) represents a clock sequence with three cycles. In the first cycle, clock c1 and c2 are pulsed; in the second cycle, clock c1 is pulsed; in the third cycle, clock c2 and c3 are pulsed.

- **-SAME_clocks_pulse** {**SIngle_clock_per_procedure** | **GRoup_noninteracting_clocks_per_procedure**} [*pulse_times*]

  A switch, literal, and optional value that automatically generate intra clock domain named capture procedures.

  SIngle_clock_per_procedure — A literal that pulses a single clock in every pulse cycle. The command creates a named capture procedure automatically for every clock in the design that is not constrained to its off state value. Also, all clocks defined as its equivalence are pulsed when a single clock is pulsed in every cycle.

  GRoup_noninteracting_clocks_per_procedure — A literal pulses a clock group, which consists of a group of non-interacting clocks, in every pulse cycle. Each clock group consists of a maximal set of non-interacting clocks. Once clocks are classified into groups, a named capture procedure is created for every clock group. (To understand the difference between non-interacting and interacting clocks, see the description of interacting clocks in the “domain_clock” and “-any_interaction” options in the set_clock_restriction command.)

  pulse_times — An optional value that specifies the number of cycles that pulse clock(s). By default, *pulse_times* is 2 if not specified.

- -FORce_pi [*cycle_number*...]

  An optional switch and repeatable integer that force PI values in the specified cycle(s). If *cycle_number* is not specified, all the cycles contain the force_pi statement. By default, only the first cycle contains the force_pi statement. The cycle number starts at 1.

- -MAsk_po

  An optional switch that specifies that the last cycle contains no measure_po statement. By default, the last cycle contains measure_po to indicate the time when the PO measurement happens unless you specify “set_output_masks on” or “add_output_masks -all”.

- -Slow_cycles {*cycle_number*...}

  An optional switch and repeatable integer that specify the slow cycles of the procedure.

- -Load_cycles {*cycle_number*...}

  An optional switch and repeatable integer that specify where the scan load happens before the capture cycles of the procedure. By default, only the single scan load happens before the first cycle.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -COndition *pinpathname value* [*cycle_number*]

  An optional switch, pin pathname, integer, and cycle integer that specify a condition of the scan cells in the named capture procedure. If you do not specify *cycle_number*, the condition is a global condition; otherwise, the condition is a local condition associated with the specified cycle only.

- -TImeplate [Fast | Slow] *timeplate_name*

  An optional switch, literal, and string that specify the timeplate for the fast or slow cycles of the named capture procedure. By default, the specified timeplate applies to the fast cycles of the named capture procedure. If you specify "slow *timeplate_name*", the tool applies this timeplate to the beginning of each scan load cycle.

  If you do not specify this optional switch, the named capture procedures use the default timeplate. You can define the default timeplate by specifying the Set Default_timeplate statement in the procedure file.

- -EXT_Clock_sequence *clock_sequence* | -EXT_Same_clock_pulse *external_clock pulse_times* [-EXT_Timeplate *timeplate_name*]

  An optional switch, string, and value that automatically create the external mode of the named capture procedures. Do not use this switch together with the **-Patterns** switch.

  -EXT_Clock_sequence *clock_sequence* — An optional switch and string pair that specifies the clock sequence for the external mode of the named capture procedure. *clock_sequence* is a sequence of cycles. A cycle is denoted by one or multiple clocks enclosed by a pair of parentheses. Adjacent clocks within a cycle must be separated by either white space or a comma. The designated clock or clocks must be an external clock or clocks.

  -EXT_Same_clock_pulse *external_clock pulse_times* — An optional switch, string, and value that specify to pulse an external clock in every cycle in the external mode. You can specify the external clock by its gate identification or pin pathname. *pulse_times* specifies the number of cycles that pulse the external clock.

  -EXT_Timeplate *timeplate_name* — An optional switch and string that specifies the timeplate to use for the external mode of the named capture procedure. The default timeplate is used if this optional switch is not specified.

- > *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list to create or replace the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list to append the contents of *file_pathname*.

Examples

Example 1

The following example creates named capture procedures based on the clock sequences extracted from internal patterns.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_system_mode analysis
create_patterns
create_capture_procedures -name my -patterns internal
```

Example 2

The following example creates named capture procedures based on the clock sequences extracted from the 1st, 12th, and 20th patterns in the external pattern set.

```
read_patterns pattern_file
create_capture_procedures -name my -patterns external 1 12 20
```

Example 3

The following example creates a named capture procedure of two cycles for every non-constrained clock in the design.

```
create_capture_procedures -same_clocks_pulse  single_clock_per_procedure 2
```

Example 4

The following example creates a named capture procedure with a user-defined clock sequence. The defined clock sequence has four clock cycles. In cycle 1, clock1 is pulsed. In cycle 2, clock1 and clock2 are pulsed simultaneously. In cycle 3, clock1 and clock3 are pulsed simultaneously. In cycle 4, clock1 is pulsed and PO is measured. In this example, all cycles are at-speed cycles.

```
create_capture_procedures -clock_sequence (clock1) (clock1, clock2) (clock1, clock3) \
(clock1)
```

Example 5

The following example creates a named capture procedure using the same clock sequence defined in the previous example. In the first cycle, the scan cell needs to satisfy the condition of having the loading value of 1.

```
create_capture_procedures -clock_sequence (clock1) (clock1, clock2) (clock1, clock3) \
(clock1) -condition /inst_core/inst_iacc/inst_reg 1 1
```

Related Topics

delete_capture_procedures

report_capture_procedures

set_capture_procedures

write_procfile

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_connections

Context: all contexts

Mode: insertion

Connects pin, net, or port objects.

Usage

Usage 1: N-to-N pairing connections

create_connections ***object_spec1 object_spec2***
[-net_name *net_name*] [-net_uniquification_suffix *net_uniquification_suffix*] [-silent]

Usage 2: 1-to-N broadcast connections

create_connections ***object_spec1 object_spec2***
[-net_name *net_name*] [-net_uniquification_suffix *net_uniquification_suffix*]
[-treat_common_ancestor_as_unique on | off] [-silent]

Usage 3: N-to-1 constant connections

create_connections ***object_spec1*** **-constant {1 | 0}**
[-net_name *net_name*] [-net_uniquification_suffix *net_uniquification_suffix*] [-silent]

Description

You can connect any of the object types pins, nets, or ports as described here:

Usage 1: When object_spec1 and object_spec2 contain the same number of elements, the tool pairs the elements from the two lists in the order of the two lists and creates a connection between each pair. Only one driver element per pair is permitted.

Usage 2: When object_spec1 has one element, and object_spec2 has one or more elements, object_spec2 cannot include any driver elements, such as an input port or an output pin.

Usage 3: When you specify -constant, you cannot specify object_spec2 and object_spec1 cannot contain any driver element, such as an input port or an output pin.

When the -net_name option is not specified, the net and pin name used to create the connection is automatically inferred from the name of the elements within object_spec1 using the following algorithm:

- If the element in object_spec1 is a pin, the tool uses the leaf_pin_name unless it has fewer than three characters; if it has fewer than three characters, the tool uses the leaf_instance_name of the pin concatenated with the leaf_name of the pin and separated by an underscore.
- If the element in object_spec1 is a net or a port, the tool uses the leaf name irrespective of the number of character it has.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can override the default net name by using the -net_name switch and specifying the net_name option to be used to create the intermediate nets and pins. Regardless of whether the default or a user-specified net name is specified, the tool checks for net conflicts and uniquifies the newly created net to avoid accidental shorts. The uniquify operation is performed using the net_uniquification_string suffix specified with the set_insertion_options command.

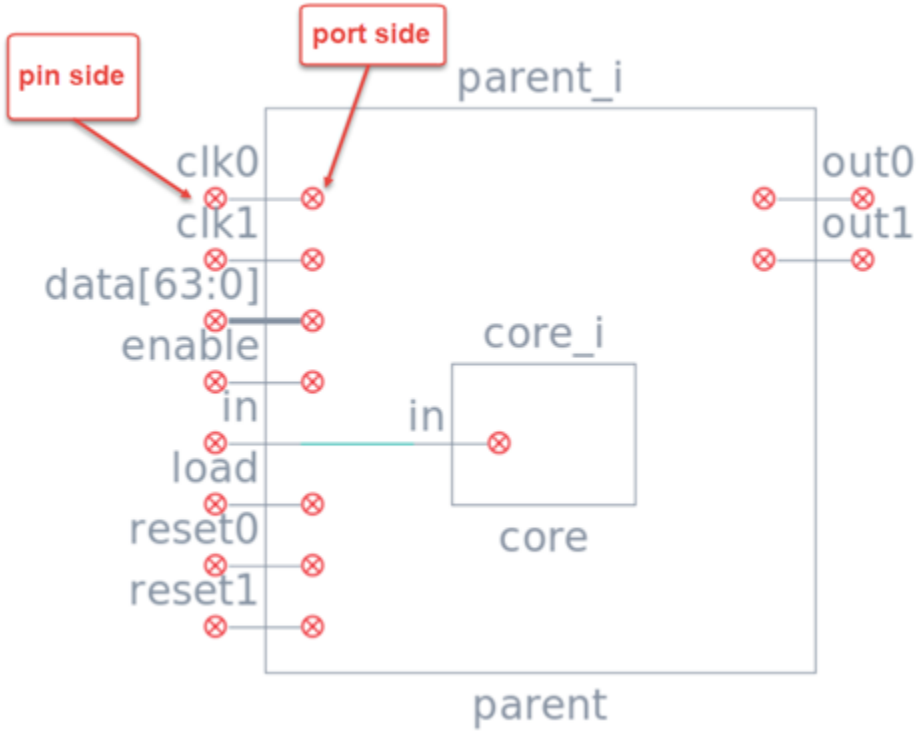
The two endpoints of each connection may be in different parts of the design hierarchy. The tool creates all of the intermediate pins needed to create the connection through the hierarchy. The tool automatically reuses existing pins that already make up part of the requested connection.

If you connect more than one pin within a given instance to a common source outside the instance, the tool creates only a single pin on the instance interface and reuses that same pin for all of the connections you want to be specified within the instance.

Note

The tool connects to the port side of the pin on the parent instance instead of the pin side when you want to connect the pin of a parent instance to a pin of one of its child instances.

Figure 3-30. Pin and Port Definition on an Instance



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***object_spec1 object_spec2***

  A required value that specify a set of endpoints of the connections to be created. The value of the object_spec argument can be a Tcl list of one or more pin, port, or net names; or a collection of one or more pin, port, or net objects.

  The size of object_spec1 must either be one or equal to the size of the object_spec2 when object_spec2 is specified. The object_spec1 argument cannot include any driver elements when you specify the -constant option. At most, one driver can exist in each connection pair when you specify both object_spec1 and object_spec2.

  If an object does not exist within one of the object_spec and you specify the -silent option, the tool ignores the object and skips the connection request. If the object does not exist and you do not specify the -silent option, the tool generates an error.

- **-constant 1 | 0**

  A switch and literal pair that specifies a constant value to connect to all elements within object_spec1. The -constant switch is mutually exclusive with the object_spec2 argument.

  When this option is specified, the object_spec1 argument cannot contain any driver elements such as input ports or output pins. In Verilog, a 1'b0 or 1'b1 connection is used as the connection to the pins, and a buf primitive or an assign statement with its input connected to 1'b0 or 1'b1 is used as the connection to output ports or to nets.

- -net_name *net_name*

  An optional argument that specifies the name of the pin created on object_spec1 or object_spec2 to route the connection. If -net_name is not specified, the tool uses the leaf name of object_spec1 to name the pin. If object_spec1 is a pin and its leaf name has fewer than three characters, the tool uses the leaf instance name of the base name of the pin concatenated with the leaf pin name and separated by an underscore.

- -net_uniquification_suffix *net_uniquification_suffix*

  An optional argument that defines the string that Tessent Shell uses to uniquify a net when the net that is trying to be created already exists in the design. The string must contain letters, numbers, or underscores and one and only one # symbol at the end identifying a counter that is auto-incremented starting from 1 until the net name does not already exist. For example, specifying -net_uniquification_suffix of _TS# appends the string _TS# to the net where # is replaced by the first integer above 0, making the net name unique. If net A, B, and B_TS1 already exist in the design, Tessent Shell would use A_TS1 if it needed to create a new net with default name A and would use B_TS2 if it tried to create a new net with default name B.

  When unspecified, the suffix defaults to the value set by the set_insertion_options -net_uniquification_suffix command.

- -treat_common_ancestor_as_unique <u>on</u> | off

  An optional switch and literal pair that specifies whether the create_connections command treats the common ancestor module of a one-to-N connection as unique. By default, the tool

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

does not lift the connections as if the common ancestor is unique (that is, instantiated only once). This results in the same one-to-N connections being made in all instances of the common ancestor module. This is the setting used when instantiating the instruments as part of the process_dft_specification command.

When set to off, the connection is lifted above the common ancestor if the common ancestor is non-unique and the connection does not include all repeated instances of the destinations. This enables the remaining destinations to be connected to a different source. This functionality is only available when auto-uniquify is off.

When unspecified, the setting defaults to the value set in the "set_insertion_options -treat_common_ancestor_as_unique" command.

- -silent

  An optional argument that specifies to ignore specified objects that do not exist and to not generate error messages. Connections with at least one non-existent object are skipped.

Examples

Example 1

The following example illustrates Usage 1 and creates a connection between three scalar ports and a 3-bit bus pin. The pin name created on u2 to route the connection is u2_a[2:0]. This pin name is constructed by concatenating the leaf_instance_name of u1/u2/a with the leaf_name of pin a with an underscore, because it has fewer than three characters.

```
create_connections u1/u2/a[2:0] {a b c}
```

Example 2

The following example illustrates Usage 2 and creates three connections between port IN1 and pins blockA/modB/u1/a, blockA/modB/u2/a, and blockA/modC/u1/a. The command creates a single net called myname on u1, u1/u1 to make all three connections.

```
create_connections IN1 {blockA/modB/u1/a blockA/modB/u2/a blockA/modC/u1/a} \
-net_name myname
```

Example 3

The following example illustrates Usage 3 and connects all pins matching the name ABC*/TE to a constant 0 value.

```
create_connections [get_pins ABC*/TE -hier] -constant 0
```

Example 4

The following example illustrates Usage 2 and shows a one-to-N connection including Verilog generate loop statements. The ltest_to_en output on a module outside the generate loop is broadcast to the LV_TM inputs on instances inside two different generate loops.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
create_connections {
  TOP_rtl_tessent_mbist_bap_inst/ltest_to_en
  TOP_rtl_tessent_mbist_bap_inst/ltest_to_en
  TOP_rtl_tessent_mbist_bap_inst/ltest_to_en
  TOP_rtl_tessent_mbist_bap_inst/ltest_to_en
} {
  param_parent_genloop2_l/child_genloop[0].memory_interface_inst/LV_TM
  param_parent_genloop2_l/child_genloop[1].memory_interface_inst/LV_TM
  param_parent_genloop3_l/child_genloop[0].memory/memory_interface_inst/LV_TM
  param_parent_genloop3_l/child_genloop[1].memory/memory_interface_inst/LV_TM
}
```

Related Topics

delete_connections

get_icl_pins

get_icl_ports

get_nets

move_connections

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_custom_bisr_register

Context: dft

Mode: setup

Creates a customized BISR register that is manually instantiated by the user in the functional design and is used to control functional logic.

Usage

create_custom_bisr_register ***register_name*** [-size *integer*] [-output_directory *output_dir*] [-rtl_extension *rtl_extension*] [-so_retiming on | off] [-with_parallel_load] [-replace]

Description

The create_custom_bisr_register command creates a BISR register with the specified ***register_name***. An example implementation may be where the custom BISR register is used to control functional logic that configures memory read/write margins.

The command generates the RTL and ICL files for the BISR register and places them in the directory specified by the -output_directory property. You must manually instantiate the register in the functional design after it is generated, prior to running DFT insertion. When you run the check_design_rules command, Tessent Shell identifies the custom BISR register instances and lists them inside the .bisr_segment_order file, as specified by the MemoryBisr wrapper in the DftSpecification. During process_dft_specification, the custom BISR registers in the design are combined into the BISR chains, along with the other BISR registers that are used for memory repair.

The size of the custom BISR registers should be kept comparable to the size of the BISR registers used for memory repair, and if there are several, the default value should be “0” in most cases. The objective is to keep a low density of bits requiring the value “1”, to minimize the impact on the number of fuses required to store information. Refer to the “Compression Algorithm and Fuse Box Organization” section for additional information. Information requiring large custom registers or a large number of custom registers that would cause a high density of 1s, should be stored directly in the fuse box using the TAP access mode described in “FuseBox Access” rather than using the autonomous self_fuse_box_program operation.

The contents of the BISR register can be loaded serially using the PatternsSpecification MemoryBisr bisr_chain_access run mode. Data can be captured from the BISR register D input ports with load_bisr_chain operation, which is run in the MemoryBisr autonomous run mode. The contents of the custom BISR register is compressed into the fuse box during the self_fuse_box_program autonomous operation. During a power-up, the contents of the fuse box is decompressed and the values of the custom BISR register are restored.

Arguments

- ***register_name***

  A required string pair that specifies the name of the custom BISR register.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -size *integer*

  An optional switch and integer pair that specifies the number of bits contained within the custom BISR register.

- -output_directory *output_dir*

  An optional switch and string pair that specifies the output directory where the RTL and ICL file are written. If not specified, the default location is the current directory.

- -rtl_extension *rtl_extension*

  An optional switch and string pair that specifies the extension for the Verilog RTL file. If not specified, the ".v" extension is used.

- -so_retiming on | off

  An optional switch and boolean pair that specifies whether the final output of the custom BISR pipeline register has a negedge register before the scanout port of the custom BISR pipeline module. This is the default configuration.

  The presence of the negedge register reduces setup margin by half a clock period, however it simplifies the BISR hold timing closure with the addition of a half clock period of margin. You should only need a BISR pipeline register without the final negedge register in special cases where you do not meet setup timing closure due to excess delay from the transmitting BISR register to the receiving register. Additionally, when the negedge register is not present, it is mandatory to balance the BISR clock throughout the design to control skew and avoid hold time closure issues.

- -with_parallel_load

  A switch that you must specify when you are creating a custom BISR register used with the Fast BISR Loading method for the BISR controller. When specified, the custom BISR register includes multiplexers that select the parallel inputs.

- -replace

  An optional switch that overwrites any existing files when specified.

Examples

The following example shows the creation of a custom BISR register named "user_bisr" that is 32 bits wide. The Verilog RTL and ICL files for the register are populated in the current directory and the default extension is used for the Verilog RTL file.

```
SETUP>create_custom_bisr_register usr_bisr -size 32
//     Generating design files for BISR module usr_bisr
//       Verilog RTL : ./usr_bisr.v
//       IJTAG ICL   : ./usr_bisr.icl
SETUP>
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_dft_specification

Context: dft (with no sub context)

Mode: analysis

Creates the DftSpecification(*design_name*,*id*) configuration wrapper and returns the newly created wrapper object so that it can be stored in a variable and used to customize the created specification.

Usage

create_dft_specification [-existing_ijtag_host_scan_in *host_scan_in_design_pin_spec*]
[-existing_primary_tap_scan_out *primary_tap_client_scan_out_design_pin_spec*]
[-existing_bscan_host_scan_in *bscan_host_scan_in_design_pin_spec*]
[-tile_ijtag_host_list *tile_ijtag_hosts*]
[-stap_host_list *stap_nodes*]
[-active_high_compliance_enables *enable_port_name* ...]
[-active_low_compliance_enables *enable_port_name* ...]
[-sri_sib_list *sri_sib_list*]
[-sti_sib_list *sti_sib_list*]
[-replace]

Description

This command finds IJTAG instances in the design that need to be connected to the IjtagNetwork and automatically creates the IjtagNetwork wrapper to connect them. The command supports multiple DFT insertion passes into any given module. The tool excludes elements seen to already be connected to the network from the current specification. An IJTAG instance is automatically seen by the command if its ICL file was loaded and the ICL module was matched to the design module during elaboration. Refer to "set_design_sources -format" and the set_module_matching_options command descriptions for more details about this process. Use report_ijtag_instances or get_ijtag_instances to find which instances have a matched ICL module.

This command also creates the necessary IJTAG network to control created memory BIST and memory BISR controllers as well as providing a boundary scan host port to BoundaryScan. To create a DFT Specification for any of these three features, you must enable them in setup mode using the set_dft_specification_requirements command prior to issuing the check_design_rules command. Refer to the BoundaryScan sections, if you are not familiar with these flows. Even though they are documented in separate sections, you can insert all instrument types in one insertion pass or in successive passes. The tool supports both methods and is simply controlled by what options you use with the set_dft_specification_requirements command prior to issuing the check_design_rules command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

If you have used add_dft_signals scan_en to define top level ports as DFT signals of type scan_en, it is automatically added to the AdvancedOptions/ConstantPortSettings wrapper such that the scan enable signals are automatically asserted off during the pattern generation. If you have other scan enable ports not identified with the add_dft_signals, you must add them yourself in the create PatternsSpecification.

Note

If you want to perform boundary scan insertion and TAP insertion pointing directly to internal pins instead of top-level ports, refer to the instructions in Example 2 of the IjtagNetwork wrapper section.

How the IjtagNetwork Specification is Created

The set_design_level command is used to define the target usage level of the current design. When the level is set to “physical_block” or “sub_block”, the IJTAG interface created by the create_dft_specification command is a client IJTAG interface. When the level is set to “chip”, it is a client TAP interface. The TAP interface port names can be set by create_dft_specification if the ports are identified with the Port built-in attribute “function” set to tdi, tdo, tms, tck, or trst. If a port function is not set, then create_dft_specification reverts to the DefaultsSpecification value or the default value of the property.

For the first dft insertion pass containing any IJTAG component, a root level IJTAG ring is created with one or two Sibs in series. The two sibs are called Sib(sti) and Sib(sri) and are inserted on demand when needed by the current insertion pass if it was not already inserted by a prior insertion pass. The Sib(sti) provides access to the IJTAG network for all instruments in the current design that are to be scan tested along with the rest of the functional logic by the logictest modes such as EDT or logicBIST. The acronym “sti” refers to “Scan Tested Instrument”. As shown in Figure 13-18, that Sib provides isolation between its host port and its clients. The Sib(sri) is used to provide access to the IJTAG network for child physical blocks, and for logictest instruments and Tdrs used to configure the logictest modes. The acronym “sri” refers to “Scan Resource Instrument”. These IJTAG nodes must remain active and undisturbed during the logictest modes, and are excluded from the logictest scan chains.

An ICL module is identified as a Scan Resource Instrument when it has the ICL module attribute called keep_active_during_scan_test set to “true”, which is the default, and that the instrument is treated as a Scan Resource Instrument and placed under the Sib(sri). If you have an instrument that you want to be tested like any other functional logic during scan test, you must specify the keep_active_during_scan_test attribute to “false” in its ICL module.

At the chip level, the first insertion pass inserts a TAP controller unless you use the -existing_ijtag_host_scan_in option of the create_dft_specification command to reference a design pin. The design pin must be on an instance having an associated ICL module, and the pin must be a ScanInPort referenced by a Host ScanInterface wrapper. Example 1 shows those different scenarios.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Any instance of a module having a corresponding ICL module is, by default, inserted into the IjtagNetwork wrapper of the DftSpecification unless it is seen as being already connected to the network. The set_ijtag_instance_options command or the tessent_use_in_dft_specification icl_module attribute can be used to change the default behavior.

Different ICL elements (ScanInterfaces, DataInPorts, and DataOutPorts) seen in the circuit are handled by this command as described here:

- **Client ScanInterface** — A Client ScanInterface is referenced by a DesignInstance wrapper when it is to be inserted into the network as part of the current DftSpecification. It is not referenced by the current IjtagNetwork when the ScanInPort associated to the Client ScanInterface has its attribute "has_functional_source" set to true. You can use the set_ijtag_instance_options command to force its inclusion or exclusion independent of the value of the has_functional_source attribute.

  If the ScanInterface is on an instance that is defined as a physical block (as defined by the tessent_is_physical_module attribute of its ICL module), it is inserted below the Sib(sri).

  If the ScanInterface is on an instance that is not defined as a physical block (as defined by the tessent_is_physical_module attribute of its ICL module), it is inserted behind the Sib(sti) or Sib(sri) based on the value of the keep_active_during_scan_test attribute on its ICL module.

  If you have read in UPF or CPF descriptions using the read_upf or read_cpf commands, the information is used to make sure that groups of DesignInstances associated to different power domains are isolated by a SIB. This is done to avoid daisy chaining them with elements from a different power domain and running into issues when accessing the elements in a given power domain while the other power domains are off.

- **DataInPorts** — A DataInPort is referenced by a connection property inside the Tdr/DataOutPorts wrapper when it is to be controlled by the network. The DataInPort is not referenced by the current IjtagNetwork when there is already a DataOutPort or a primary input port in its fan-in or the DataInPort has the icl_port the tessent_use_in_dft_specification attribute set to "false". If the tessent_use_in_dft_specification attribute is set to "auto_no_pi", the port is sourced by a TDR port even if it is sourced by a primary input port. Use this value on special ports like the reset port of a PLL or clock divider to make sure it is locally controllable by IJTAG. This concept is further explained in the ProcedureStep wrapper section of the PatternsSpecification chapter. It contains a complete example of a clock divider re-used by memory BIST and initialized by IJTAG. If the DataInPort has a functional source, a multiplexer is inserted to control the DataInPort by the IjtagNetwork in test mode without disturbing its functional source its mission mode. Use the set_ijtag_instance_options command to force its inclusion or exclusion independent of whether it has a DataOutPort in its fan-in.

  If the instance on which the DataInPort exists has the keep_active_during_scan_test attribute defined and set to "true", the Tdr sourcing the DataInPort is inserted behind the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Sib(sri) so that it is kept active and not part of the scan chain during the scan test modes. If the keep_active_during_scan_test attribute is “false”, then the Tdr sourcing the DataInPort is inserted behind the Sib(sti) so that it can be scan tested along with the rest of the functional logic.

The reset value of the TDR bits are determined first by the constant tie value of the design pin, or second by the ICL attribute default_load_value of the ICL description of that port if the design pin is not tied. The reset value defaults to 0 if none of these conditions are found.

You can use the tessent_enable_group attribute to group the DataInPorts into arbitrary groups. One Tdr is inserted per enable group enabling the multiplexers to be switched to the IJTAG side independently for each group to take control of one group of ports while leaving the others under functional control.

You can also use the tessent_common_tdr_source attribute to drive the DataInPorts from a single TDR bit. Every unique source name creates a new DataOutPort on a single TDR across all design instances. If the tessent_enable_group is also specified, a new TDR is created for each group to drive all the common sources of that group.

- **DataOutPorts** — A DataOutPort is referenced by a connection property inside the Tdr/DataInPorts wrapper when it is to be observed by the network. It is not referenced by the current IjtagNetwork when there is already a DataInPort in its fanout or the DataOutPort has the icl_port attribute tessent_use_in_dft_specification set to “false”. You can use the set_ijtag_instance_options command to force its inclusion or exclusion independent of whether it has a DataInPort in its fanout.

  If the instance on which the DataOutPort exists has the icl_port attribute defined and set to true, the Tdr observing the DataOutPort is inserted behind the Sib(sri) so that it is kept active and not part of the scan chain during the scan test modes. If the keep_active_during_scan_test attribute is “false”, then the Tdr observing the DataOutPort is inserted below the Sib(sti) so that it can be scan tested along with the rest of the functional logic.

Licensing

The tool acquires the necessary Tessent product licenses by means of “set_context dft -license”, or as implied by your set_dft_specification_requirements command settings. However, if your intent is to insert only an IJTAG network, then the tool requires any one of the following licenses: IJTAG, MemoryBIST-LV, MemoryBIST-LVNS, MemoryBIST-TS, LogicBIST, TestKompress, or MissionMode.

The TESSENT_LICENSE_ORDER environment variable modifies the priority and order of the available licenses. Any available licenses not explicitly listed in the value of the TESSENT_LICENSE_ORDER environment variable are appended to the list in their original order.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can refer to the set_license_queue_timeout and get_license_queue_timeout commands for more information about the license queuing feature.

Arguments

- -existing_ijtag_host_scan_in *host_scan_in_design_pin_spec*

  An optional switch and value pair that defines an existing host scan interface to which the entire IjtagNetwork should attach inside the created DftSpecification. The design pin must be on an instance having an associated ICL module and the pin must be a ScanInPort referenced by a Host ScanInterface wrapper. Specify this option during the first insertion pass. By default, the interface to the IJTAG network is determined using several criteria. When it is not the first insertion pass into the current design and either the Sib(sti) or the Sib(sri) is present, the tool automatically uses it as the connection point for the current insertion pass, as shown in step 2 of the example below. If it is the first insertion pass, the tool inserts a TAP to host the root level Sib ring when design level is "chip". The tool connects the root level Sib ring to ports when at the physical_block or sub_block level. Use the set_design_level command to specify the design level as chip, physical_block, or sub_block.

- -existing_primary_tap_scan_out *primary_tap_client_scan_out_design_pin_spec*

  An optional switch and value pair specifying that a current TAP client interface is to be used as primary instead of creating a new one. It can only be used when the design level is "chip".

  If other TAP or ijtag client interfaces, or if the dft requirements need additional ijtag nodes and this option is specified, you must also specify the -existing_ijtag_host_scan_in switch and value pair.

  If the option is not used, the tool creates a TAP as the primary TAP and additional TAP client interfaces already in the design are muxed — refer to "Tap" on page 4012 for more information. Each additional TAP client interface adds one scanMux and one DataOutPort in a new TDR (only a single TDR for many TAP client interface, 1 bit for each TAP.)

- -existing_bscan_host_scan_in *bscan_host_scan_in_design_pin_spec*

  An optional switch and value pair that defines an existing host bscan interface to which the boundary scan chain is to connect to. You typically do not need to specify this option. This switch value and value pair is mutually exclusive with the following switches:

  - -active_high_compliance_enables
  - -active_low_compliance_enables

  If no TAP already exists in the design and the set_design_level command was used to specify level "chip", a Tap wrapper is automatically inserted in the IjtagNetwork wrapper to host both the ijtag network and the boundary scan chain. If a TAP controller was inserted in a previous insertion pass and you did not delete the HostBscan wrapper, it is automatically seen and the boundary scan chain is automatically hooked up to it. If you have your own third-party TAP controller and you described it with an ICL file as described in section BoundaryScan it too is recognized by the command and the boundary scan chain is automatically hooked up to it.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You specify this option if you have more than one TAP controller already present in the design that meets the requirements described in "BoundaryScan" on page 4153. You also need to specify this option if the ICL description of your TAP does not call the ScanInterface associated with the host bscan interface named "bscan" exactly.

For example, the LogicVision TAP can be used to host the boundary scan chain, but if it was generated prior to version 2014.2, the ScanInterface for the boundary scan chain was not called "bscan". In this case, you need to point to the "fromBscan" pin of the TAP using the -existing_bscan_host_scan_in option tin order to have create_dft_specification command connect the boundary scan register to it.

- -tile_ijtag_host_list *tile_ijtag_hosts*

  An optional switch and value pair that specifies a list of tile host ports to add to the IJTAG network of this insertion pass. The tool adds a tile host collector when you use this switch. Use this option if the current design is a tile that should connect to neighboring tiles.

  The *tile_ijtag_hosts* value is a Tcl list of names. The order of the list is important. The tool daisy-chains the hosts in the order found in the list, where index 0 of the list is on the scan_out side, and the maximum index is on the scan_in side. The tool adds a pipelining host SIB for each host port.

  See Example 3 of IjtagNetwork for usage and an example of processing the C2 and C3 tiles.

- -stap_host_list *stap_nodes*

  An optional switch and value pair that specifies a list of STAP nodes for the tool to add within either the Tap/HostSTap wrapper or a dedicated IjtagNetwork/HostScanInterface wrapper. The tool inserts the STAP nodes within the Tap/HostSTap wrapper when it creates the main HostScanInterface that contains a Tap wrapper. Otherwise, the tool inserts the STAP nodes in a dedicated HostScanInterface(stap) wrapper. When the tool creates a dedicated HostScanInterface wrapper, it populates the Interface wrapper port names with the default port names specified in the IjtagNetwork/HostScanInterface/STap wrapper.

  The *stap_nodes* value is a TCL list of strings that contains any combination of *id*, pipe(*id*), or sec_host(*id*) node entries. The tool orders the specified nodes with scan_in on the left and scan_out on the right. See Example 1 and Example 2 in Tap/HostSTap for usage of this switch. Valid *stap_nodes* values are as follows:

  - *id* — Specifies the creation of a STap(*id*) wrapper. If you specify this as the right-most element of the list and specify the set_dft_specification_requirements -design_type switch as tile, the tool sets the STap(*id*)/so_retiming property to off. This setting provides a full TCK period for the interface loop timing.
  - pipe(*id*) — Specifies the creation of a STapPipeline(*id*) wrapper. If you specify this as the right-most element of the list and specify the set_dft_specification_requirements -design_type switch as tile, the tool sets the STapPipeline/so_retiming property to off. This setting provides a full TCK period for the interface loop timing.
  - sec_host(*id*) — Specifies the creation of a SecondaryHostScanInterface(*id*) wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -active_high_compliance_enables *enable_port_name* ...

  An optional switch and value pair that defines one or more ports that need to be set to 1 in order to access the TAP scan path that are created with the resulting DftSpecification.

  Use this option to create a Tessent TAP that shares the ports with a pre-existing connected TAP, which may or may not have an ICL description. This option can only be specified when design level is "chip" and cannot be used with -existing_bscan_host_scan_in, -existing_ijtag_host_scan_in, or -existing_primary_tap_scan_out options.

  For more information, refer to the HostScanInterface/Interface/ComplianceEnable wrapper's active_high_enables property and also Example 3 of the IjtagNetwork/ HostScanInterface/Interface wrapper section.

  Note

  If the compliance enabling logic is already present in your design and you want to connect the TAP to it, follow the instructions described in Example 2 of the IjtagNetwork wrapper section.

- -active_low_compliance_enables *enable_port_name* ...

  An optional switch and value pair that defines one or more ports that need to be set to 0 in order to access the TAP scan path that are created with the resulting DftSpecification.

  Use this option to create a Tessent TAP that shares the ports with a pre-existing connected TAP that may or may not have an ICL description. This option can only be specified when design level is "chip" and cannot be used with -existing_bscan_host_scan_in, -existing_ijtag_host_scan_in, or -existing_primary_tap_scan_out options.

  For more information, refer to the HostScanInterface/Interface/ComplianceEnable wrapper's active_low_enables property and also Example 3 of the IjtagNetwork/ HostScanInterface/Interface wrapper section.

  Note

  If the compliance enabling logic is already present in your design and you want to connect the TAP to it, follow the instructions described in Example 2 of the IjtagNetwork wrapper section.

- -sri_sib_list *sri_sib_list*

  An optional switch and value pair that specifies a list of string ID that is used to create Sib(id) wrappers in the created IjtagNetwork wrapper. The Scan Resource Instrument host Sib is created if not already present. If it is already present, it is referenced using the HostScanInterface wrapper and the requested Sibs are inserted below it. This option is useful if you want to add your own instrument inside the network and this instrument needs to remain active during scan test. This option provides the host for your instrument in the created DftSpecification. This option is also useful to create a host for the OCC controllers. You run create_dft_specification with -sri_sib_list occ option, then you simply specify the host_scan_interface property in the OCC wrapper to Sib(occ) as shown in Example 2 below.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -sti_sib_list *sti_sib_list*

  An optional switch and value pair that specifies a list of string ID that is used to create Sib(id) wrappers in the created IjtagNetwork wrapper. The Scan Tested Instrument host Sib is created if not already present. If it is already present, it is referenced using the HostScanInterface wrapper and the requested Sibs are inserted below it. This option is useful if you want to add your own instrument inside the network and this instrument is to be tested as normal functional logic during scan test. This option provides the host for your instrument in the created DftSpecification.

- -replace

  An optional switch that suppresses the error that normally occurs if the DftSpecification(*design_name*,id) configuration wrapper already exists in memory. Instead, the existing one is deleted and replaced by the newly created one.

Examples

Example 1

The following example illustrates the created DftSpecification that is generated by the create_dft_specification command when running on different modules of a chip. The examples below are useful to understand how instruments are connected to the IjtagNetwork in a multi-insertion pass scenario.

Step 1

The first run is performed on a core called CoreA to create and hook up instruments to the network. These instruments are scan testable (keep_active_during_scan_test is “false”). It is the first dft insertion pass involving the IjtagNetwork in that module so the tool creates the Sib(sti) in this run.

```
set_context dft -rtl
#commands to load design appears here

set_current_design CoreA
set_design_level physical_block

set_system_mode analysis

set spec [create_dft_specification]
```

The generated DftSpecification is shown below. The Sib(sti) is identified with the “tessent_dft_function” attribute set to “scan_tested_instrument_host”. All local scan interfaces are daisy chained below the Sib(sti_local).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(CoreA,rtl) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Sib(sti) {
        Attributes {
          tessent_dft_function : scan_tested_instrument_host;
        }
        Sib(sti_local) {
          DesignInstance(tdr_1) {
            scan_interface : client;
          }
          DesignInstance(tdr_1) {
            scan_interface : client;
          }
        }
      }
    }
  }
}
```

Step 2

The second run is again performed on the core called CoreA at a later time to insert a Tdr used to control DataInBits on a scan control module that was user-inserted after the original insertion phase. The scan control module has an associated ICL file that documents its DataInPorts and with the ICL module attribute keep_active_during_scan_test set to “true”. Notice that because this is a second insertion pass into the same design module, the *design_id* is specified as rtl2 in the set_context command to differentiate the insertion pass from the previous one. An error would have been generated had it not been specified because a conflict would have been noticed from the existence of instances of module names whose names start with “CoreA_rtl_tessent_”.

```
set_context dft -rtl -design_id rtl2
#commands to load design appears here

set_current_design CoreA
set_design_level physical_block

set_system_mode analysis

set spec [create_dft_specification]
```

The generated DftSpecification is shown below. The Interface wrapper of the HostScanInterface wrapper is used to specify that the new Sib(sri) is to be inserted in series with the client interface of the Sib instance “CoreA_rtl_tessent_sib_sti_inst”, which is the Sib(sti) created in the first insertion pass.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(CoreA,rtl2) {
  IjtagNetwork {
    HostScanInterface(sri) {
      Interface {
        design_instance : CoreA_rtl_tessent_sib_sti_inst;
        scan_interface : client;
      }
      Sib(sri) {
        Attributes {
          tessent_dft_function : scan_resource_instrument_host;
        }
        Sib(sri_local) {
          Tdr(sri_tdr1) {
            DataOutPorts {
              connection(0) : scan_control_inst/tm;
              connection(1) : scan_control_inst/ac_mode;
              connection(3:2) : scan_control_inst/config[1:0];
            }
          }
        }
      }
    }
  }
}
```

Step 3

The third run is performed on the chip called ChipA. It is used to provide access to the IJTAG scan interface of the 2 instances of CoreA and to again insert a Tdr to control the data_in ports of the scan control module situated at the top level. A Tap wrapper is created as the -existing_ijtag_host_scan_in option was not specified and the Tap from a previous insertion pass was not seen as this is the first insertion pass at the chip level.

```
set_context dft -rtl
#commands to load design appear here

set_current_design ChipA
set_design_level chip
set_attribute_value [get_ports TCK_p] -name function -value tck
set_attribute_value [get_ports TDI_p] -name function -value tdi
set_attribute_value [get_ports TDO_p] -name function -value tdo
set_attribute_value [get_ports TMS_p] -name function -value tms
set_attribute_value [get_ports TRST_p] -name function -value trst
set_system_mode analysis

set spec [create_dft_specification]
```

The generated DftSpecification is shown below. Note how a Tap wrapper is being specified. It includes a HostBscan wrapper such that the TAP is ready for boundary scan-in a future insertion pass. One sib is inserted into the root level Sib ring to interface to the two instances of CoreA, and to provide access to the Tdr that is inserted to control the DataIn ports of the scan control module. Each child physical block is also wrapped by a local Sib to make sure other parts of the network are always accessible even if the physical block is defective, powered down, or in simulation, black-boxed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(ChipA,rtl) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Interface {
        tck  : TCK_p;
        trst : TRST_p;
        tms  : TMS_p;
        tdi  : TDI_p;
        tdo  : TDO_p;
      }
      Tap(main) {
        HostBscan {
        }
        HostIjtag(1) {
          Sib(sri) {
            Attributes {
              tessent_dft_function : scan_resource_instrument_host;
            }
            Sib(pb1) {
              DesignInstance(CoreA_I1) {
                scan_interface : ijtag;
              }
            }
            Sib(pb2) {
              DesignInstance(CoreA_I2) {
                scan_interface : ijtag;
              }
            }
            Sib(sri_local) {
              Tdr(sri_tdr1) {
                DataOutPorts {
                  connection(0) : scan_control_inst/tm;
                  connection(1) : scan_control_inst/ac_mode;
                  connection(2) : scan_control_inst/config2;
                }
              }
            }
          }
        }
      }
    }
  }
}
```

Step 3 under Example 1 could have been modified to connect to an existing TAP controller by simply using the -existing_ijtag_host_scan_in option as follows:

create_dft_specification -existing_ijtag_host_scan_in mytap/from_bist0

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The created DftSpecification would then be as follows:

```
DftSpecification(ChipA,rtl) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Interface {
        DesignInstance (mytap)
        scan_interface : BistPort0;
      }
      Sib(sri) {
        Attributes {
          tessent_dft_function : scan_resource_instrument_host;
        }
        Sib(pb1) {
          DesignInstance(CoreA_I1) {
            scan_interface : ijtag;
          }
        }
        Sib(pb2) {
          DesignInstance(CoreA_I2) {
            scan_interface : ijtag;
          }
        }
        Sib(sri_local) {
          Tdr(sri_tdr1) {
            DataOutPorts {
              connection(0) : scan_control_inst/tm;
              connection(1) : scan_control_inst/ac_mode;
              connection(2) : scan_control_inst/config2;
            }
          }
        }
      }
    }
  }
}
```

Example 2

This example shows how the -sri_sib_list option is used to create a host node for the OCC wrapper. The OCC wrapper makes reference back to the Sib(occ) using the ijtag_host_interface property.

```
set spec [create_dft_spec -sri_sib_list occ]
report_config_data $spec
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(corea,gate) {
  use_rtl_cells : on;
  IjtagNetwork {
    ImplementationOptions {
      scan_path_retiming : flop;
    }
    HostScanInterface(ijtag) {
      Sib(sti) {
        Attributes {
          tessent_dft_function : scan_tested_instrument_host;
        }
      Sib(sri) {
        Attributes {
          tessent_dft_function : scan_resource_instrument_host;
        }
        Sib(sri_ctrl) {
          Tdr(sri_ctrl) {
            Attributes {
              tessent_dft_function : scan_resource_instrument_dft_control;
            }
          }
        }
        Sib(occ) {
        }
      }
    }
  }
}
```

```
read_config_data -in_wrapper $spec -from_string {
 Occ {
  ijtag_host_interface : Sib(occ);
  Controller(c1) {
   clock_intercept_node : clk;
  }
 }
}
```

Example 3

The following example runs a script to push down each Tdr wrapper below its own Sib wrapper when there are more than one below the Sib(sti_local) or Sib(sri_local) wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
proc push_tdr_below_sibs {spec} {
# Only works if the spec was created by create_dft_specification.
  set sib_locals [get_config_elements Sib(*_local) -below $spec -hier -silent]
  foreach_in_collection sib_local $sib_locals {
    set tdrs [get_config_element Tdr -below $sib_locals -silent]
    if {[sizeof_collection $tdrs] > 1} {
      foreach_in_collection tdr $tdrs {
        set id [get_config_value $tdr -id <0>]
        set sib [add_config_element Sib(sib_${id}) -in $sib_local]
        #Place New Sib in same parent instance as the TDR
        unset pin
        if {[get_config_value DataOutPorts/connection(0) -in $tdr -exist]} {
          set pin [get_config_value DataOutPorts/connection(0) -in $tdr]
        } elseif {[get_config_value DataInPorts/connection(0) -in $tdr -exist]} {
          set pin [get_config_value DataInPorts/connection(0) -in $tdr]
        }
        if {[info exists pin]} {
          set parent_instance [get_attribute_value [get_pins [list $pin]] \
            -name parent_instance]
        }
        if {$parent_instance ne ""} {
          set_config_value parent_instance -in $sib $parent_instance
          set_config_value parent_instance -in $tdr $parent_instance
          move_config_element \
           [get_config_value parent_instance -in $tdr -object]-first
        }
        move_config_element $tdr -in $sib
      }
    }
  }
}
set spec [create_dft_specification]
push_tdr_below_sibs $spec
```

Here is the Tdr wrapper before and after it was pushed below its own Sib Wrapper.

Before:

```
Tdr(sri_tdr4) {
  DataOutPorts {
    connection(0) : sub1_i2/sub2_i2/sub3_i1/di[0];
    connection(1) : sub1_i2/sub2_i2/sub3_i1/di[1];
    connection(2) : sub1_i2/sub2_i2/sub3_i1/di[2];
    connection(3) : sub1_i2/sub2_i2/sub3_i1/di[3];
    connection(4) : sub1_i2/sub2_i2/sub3_i1/di[4];
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

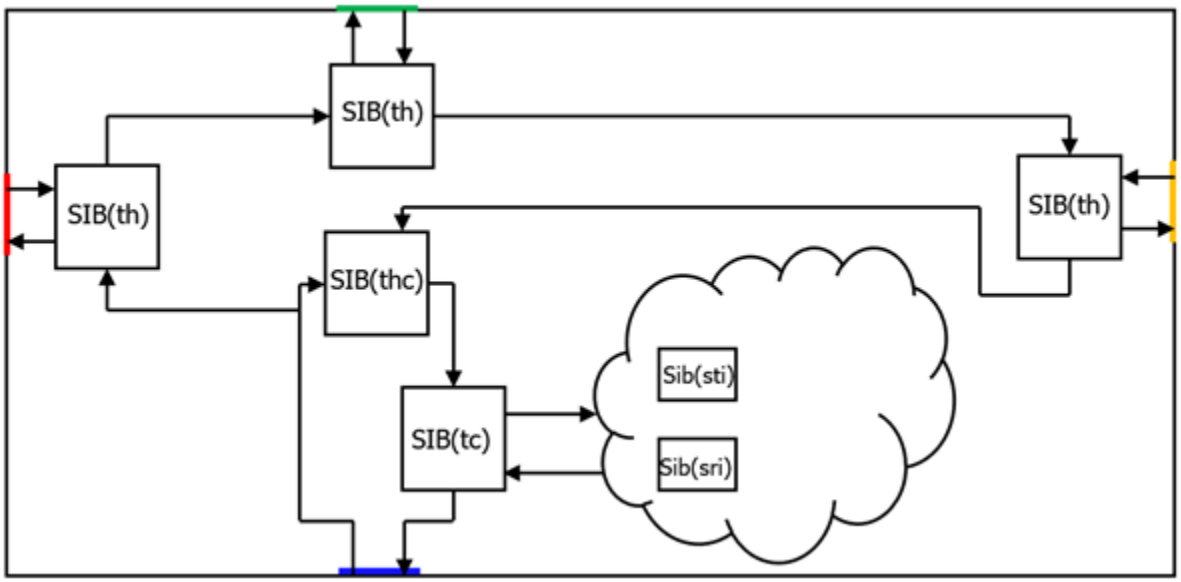
After:

```
Sib(sib_sri_tdr4) {
  parent_instance : sub1_i2/sub2_i2;
  Tdr(sri_tdr4) {
    parent_instance : sub1_i2/sub2_i2;
    DataOutPorts {
      connection(0) : sub1_i2/sub2_i2/sub3_i1/di[0];
      connection(1) : sub1_i2/sub2_i2/sub3_i1/di[1];
      connection(2) : sub1_i2/sub2_i2/sub3_i1/di[2];
      connection(3) : sub1_i2/sub2_i2/sub3_i1/di[3];
      connection(4) : sub1_i2/sub2_i2/sub3_i1/di[4];
    }
  }
```

Example 4

The following example shows how to use the "-tile_ijtag_host_list *tile_ijtag_hosts*" switch to create a DftSpecification suitable for a tiled design. In Figure 3-31, SIB(tc) is the tile client for all IJTAG instruments in the core. It has a scan-in buffer stage but does not have a scan-out retiming latch. SIB(tc) connects in series with the tile host collector SIB(thc). The tool creates a tile host SIB(th) for each new tile host port specified in the *tile_ijtag_hosts* list. Each SIB(th) has a scan-in pipelining stage hosted by the tile host controller SIB(thc).

Figure 3-31. Design With IJTAG Host Tiling




```
set_dft_specification_requirements -design_type tile
check_design_rules
set spec [create_dft_specification -tile_ijtag_host_list {left top right}]
report_config_data $spec
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(tile_core,gate) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Sib(tc) {
        Attributes {
          tessent_dft_function : tile_client_sib;
        }
        to_scan_in_feedthrough : pipeline;
        so_retiming : off;
        Sib(sti) {
          [...]
        }
        Sib(sri) {
          [...]
        }
      }
      Sib(thc) {
        Attributes {
          tessent_dft_function : tile_host_collector;
        }
        Sib(th_right) {
          to_scan_in_feedthrough : pipeline;
          SecondaryHostScanInterface(right) {
          }
        }
        Sib(th_top) {
          to_scan_in_feedthrough : pipeline;
          SecondaryHostScanInterface(top) {
          }
        }
        Sib(th_left) {
          to_scan_in_feedthrough : pipeline;
          SecondaryHostScanInterface(left) {
          }
        }
      }
    }
  }
}
```

Related Topics

apply_specification_defaults

process_dft_specification

report_config_syntax

set_design_level

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_diagnosis_patterns

Context: patterns -scan_diagnosis

Mode: analysis

Creates targeted patterns for logic failure or scan chain failure iterative diagnosis.

Usage

Logic Failure Diagnosis

create_diagnosis_patterns [-DIAGnosis_report {*diagnosis_report1 diagnosis_report2 ...*}] [-MAX_pattern_count *number*] [{-TRAnsition | -STUCK}]

Scan Chain Failure Diagnosis

create_diagnosis_patterns [-DIAGnosis_report {*diagnosis_report*}] [-CHAIN *chain_name* [-CELL_range *lowerbound upperbound*] [-FAULT_type {stuck_at_0 | stuck_at_1 | slow_to_rise | slow_to_fall | slow | fast_to_rise | fast_to_fall | fast}]]

Description

For usage cases using this command, refer to "Iterative Logic Diagnosis Examples" in the *Tessent Diagnosis User's Manual*. You can use iterative diagnosis with the layout-aware diagnosis flow as long as you use the layout in the same iterative diagnosis session—see "Layout-Aware Diagnosis" in the *Tessent Diagnosis User's Manual*.

Note

After issuing this command, you should write out the test patterns to an external file using the write_patterns command.

Logic Failure Diagnosis: The tool loads the diagnosis result from a prior run or a diagnosis report file, create a fault list, and generates the patterns targeting suspect signals in the diagnosis report.

Scan Chain Failure Diagnosis: The tool loads either of the following inputs you specify:

- A diagnosis report file containing the diagnosis results of scan chain failure.
- The chain names, scan cell range, and the fault type.

For scan chain failure diagnosis, you can also provide a chain name (using the -chain switch) and a diagnosis report containing multiple faulty scan chains. In this case, the diagnostic test generation only creates patterns for the scan chain specified by the -chain switch.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

Logic Failure Diagnosis Arguments

- -DIAGnosis_report {*diagnosis_report1 diagnosis_report2 ...*}

  An optional switch and one or more strings that specifies the diagnosis report file(s) to be targeted by diagnostic ATPG. By default, the tool uses the diagnosis results from previous diagnosis run. You can also supply more than one diagnosis report for suspects from multiple diagnosis report files are targeted in one diagnosis test generation run. The diagnosis report must be in the default ASCII format.

- -MAX_pattern_count *number*

  An optional switch and integer pair that specifies the maximum pattern count the tool generates. The default is no limit. Use when ATPG run time is a concern. The limit defined by -MAX_pattern_count is applied per chain if used for chain pattern generation.

- -TRAnsition

  An optional switch that specifies creating transition test patterns. By default, the transition patterns generated are launch off capture (equivalent to the old no_shiftlaunch option).

- -STUCK

  An optional switch that specifies creating create stuck-at diagnosis patterns.

Scan Chain Failure Diagnosis Arguments

- -DIAGnosis_report {*diagnosis_report*}

  An optional switch and string pair that specifies the diagnosis report file to be targeted by diagnostic ATPG. By default, the tool uses the diagnosis results from previous diagnosis run. Note that only a single diagnosis report is permitted as input for chain failures.

- -CHAIN *chain_name*

  An optional argument and string pair that specifies a single chain.

- -CELL_range *lowerbound upperbound*

  An optional argument and pair of integers for the -chain option that specifies the beginning and ending of a range of cells.

- -FAULT_type {stuck_at_0 | stuck_at_1 | slow_to_rise | slow_to_fall | slow | fast_to_rise | fast_to_fall | fast}

  An optional argument and literal pair for the -chain option that specifies the fault type.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_feature_statistics

Context: patterns -scan_diagnosis

Mode: analysis

Calculates the root cause deconvolution (RCD) constants from the LDB for RCD diagnosis and analysis.

Usage

create_feature_statistics [-temp_dir *dir_name*] [-add_processors {*hostname*[:{*cpus* | MAXcpu}]}…] [-force]

Description

The tool analyzes the design for all possible diagnosis suspects for certain features in the layout. It determines which bridge pairs, net segments, and cells are tested, and extracts the relevant physical information such as the critical area, via count, and cell instance count for each possible suspect.

The tool also performs threshold ratio checking and returns information to warn you about low detection of the RCD root causes by fault simulation. This could indicate problems with the pattern file. The information is only issued if at least one cell, open, or bridge detected ratio is less than 10 percent.

You must create the RCD constants statistics for every flat model and pattern combination. To do this, you can process the combinations in parallel in different tool sessions.

RCD analysis in Tessent YieldInsight determines the probability sums for the features across the diagnoses and uses the RCD constants to normalize their weights to account for diagnosis noise.

To use this command, you must have previously loaded your patterns and opened a LDB. Otherwise, the tool issues an error message. The feature statistics are stored in the LDB for later use by Tessent YieldInsight RCD analysis. They are not used during the diagnosis process.

For more information, see "Diagnosis for Root Cause Deconvolution Analysis" in the *Tessent Diagnosis User's Guide*.

Arguments

- -temp_dir *dir_name*

  An optional switch and string pair that specifies a temporary directory to use for command execution.

- -add_processors {*hostname*[:{*cpus*| MAXcpu}]}…

  An optional switch and string that specifies to run multiple processors in parallel on multiple machines to reduce the runtime for calculating the RCD constants. The tool calls the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_processors command and uses a multithreading process as described for the add_processors command.

Although the -add_processors switch supports the same options as the add_processors command, for the best performance, specify the localhost with 8 processors.

- -force

  An optional switch that specifies to delete from the LDB the temp file or .rcddb file associated with the current flat model before attempting to generate feature statistics. This enables you to re-run RCD creation in the event of a premature termination, which can result in temporary locked files. The tool deletes the temp files, issues a warning and continues processing.

Examples

Example 1

The following example loads patterns and opens the layout before calculating the feature statistics.

read_patterns external my_test_patterns.wgl

```
//  Reading WGL input file "my_test_patterns.wgl"
```

open_layout my_layout.db

```
//  DieArea: minx=-2107.000000, miny=-2002.000000, maxx=2107.000000,
//    maxy=2002.000000 in microns
//  Note: Checking layout consistency...
//  Note: Layout Database opened successfully.
```

create_feature_statistics

```
//  Determining feature detection information
//  10% Done.
//  22% Done.
……
//  100% Done.
//  Determining Open Feature Statistics
//  11% Done.
//  21% Done.
……
//  100% Done.
//  Determining Bridge Feature Statistics
//  13% Done.
……
//  Feature Statistics Calculation Complete. Information is written into
//  the layout database.
```

Example 2

The following example shows the message that the tool returns when you issue the create_feature_statistics when the feature statistics were previously calculated and stored in the LDB.

create_feature_statistics

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Note: Feature statistics for this flat model and pattern set have
//  already been calculated and stored in the layout database. There is no
//  need to perform this operation again.
```

Example 3

The following example shows the information that the tool returns when threshold ratio checking results in one or more cell, open, or bridge detected ratios that are less than 10 percent.

```
//    detected cell ratio: 48.867
//    detected open ratio: 26.5447
//  detected bridge ratio: 81.2205
//         detected cells: 4162 opens: 19813 bridges: 60623
//            total cells: 8517 total nets: 74640
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_flat_model

Context: all contexts

Mode: setup

Creates a primitive gate simulation representation of the design. The command also flattens the hierarchy of the design so that the design can be viewed in the Tessent Visualizer Flat Schematic window.

Usage

create_flat_model

Description

The tool automatically flattens the design hierarchy down to the logically equivalent design when you exit Setup mode. However, there may be times that you would like to access the flat model without having to exit Setup mode. For example, you may want to add ATPG constraints and functions before you exit Setup mode.

If you exit Setup mode and then add ATPG constraints and functions, the design rule checker does not have access to those ATPG constraints during the rule checking. If you issue the create_flat_model command in Setup mode and then add those ATPG constraints, the design rule checker has access to them during the rule checking.

Note

If you are preparing a flat model for later use in Tessent Diagnosis, save the flat model immediately after performing ATPG (issuing the create_patterns command) and saving final patterns. For more information, refer to "Preparing the Design Netlist" in the *Tessent Diagnosis User's Manual*.

Arguments

None

Examples

The following example shows flattening the design to the simulation primitives before adding constraints that the rule checker then uses when you run the design rule checker. The rule checker runs when you first attempt to exit Setup mode

```
create_flat_model
add_atpg_functions and_b_in and /i$144/q /i$141/q /i$142/q
add_atpg_constraints 0 /i$135/q
add_atpg_constraints 1 and_b_in
set_system_mode analysis
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following example sets the system mode to setup, creates a flat model, and opens Tessent Visualizer:

```
set_system_mode setup
create_flat_model
open_visualizer
```

Related Topics

add_atpg_constraints

delete_atpg_functions

set_system_mode

write_flat_model

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_icl_flush_pattern

Context: patterns -silicon_insight -ijtag

Mode: setup

Creates an ICL flush pattern.

Usage

create_icl_flush_pattern ***pattern_name*** [-pattern_type {stil | svf}] [-bit_pattern {000..1 | 1..000}] [-bit_pattern_length *value*] [-scanout_skew *value*]

Description

Use the flush pattern to test the functionality of the IJTAG network for the configured scan path. This utility command is useful when you are creating your own algorithms to debug ICLNetwork test suspects.

Use this command when manually debugging IJTAG networks as described in "Diagnosing and Debugging IJTAG Networks" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

This command creates a pattern file that flushes the specified bit pattern. The pattern is saved in a pattern file named *pattern_name.pattern_type*.

Arguments

- ***pattern_name***

  A required string that specifies the name of the flush pattern to be generated.

- -pattern_type {stil | svf}

  An optional switch and literal pair that specifies the pattern type: STIL or SVF. The default is SVF.

- -bit_pattern {000..1 | 1..000}

  An optional switch and literal pair that specifies the type of bit pattern to generate. Two bit pattern types are supported: 000..1 and 1..000. The default is 000..1.

- -bit_pattern_length *value*

  An optional switch and value pair that specifies the length of the bit pattern to generate. The value must not be less than the current length of the configured scan path. The length is normally the same length of the scan path, which you can verify using the get_icl_network -current_scan_path command.

- -scanout_skew *value*

  An optional switch and value pair that skews the "1" in the specified bit pattern accordingly for the scanned-out bit pattern. If the bit pattern is "000..1", the "1" s shifted to the left by the specified skew value. If the bit pattern is "1..000", the "1" is shifted to the right by the specified skew value.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

create_icl_flush_pattern my_flush

**create_icl_flush_pattern my_flush_2 -bit_pattern 1..000 -bit_pattern_length 145 \
-scanout_skew 5**

Related Topics

analyze_icl_suspect

assert_icl_sib

create_icl_setup_pattern

deassert_icl_sib

get_icl_network

get_icl_scan_mux

isolate_icl_node

report_icl_network

set_icl_network

set_icl_scan_mux

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_icl_setup_pattern

Context: patterns -silicon_insight -ijtag

Mode: setup

Creates an ICL pattern to set up the scan path of interest in the current ICL network.

Usage

create_icl_setup_pattern ***pattern_name*** [-pattern_type {stil | svf}]

Description

When debugging an ICL network, use this command to generate a pattern file that, when you run it, sets up the ICL network to use the scan path that you have previously configured. This utility command is useful when you are creating your own algorithms to debug ICLNetwork test suspects.

Use this command when manually debugging IJTAG networks as described in "Diagnosing and Debugging IJTAG Networks" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

The pattern is saved in a pattern file named *pattern_name.pattern_type*.

Arguments

- ***pattern_name***

  A required string that specifies the name of the setup pattern to be generated.

- -pattern_type {stil | svf}

  An optional switch and literal pair that specifies the pattern type: STIL or SVF. The default is SVF.

Related Topics

analyze_icl_suspect

assert_icl_sib

create_icl_flush_pattern

deassert_icl_sib

get_icl_network

get_icl_scan_mux

isolate_icl_node

report_icl_network

set_icl_network

set_icl_scan_mux

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_icl_verification_patterns

Context: patterns -ijtag

Mode: analysis

Prerequisites: You must open a pattern set with the open_pattern_set command to use this command.

Specifies the creation of a structural pattern to verify the proper operation of the IJTAG network.

Usage

create_icl_verification_patterns [-scan_test {on | off}]
[-data_pin_test {on | off | end_points_only}]
[-modules *modules*] [-exclude_modules *exclude_modules*]
[-instances *instances*] [-exclude_instances *exclude_instances*]
[-exclude_scan_registers *exclude_scan_registers*]
[-exclude_data_pins *exclude_data_pins*]
[-include_branches_through_excluded_elements]

Description

In a simulation, the pattern set is useful for identifying mismatches between the ICL and Verilog model views. In a manufacturing test, the pattern set covers manufacturing defects that can exist in the IJTAG network circuitry. You do not need to use this command if you use the more automated create_patterns_specification/process_patterns_specification flow. For more information about this flow, see the ICLNetworkVerify wrapper.

When you run this command with the -scan_test option set to on, the tool creates verification patterns by tracing backward from all ScanOutPorts and configuring all ScanMuxes found along the way; this exercises all scan configurations.

The tool considers all instances for testing unless given optional information. The tool generates a list of instances to exclude using the information you provide with the -exclude_instances, -exclude_modules, -instances, and -modules switches. This optional list is empty by default.

ICL verification patterns do not target strictly excluded instances. To strictly exclude an instance, assign true as the value of the tessent_ignore_during_icl_verification attribute.

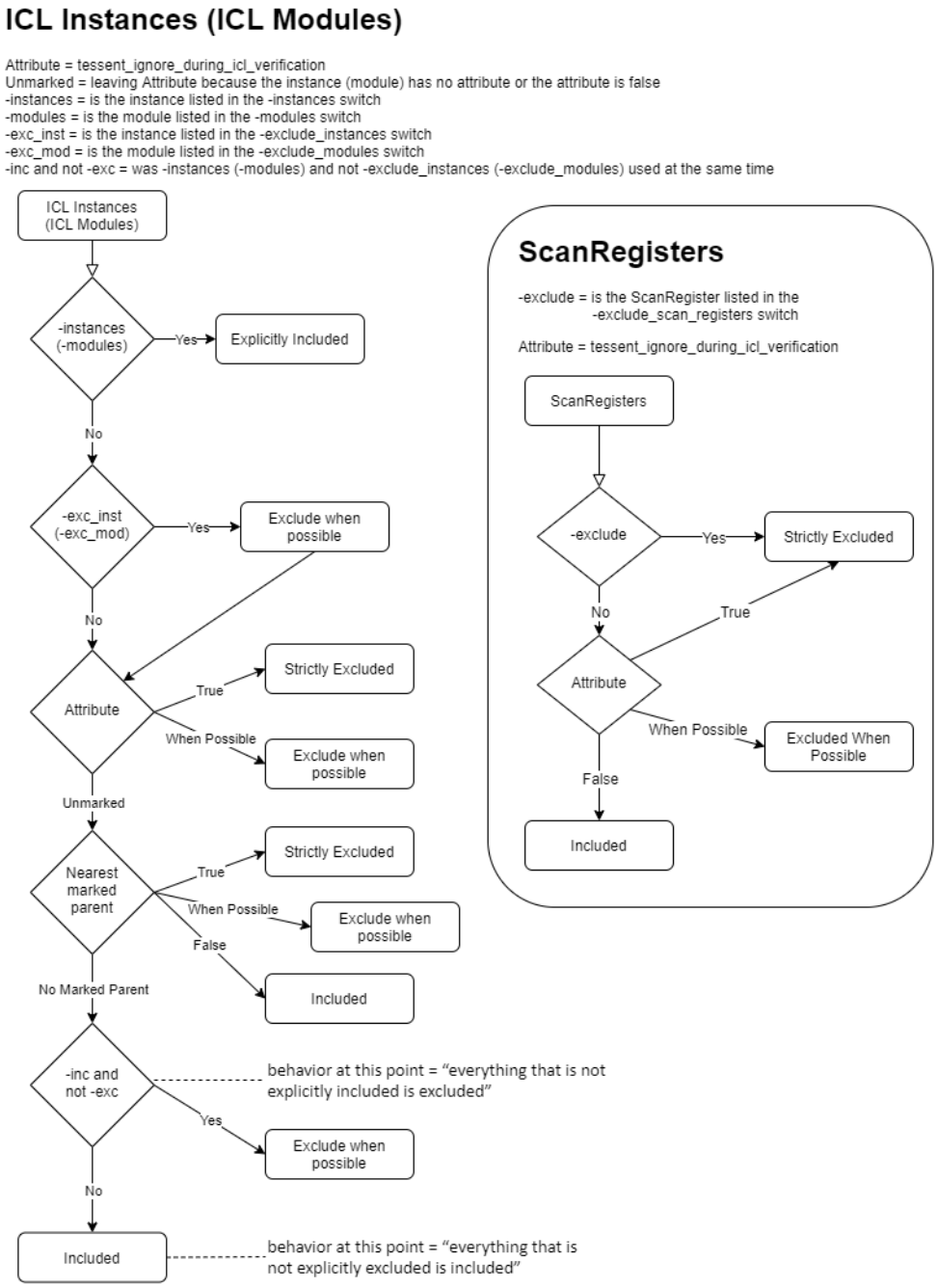
ICL verification patterns do not directly target instances that are marked as excluded when possible. To exclude an instance when possible, use the -exclude_instances switch or assign when_possible as the value of the tessent_ignore_during_icl_verification attribute.

The process to exclude modules uses the -exclude_modules switch or uses the same values, true or when_possible, for the tessent_ignore_during_icl_verification attribute as described to exclude instances.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For example, the tool can use a scan path through an excluded when possible instance A to test an included instance B. However, the tool cannot use any scan path through a strictly excluded instance A to test an included instance B. Figure 3-32 illustrates the flow that generates the exclude list.

Figure 3-32. Flow to Generate an Exclude List



On the first scan load, the predictably captured values of the ScanRegisters are compared, and the scan registers are loaded with a pattern consisting of alternating sequences of 0s and 1s with

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

an increasing sequence length. The TAP controller is then put in the pause state, and the ScanRegisters are unloaded in a second scan load to make sure the same data comes out. This ensures that the ScanRegisters are the length documented in the ICL model.

When you set the -data_pin_test option to on, the tool forces any internal DataOutPorts that do not have a source described in ICL and observes those values by unloading ScanRegisters. The tool controls internal DataInPorts that do not have a destination described in ICL by loading ScanRegisters and observing the resulting value on the internal DataInPorts. This feature is only usable in simulation, where it is useful for finding mismatches between the circuits and their ICL descriptions.

Arguments

- -scan_test {on | off}

  An optional switch that specifies whether the patterns are to verify the integrity of the scan paths. When the option is set to on, which is the default, the tool creates the patterns by tracing backward from all ScanOutPorts and configuring all ScanMuxes found along the way to exercise all scan configurations. On the first scan load, the predictably captured values of the ScanRegisters are compared and the scan registers are loaded with a pattern consisting of alternating sequences of 0s and 1s with an increasing sequence length. The TAP controller is then put in the pause state and the ScanRegisters are unloaded in a second scan load to make sure the same data comes out; this ensures that the ScanRegisters are actually the length documented in the ICL model. While shifting the scan path verification pattern out of the ScanRegisters, the next ICL network configuration is already shifted into the network.

  ```
  load 1011001111000011111111100000000...   expect captures values described in ICL
  load the next ICL network configuration  expect 1011001111000011111111100000000...
  ```

  The ICL verification pattern generator uses two different strategies to create the patterns. First, it attempts to cover all paths with the previously discussed tracing strategy. If any ICL ScanRegisters cannot be covered with this method, the ICL verification pattern generator will attempt to target them with individual scan loads for testing.

- -data_pin_test {on | off | end_points_only}

  An optional switch that specifies whether the patterns are to verify the values observed by Internal DataInPorts and controlled by internal DataOutPorts.

  on — With this value, not only are the DataInPorts with no destination and the DataOutPorts with no source observed and controlled, but all DataInPorts having an observation path through the IJTAG network are forced in the testbench and observed by the IJTAG network to confirm that the ICL and HDL descriptions are consistent. Also, all DataInPorts having some controllability from the network are sensitized to a one and zero value and that value is directly observed on the DataInPort in the testbench.

  off — With this value, no force and compare are done on the DataInPorts and DataOutPorts.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

end_points_only — With this value, the internal DataOutPorts that do not have a source described in ICL are forced, and the forced values observed using the IJTAG network. The internal DataInPorts that do not have a destination described in ICL are controlled using the IJTAG network, and the resulting value is directly observed on the internal DataInPorts.

Note that this feature is only usable in simulation where it is useful for finding mismatches between the HDL and associated ICL descriptions. For this feature to be usable, the ICL should have been created using extract_icl such that each ICL instance has the tessent_design_instance attribute defined to reflect the original design instance path associated to the ICL instance. Verifying the internal data pins can substantially increase the pattern count as well as the time to generate the verification patterns. The increase depends on the number of DataInPorts and DataOutPorts and the overall ICL network complexity.

Also, note that ICL Extraction sets the tessent_design_instance attribute to the pre-synthesis instance names whenever those names are available. During the creation of the data pin verification patterns, the actual design instance names are ignored. The verification pattern generator only uses the tessent_design_instance attribute to obtain the references to the design pins. Consequently, the rtl design files provided to ICL extraction must be used during the simulation of the verification patterns, at least in case that the synthesis modifies the design hierarchy (for example, because of "generate" loops). To obtain data pin verification patterns that can be simulated using the synthesized netlist with instance names that have been modified by synthesis, you either have to modify the tessent_design_instance attributes accordingly or run ICL Extraction based on the synthesized netlist from the beginning (without providing rtl files to the tool).

- -modules *modules*

  An optional switch that specifies the ICL modules the test should focus on. The modules parameter value contains glob pattern matching ICL module names instantiated below the current ICL design. For more information on glob pattern matching, refer to "Glob and Regular Expression Pattern Matching Syntax" on page 3834.

- -exclude_modules *exclude_modules*

  An optional switch that specifies the ICL modules to exclude from the test. The exclude_modules parameter value contains glob pattern matching ICL module names instantiated below the current ICL design. For more information on glob pattern matching, refer to "Glob and Regular Expression Pattern Matching Syntax" on page 3834.

  ICL modules that have the tessent_ignore_during_icl_verification attribute set to true are automatically excluded. You can use this attribute on ICL modules that are not truly IJTAG compliant and cannot pass the structural test performed in this test. Tessent MemoryBIST ICL modules have the tessent_ignore_during_icl_verification attribute set to true and are automatically excluded from the create_icl_verification_patterns command. If a network verification of the MemoryBIST controller is needed, refer to the PatternsSpecification MemoryBist/run_mode : setup_chain_test property value description.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -instances *instances*

  An optional switch that specifies the only ICL instances the test should be performed on. The instances parameter value contains glob pattern matching ICL instances found below the current ICL design. For more information on glob pattern matching, refer to "Glob and Regular Expression Pattern Matching Syntax" on page 3834.

- -exclude_instances *exclude_instances*

  An optional switch that specifies the set of ICL instances to exclude from the test. The exclude_instances parameter value contains glob pattern matching ICL instances found below the current ICL design. ICL instances that have the tessent_ignore_during_icl_verification attribute set to true are automatically excluded.

- -exclude_scan_registers *exclude_scan_registers*

  An optional switch that specifies the set of ICL scan registers to exclude from the test. The exclude_scan_registers parameter value contains glob pattern matching ICL instances found below the current ICL design. ICL Scan Registers that have the tessent_ignore_during_icl_verification attribute set to true are automatically excluded. This switch is used in the LogicBIST ICL module to exclude the single chain diagnostic registers because they are too long to be simulated serially. This switch is only used when -scan_test is set to on.

- -exclude_data_pins *exclude_data_pins*

  An optional switch that specifies the set of ICL pins to exclude from the data pin test. The exclude_data_pins parameter value contains glob pattern matching ICL pins found below the current ICL design. ICL DataInPorts and DataOutPorts that have the tessent_ignore_during_icl_verification attribute set to true are automatically excluded. This switch is only used when -data_pin_test is set to on.

- -include_branches_through_excluded_elements

  An optional switch that specifies the generation of at least one verification pattern for every possible branch of a ScanMux. When this optional switch is not present, the tool will not test ScanMux branches through excluded elements. For this case, the tool still works to test all included ScanRegisters by any means necessary.

Note

The ICL verification pattern generator attempts to utilize a minimal scan path to reach all ICL instances which are to be tested by ICL verification patterns. This scan path might traverse ICL instances that have been excluded through the -exclude_modules or -exclude_instances switches. ICL modules with the "tessent_ignore_during_icl_verification" attribute are not considered when searching for a minimal scan path.

Examples

The following example loads an ICL file, generates ICL verification patterns, and writes a simulation pattern file. The tool automates processing in the create_patterns_specification and process_patterns_specification flow. (These flows also provide simulation automation using the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

run_testbench_simulations command.) Processing typically does not need to be performed as shown here.

```
set_context patterns -ijtag
read_icl my_instrument.icl
set_current_design
set_system_mode analysis
open_pattern_set icl_verify
create_icl_verification_patterns -data_pin_test on
close_pattern_set
write_pattern icl_verify.v -verilog
```

Related Topics

create_patterns_specification

ICLNetworkVerify

process_patterns_specification

run_testbench_simulations

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_initialization_patterns

Context: dft -edt, patterns -scan

Mode: analysis

Creates RAM initialization patterns and places them in the internal pattern set.

Usage

create_initialization_patterns ***RAM_instance_name*** | ***RAM_gate_id#***

Description

The create_initialization_patterns command creates RAM initialization patterns that write values into the specified RAM. The tool places these patterns into the internal pattern set from which you may save them into a file.

You can identify the RAM by its instance name or gate ID number. An error condition occurs if there is not a single RAM gate inside the instance or if the specified gate is not a RAM. An error condition also occurs if the RAM does not have an initialization file, or if the RAM did not successfully pass stability checking (Design Rules A1 and A6).

The tool creates the initialization patterns by doing an independent test generation for each valid address that has a non-X state on at least one data line. If the test generation aborts, the command terminates with an error message. The patterns contain a measure PO statement, but all values are X. There may also be an unload statement, but all values are X. If the patterns are re-simulated (as in pattern compression), the tool deletes patterns that do not detect faults. For those the tool does not delete, simulation values replace the Xs in the measure PO statement.

Arguments

- ***RAM_instance_name***

  A string that specifies the instance name of the RAM for which you want to create_initialization_patterns.

- ***RAM_gate_id#***

  An integer that specifies the gate identification number of the RAM for which you want to create_initialization_patterns.

Examples

The following example creates RAM initialization patterns for p1.ram/u1, places the patterns into the internal pattern set during the ATPG run, and saves the patterns to a pattern file with the name *patfile*:

```
add_write_controls 0 w1
set_system_mode analysis
add_faults -all
create_initialization_patterns p1.ram/u1
create_patterns
write_patterns patfile
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

read_modelfile

write_modelfile

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_instance

Context: all contexts

Mode: insertion

Instantiates an instance of a module mod_spec inside a design module in the current design.

Usage

create_instance ***hierInstPath*** **-of_module** ***mod_spec***
[-below_instance *instance_object*]
[-allow_instance_uniquification [-instance_uniquification_suffix *string*]]
[-parameter_values *parameter_list*]
[-silent]

Description

If the command succeeds, it returns a collection containing the newly created instance.

The pathname to the location must exist in the design up to the leaf level. If the leaf level already exists and the -allow_instance_uniquification switch is specified, Tessent Shell adds a uniquification suffix to the new instance to create a unique instance name. The uniquification suffix defaults to _# where # is replaced by the first available integer starting from 1. You can specify a different uniquification suffix using the -instance_uniquification_suffix option in the create_instance command or in the set_insertion_options command.

In addition, ports of the module that have the function attribute set to “tie0” or “tie1” are automatically tied on the created instance.

Arguments

- ***hierInstPath***

  A required string that specifies the hierarchical path in which the instance is to be created concatenated to the leaf instance name of the new instance. The specified hierarchical path in which the instance is to be created must exist in the design. If an instance of the module to be instantiated already exists in that module, by default the tool uniquifies the new name by adding a “_#” suffix to the instance name. You can use the instance_uniquification_suffix switch to specify a different suffix.

- **-of_module** ***mod_spec***

  A required switch and value pair that specifies the module to be instantiated. mod_spec can be the name of a module or a module object returned by the get_modules command.

  Note

  Currently, any instantiation of parameterized modules can only be done with default values.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -below_instance *instance_object*

  An optional switch and value pair that specifies to create the instance below the specified *instance_object*. The instance specified with the ***hierInstPath*** value is created below the specified instance using the -below_instance switch. Using this switch enables you to create instances below a given instance without having to extract the name of that instance and use it as the beginning part of the ***hierInstPath*** value with a "/" to separate the two parts.

  Note

  If the *instance_object* is an empty collection or an empty string, the create_instance command uses the current design as the parent instance.

- -allow_instance_uniquification

  An optional switch that specifies to automatically uniquify the name of the new instance being instantiated if an instance of the same name already exists in the current design. This feature is very useful when inserting gates to intercept signals. You can always use the same leaf instance name and let the tool resolve conflicts automatically. This use model is illustrated in Examples 2, 3, 4 and 5 in the move_connections command description.

- -instance_uniquification_suffix *string*

  An optional switch that specifies a string to add to the new instance name when an instance of the same name already exists and you specify the -allow_instance_uniquification switch. If you don't specify this switch, the tool automatically uniquifies the object using the default string "_#" where # begins at 1 and increments by one until a unique name has been found. The string must always end with a # to denote the uniquification integer. You can also specify a different uniquification suffix using the set_insertion_options command.

- -parameter_values *parameter_list*

  An optional switch that specifies a Tcl list of parameter values when you use the command to create an instance of a parameterized module. The instance uses the supplied list of parameter values. For an example of how to use this switch, refer to "Example 4" on page 587.

  Restriction

  This feature is not supported for string parameters in SystemVerilog interfaces.

  This switch is available only in -rtl mode ("set_context dft -rtl"). For more information, refer to the set_context command.

- -silent

  An optional argument that specifies to not generate an error message under the following condition. When the error is suppressed, no instance is created and the command returns a null collection. When no errors are suppressed, the command returns a collection containing the newly created instance.

  - The hierarchical path above the leaf instance name does not exist in the design.
  - The specified mod_spec is not the name of an existing module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The collection supplied to the -of_module option does not contain a single module object.

Examples

Example 1

The following example creates an instance of module ModA called ModA_1 inside instance Block1_I1/Block2_I1. The return value is an instance object by the name of Block1_I1/Block2_I1/ModA_1.

create_instance Block1_I1/Block2_I1/ModA_1 -of_module ModA

```
{Block1_I1/Block2_I1/ModA_1}
```

Example 2

The following example creates two instances of module MUX called MUX inside instance Block1_I1. The tool uniquifies the leaf instance name when a name conflict occurs on the second invocation of the command.

create_instance Block1_I1/MUX -of_module MUX -allow_instance_uniquification

```
{Block1_I1/MUX}
```

create_instance Block1_I1/MUX -of_module MUX -allow_instance_uniquification

```
{Block1_I1/MUX_1 }
```

Example 3

This example creates two instances of module MUX called MUX inside instance Block1_I1. The tool uniquifies the leaf instance name when a name conflict occurs. The get_attribute_value_list command returns the value of the leaf_name attribute on the new instance name.

**get_attribute_value [create_instance Block1_I1/MUX -of_module MUX \
-allow_instance_uniquification] -name leaf_name**

```
{MUX}
```

**get_attribute_value [create_instance Block1_I1/MUX -of_module MUX \
-allow_instance_uniquification] -name leaf_name**

```
{MUX_1}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

This example creates an instance of a parameterized module using a list of parameter values. This example is based on the following Verilog module:

```
module ff(clk,d,q);
   parameter SIZE = 1;
   parameter OTHER = 0;
   input               clk;
   input  [(SIZE-1):0] d;
   output [(SIZE-1):0] q;
   reg    [(SIZE-1):0] q;
….
}
```

- The following command

  create_instance ff_i3 -of_module ff -parameter_values { SIZE 10 }

  creates the following Verilog code:

  ```
  ff #(.SIZE(10), .OTHER(0)) ff_i3(.clk(), .d(), .q());
  ```

- The following command

  create_instance ff_i4 -of_module ff -parameter_values [list SIZE 10 OTHER 8]

  creates the following Verilog code:

  ```
  ff #(.SIZE(10), .OTHER(8)) ff_i4(.clk(), .d(), .q());
  ```

- The following command

  create_instance ff_i5 -of_module ff -parameter_values {SIZE {"file1.txt"} OTHER 0}

  creates the following Verilog code:

  ```
  ff #(.SIZE("file1.txt"), .OTHER(0)) ff_i5(.clk(), .d(), .q());
  ```

This example creates an instance using two different parameter overrides. This example is based on the following Verilog code:

```
interface intf1;
parameter string STR_PARAM = "abc";
parameter P1 = 2;
endinterface

module M6(intf1 outA);
endmodule
```

The following command creates the instance with parameter P1 overridden with a value of 4:

create_instance u38_M6 -of_module M6 -parameter_values { outA.P1 {4}}

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following command attempts to create the instance with parameter STR_PARAM overridden with a value of “test”. The parameter is of type string, which is not supported, and the tool reports an error.

create_instance u38_M6 -of_module M6 -parameter_values { outA.STR_PARAM {"test"}}

Example 5

To see how to pass a parameter that is defined in a Verilog module to another parameter, consider this module declaration in *top.v*:

```
module top(input1);
input input1;
parameter my_param = 8;
endmodule
```

The following command instantiates the core in *top.v*:

create_instance core_inst -of_module top -parameter_values {SIZE my_param}

```
module top(input1);
input input1;
parameter my_param = 8;
  core #(.SIZE(my_param), .OTHER(2)) core_inst( .input1());
endmodule
```

Related Topics

copy_module

delete_instances

get_common_parent_instance

get_icl_pins

get_icl_ports

get_nets

rename_instance

replace_instances

uniquify_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_layout

Context: patterns -scan, patterns -scan_diagnosis

Mode: analysis

Generates a diagnosis tool-compatible Layout Database directory (LDB) from LEF/DEF input files.

Usage

create_layout ***layout_database_name*** [-lef *lef_filename* … | -leflist *leflist*]
[-def *def_filename* ... | -deflist *deflist*]
{[-extra_layout_hierarchy *extra_levels_of_hierarchy_in_layout*] | [-extra_design_hierarchy *extra_levels_of_hierarchy_in_design*]}
[-threshold *percentage_match*] [-topology_threshold percentage_match]
[-temp_dir *directory_name*] [-compression {on | off}]
[-keep_uncompressed_database] [-core *core_name*] [-verify_only]
[-threads *num_threads* [-min_threads *min_threads*]]
[-compact {on | off}] [-bridge_proximity_factor *bridge_proximity_factor*]
[-topology_error_count *integer*]
[-gray_box_detection]
[-hierdb *hierdb_name*] [-replace] [-continue] [-upgrade]

For Pre-Existing LDBs

create_layout ***layout_database_name*** [-adjust_vias]

Description

Refer to "Layout-Aware Diagnosis" in the *Tessent Diagnosis User's Manual* for complete information.

Before generating the LDB, the tool performs layout verification of the LEF/DEF files and reports any mismatches to the transcript—refer to "Layout Verification and Layout Database Creation Process in the *Tessent Diagnosis User's Manual*.

After the layout-verification stage, you can interrupt the LDB creation by issuing a Ctrl-C in the invocation shell. You can subsequently use the report_layout_rules command to obtain details on the layout rule violations and use this information to correct errors in LEF/DEF files, providing the correct LEF/DEF files and add any missing LEF/DEF files to the list provided to the create_layout command. Following this, you then use the create_layout command to determine if there are any additional issues. Use this method in iterative fashion to correct any problems with the LEF/DEF before creating the LDB.

After generating the LDB, the tool performs layout pre-extraction. The tool extracts bridge and open defect information and stores this information in the LDB for use with RCD constant creation (with the create_feature_statistics command) and dynamic partitioning-based diagnosis. In addition, layout pre-extraction improves layout-aware diagnosis performance.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can improve runtime performance of the layout pre-extraction step by specifying the -threads and -min_threads options. These options enables you to check out additional licenses so that Tessent Diagnosis can perform some layout pre-extraction tasks in parallel.

Tip

For maximum efficiency, run the create_layout command on a disk that is local to the machine hosting the process. Also, because this command reads the entire physical layout information into memory by default, you should run it on a machine with sufficient physical memory.

Arguments

- ***layout_database_name***

  A required string that specifies the name of the LDB that diagnosis creates from the LEF/DEF input.

- -lef *lef_filename* …

  An optional switch and string that specifies space-separated name(s) of the LEF file(s).

- -leflist *leflist*

  An optional switch and string pair that specifies a Tcl list of the LEF file(s).

- -def *def_filename* ...

  An optional switch and string that specifies space-separated name(s) of the DEF file(s).

- -deflist *deflist*

  An optional switch and string pair that specifies a Tcl list of the DEF file(s).

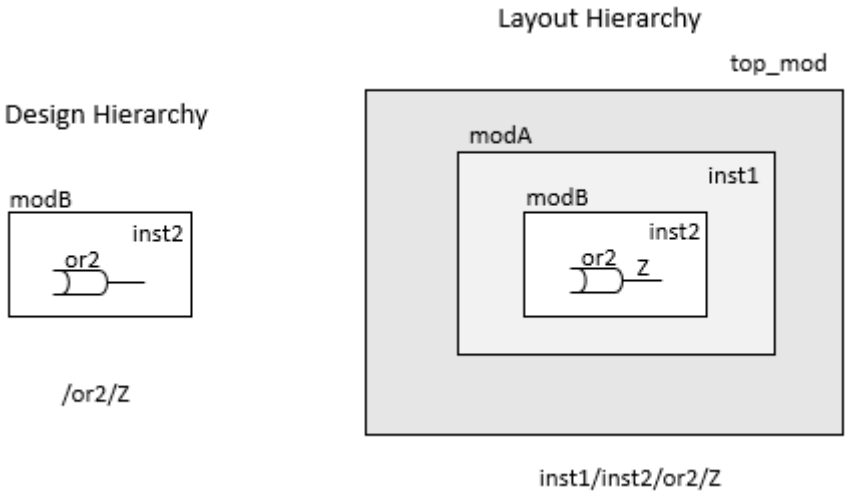
- -extra_layout_hierarchy *extra_levels_of_hierarchy_in_layout*

  An optional switch and string pair that removes extra levels of layout hierarchy in the LEF/DEF layout such that the design hierarchy and layout hierarchy match. You can only specify this switch once, otherwise the tool issues an error and exits.

  Use this switch if a core instance is a subset of a larger layout hierarchy. You can specify multiple levels of hierarchy in one string with no spaces. The following figure illustrates this usage.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 3-33. create_layout With Extra Layout Hierarchy



The following command removes the hierarchy from the LEF/DEF net, pin names, and so forth before matching with the design name:

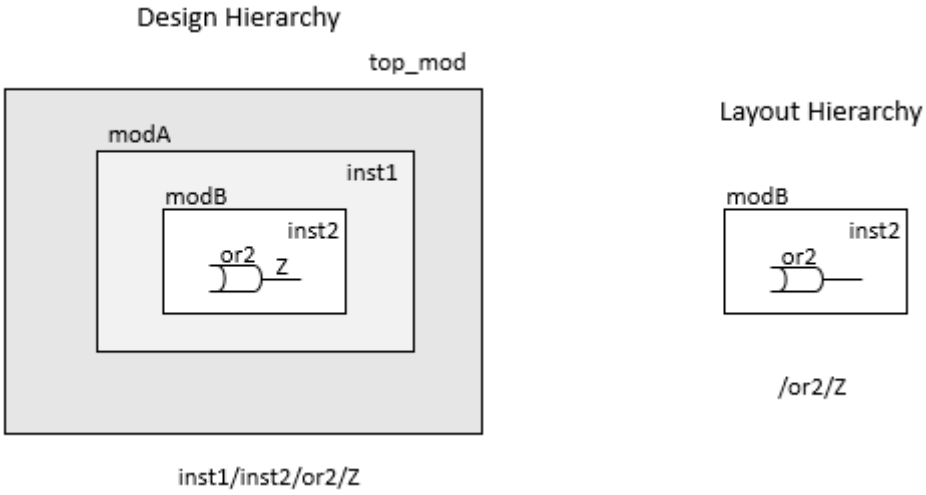
```
create_layout layout_database_name -lef lef_file -def def_file \
    -extra_layout_hierarchy inst1/inst2
```

- -extra_design_hierarchy *extra_levels_of_hierarchy_in_design*

  An optional switch and string pair removes extra levels of design hierarchy such that it matches the LEF/DEF layout hierarchy. You can only specify this switch once, otherwise the tool issues an error and exits.

  You use this switch if a core instance is a subset of a larger design hierarchy. You can specify multiple levels of hierarchy in one string with no spaces. The following figure illustrates this usage.

Figure 3-34. create_layout With Extra Design Hierarchy



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following command removes hierarchy from a design pin, net name, and so forth before matching with layout:

```
create_layout layout_database_name -lef lef_file -def def_file \
   -extra_design_hierarchy inst1/inst2
```

Note

The -EXTRA_DESign_hierarchy and -EXTRA_LAYout_hierarchy switches are mutually exclusive, and the tool issues an error if you specify both.

- -threshold *percentage_match*

  An optional switch and integer pair that specifies the minimum percentage match threshold required to successfully complete a layout verification of the LEF/DEF files, and subsequent creation of the LDB. The default threshold value is 85 percent. The following command sets the required layout match threshold to 92 percent:

  ```
  create_layout design.layout -lef lef_input.lef -def def_input.def \
     -threshold 92
  ```

- -topology_threshold *percentage_match*

  An optional switch and integer pair that specifies the minimum percentage net tracing threshold required to successfully complete topology extraction during the LDB creation process. The default threshold value is 90 percent, meaning topology extraction is considered successful if the tool can trace 90 percent of the nets. If the success rate falls below the default or user-specified *percentage_match* rate, the tool issues an error and stops LDB creation.

- -temp_dir *directory_name*

  In some cases, Tessent Diagnosis must update the LDB (for example, if you use the -verify switch). Due to the fact that the compressed LDB is a read-only database, Tessent Diagnosis automatically uncompresses, updates, and re-compresses the LDB. By default, the temporary uncompressed LADB is stored into the same location where the compressed LDB is located; if you have limited disk space in this location, then use the -TEmp_dir switch and string pair to specify an alternative location.

  For chip-mapped core LDB creation (create_layout -core), this temp_dir location will also be used to temporarily create and store a layout hierarchy database if the hierarchy database was not specified using the -hierdb switch.

  Refer to "Layout Database Compression and Decompression" in the *Tessent Diagnosis User's Manual* for complete information.

- -compression {on | off}

  An optional switch and literal that controls whether the tool compresses the LDB during database creation. The default is OFF. Specify ON to create a compressed LDB.

  Refer to "Layout Database Compression and Decompression" in the *Tessent Diagnosis User's Manual* for complete information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -keep_uncompressed_database

  An optional switch that specifies that the tool retain a copy of the uncompressed LDB as well as a compressed version of the LDB.

- -core *core_name*

  An optional switch that specifies to create an instance-aware core-level LDB for the specified core design. This option only pertains to and is required for the hierarchical layout-aware diagnosis flow.

  Refer to "Diagnosis for Hierarchical Design" in the *Tessent Diagnosis User's Manual* for complete information.

- -verify_only

  An optional switch that enables LEF/DEF design verification and stops LDB creation after verification steps have completed.

- -threads *num_threads*

  An optional switch and integer pair that specifies the number of CPUs to use so that the tool can run some create_layout tasks in parallel.

  num_threads is an integer such that:

  num_threads > 1: Use the specified number of threads.

  num_threads = 0: Use the number of CPUs of the host for the number of threads.

  num_threads < 1: Use the number of CPUs less the specified *num_threads* value.

  If the requested number of threads is greater than the host's CPUs, the tool automatically adjusts the thread count down to the number of CPUs.

  The tool requires one yieldascandiag license per thread. For example, if you specify "-threads 8", the tool checks out 7 yieldascandiag licenses in addition to the 1 yieldascandiag license that was checked out when you entered scan_diagnosis context. The tool releases the additional licenses when the create_layout command completes processing.

  Note

  If you run this command in the "patterns -scan" context, the tool requires the mtautogratpg_c license instead of the yieldascandiag license.

  If less than the requested licenses are available, license queuing occurs based on the setting from the set_license_queue_timeout command. If the tool cannot acquire the number of licenses specified by the user, the number of threads is adjusted down to the number of acquired licenses.

- -min_threads *min_threads*

  An optional switch and integer pair that is used with the -threads option and specifies the minimum number of threads that must be available. The tool aborts the create_layout command if the number of acquired licenses does not meet the -min_threads requirement.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

num_threads is an integer such that:

num_threads > 1: Use the specified number of threads.

num_threads = 0: Use the number of CPUs of the host for the number of threads.

num_threads < 1: Use the number of CPUs less the specified *num_threads* value.

- -compact {on | off}

  An optional switch that specifies whether to produce a compact LDB, which is an optimized database that is faster to create and has a smaller size on disk. Compact LDBs have the following limitations:

  - In the Tessent YieldInsight layout viewer, you do not see background polygons. (Landmark and suspect polygon display remains the same.)
  - The tool does not perform DFM rule verification during the import_dfm process. (The rest of the DFM flow and reporting remains the same.)

  You can convert existing compact LDBs by specifying "create_layout *existing_LDB* -compact off."

- -bridge_proximity_factor *bridge_proximity_factor*

  An optional switch that controls the maximum distance permitted between nets for them to be considered possible bridge defects. Specify a scale factor from 1 to 100. The default is 10. The tool calculates the maximum distance by multiplying the specified scale factor by the minimum net spacing between the layer.

  This option is useful for eliminating bridge suspects that have minimal significance for failure analysis thus improving resolution metrics.

- -topology_error_count *integer*

  An optional switch and integer that specifies the number of net errors per issue class to display in the net topology extraction transcript. The default is 5.

- -gray_box_detection

  An optional switch that specifies to populate the LDB with graybox layout information for the purposes of performing graybox-aware top-level pattern diagnosis. For more information, refer to "Top-Level Layout-Aware Diagnosis" in the *Tessent Diagnosis User's Manual*.

- -hierdb *hierdb_name*

  An optional switch that specifies the name of a layout hierarchy database as created by the analyze_layout_hierarchy command. When you specify this option, the tool implements the optimized chip-mapped core-level LDB creation process as described in "Generating Chip-Mapped Core-Level LDBs" in the *Tessent Diagnosis User's Manual*. This process optimizes core-level LDB creation by only processing the DEFs required for a specified core.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -replace

  An optional switch that overwrites existing layout pre-extraction information stored in the LDB with newly generated layout pre-extraction files.

- -continue

  An optional switch that specifies to continue generating the LDB, including the layout open and bridge defect information. Use this switch when:

  - Processing halts for any reason. The tool automatically re-starts processing where it left off.
  - You have a pre-2014.4 LDB and you would like to add layout open and bridge defect information to it.

- -adjust_vias

  An optional switch that merges vias with the same physical geometries and layer names under one name, and adjusts the LDB accordingly. This results in consistent via naming, which can improve the accuracy of RCD results.The tool does not perform verification, pre-extraction, or LDB re-creation. Refer to Example 6.

  To receive the benefit to RCD accuracy from via merging, you must re-create existing diagnosis reports by re-running diagnosis with the updated LDB. The feature statistics (create_feature_statistics) do not need to be recalculated.

- -upgrade

  An optional switch that analyzes and performs any required upgrades to the specified LDB. If upgrades are not required, no changes will be made to the LDB.

Examples

Example 1

In the following example, the diagnosis tool creates a LDB (*design.layout*) directory that includes pre-extracted bridge and open information from multiple LEF files and DEF files:

```
create_layout design.layout -lef lef_input01.lef lef_input02.lef -def def_01.def def_02.def
```

Example 2

In the following example, Tessent Diagnosis creates a LDB (*design.layout*) directory that includes pre-extracted bridge and open information from a LEF file (*lef_input.lef*) and four DEF files (*a.def, b.def, c.def, d.def*). It skips re-generating valid pre-extraction information.

```
create_layout design.layout -lef lef_input.lef -def a.def b.def c.def d.def -continue
```

Example 3

In the following example, the tool pre-extracts layout bridge and open defect information and stores it in an existing LDB. It recreates all the layout pre-extraction information.

```
create_layout design.layout -replace
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

In the following example, the set_license_queue_timeout is set for a one minute timeout and you have specified to check out four licenses (for four threads) with the create_layout command.

set_license_queue_timeout 1

create_layout results/layoutDB -def ya_demo6.def.gz -lef ya_demo5.lef.gz -threads 4

```
//  Note: multithreading license queue type: timeout wait
//  Note: Requested threads (4) requires check out of 3 additional
//  licenses.
//  Note: Acquired 2 of 3 requested licenses.
//  Note: Queueing 1 minute for 1 license ( CTRL-C to run with available
//  licenses ).
//  Waited 1 of 1 minute for 'yieldascandiag'.
//  Warning: Could not acquire 3 licenses.  Proceeding with reduced thread
//  count of 2.
```

Example 5

In the following example, the set_license_queue_timeout is set for no waiting and you have specified that you require four licenses. However, there are only three available licenses.

set_license_queue_timeout no_queue

**create_layout results/layoutDB5 -def ya_demo6.def.gz -lef ya_demo5.lef.gz -threads \
4 -min_threads 4**

```
//  Note: multithreading license queue type: no_queue
//  Note: Reqested threads (4) requires check out of 3 additional
//  licenses.
//  Note: Acquired 2 of 3 requested licenses.
//  Error: Insufficient licenses acquired (2) and -min_threads (4)
//  specified.
//  Note: Releasing 2 licenses.
//  Error: Layout feature preparation failed.
```

Example 6

The following example shows how Tessent Diagnosis merges vias with the same physical geometries and layer names when you specify create_layout -adjust_vias.

The tool begins the analysis by grouping LEF and DEF vias that have the same physical geometries and layer names. Within a group of equivalent vias, the tool arrives at the merged via name as follows:

1. Consider LEF via names first. If there is only one LEF via in the group, use its name. For example:

```
abcasc        // LEF via ---> use this name
xyyx          // DEF via
abc           // DEF via
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

2. If there are multiple LEF vias, use the LEF via with the shortest via name. For example:

```
abcasc        // LEF via
streq         // LEF via ---> use this name
xyyx          // DEF via
abc           // DEF via
```

3. If there are multiple LEF vias and two (or more) have via names of the same shortest length, use the name with the lowest alphanumeric value. For example:

```
abcasc        // LEF via
xyyx          // LEF via
abcd          // LEF via ---> use this name
abc           // DEF via
```

4. If there are no LEF vias, use the DEF via with the shortest via name.
5. If there are two (or more) DEF vias of the same shortest length, use the name with the lowest alphanumeric value.

Related Topics

close_layout

compress_layout

open_layout

report_layout_rules

uncompress_layout

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_memory_repair_flow

Context: patterns -silicon_insight

Mode: setup, analysis

Incorporates a Repair Flow Custom Test into the patterns specification.

Usage

create_memory_repair_flow [-name *test_name*] [-repair_type hard | soft]
[-vddqh *fuse_programming_high_voltage*] [-vddql *fuse_programming_low_voltage*]
[-report_memory] [-replace]

Description

Before using this command, you must have a freshly generated pattern specification for manufacturing usage from the "create_patterns_specification_manufacturing" command with the TESSENT_SI_ALLOW_DESTRUCTIVE_BISR environment variable set to 1.

When this command successfully incorporates the repair flow custom test into the patterns specification, it displays the following message:

```
PatternsSpecification /PatternsSpecification(<design_name>,<design_id>,manufacturing)
was modified for the CDP generation in order to support repair. If you want this
modification to be preserved in your PatternsSpecification for future sessions, please
use the write_config_data command to save it.
```

Arguments

- -name *test_name*

  An optional switch and string pair that specifies the name of the repair flow custom test (corresponding to the Patterns wrapper name) within the PatternsSpecification. The default value is CUSTOM_PROCFILE_my_repair_tcl_PROCNAME_my_repair if you do not specify this argument.

- -repair_type hard | soft

  An optional switch and literal pair that specifies the type of repair flow. The default value is hard.

- -vddqh *fuse_programming_high_voltage*

  An optional switch and floating number pair that specifies the fuse programming high voltage, which is only applicable to hard repair. The default value is 2.5.

- -vddql *fuse_programming_low_voltage*

  An optional switch and floating number pair that specifies the fuse programming low voltage, which is only applicable to hard repair. The default value is 1.5.

- -report_memory

  An optional Boolean switch that issues a report of memories that the tool repairs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -replace

  An optional Boolean switch that suppresses the error that the tool normally issues if a test with the same name already exists. When you use the -replace argument, the tool silently deletes the existing test before creating the new one.

Examples

In the following example, the first call of the create_memory_repair_flow command creates a hard repair custom test, and the second call creates a soft repair flow custom test, after specifying the setup specification:

```
set_context patterns -silicon_insight
read_config_data si.setup
set ::env(TESSENT_SI_ALLOW_DESTRUCTIVE_BISR) 1
create_patterns_spec manufacturing
create_memory_repair_flow
create_memory_repair_flow -name \
        CUSTOM_soft_PROCFILE_my_repair_tcl_PROCNAME_my_repair -repair_type soft
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_module

Context: all contexts

Mode: insertion

Creates a new design module with no ports.

Usage

create_module ***module_name*** [-language {verilog | vhdl | system_verilog}] [-in_library *library_name*]
[-vhdl_architecture *vhdl_architecture*] [-silent]

Description

The create_module command returns a collection of one module object where the module is a newly created module.

After creating a module with this command, you can set the new module as the top level of the design for all subsequent commands by using the set_current_design command and providing the name of the new module.

You can add ports to this new design module using the create_port command, add instances using the create_instance command, and then connect the pins using the create_connections command. Optionally, you can save the completed module to disk using the write_design command. Refer to "Design Editing Examples" section of the *Tessent Shell User's Manual*.

After a new module is created, the current design can be reset to a different level with the set_current_design command and the module can be inserted with a create_instance command.

Arguments

- ***module_name***

  A required string that specifies the name of the module to create.

- -language verilog | vhdl | system_verilog

  An optional switch and literal pair that specifies which language (Verilog, VHDL, or SystemVerilog) to use when creating the module. By default, the language is Verilog when you do not specify this option. When you choose the system_verilog option, the tool writes SystemVerilog 2009 format. The -language option is only permitted in the rtl context.

- -in_library *library_name*

  An optional switch and string pair that specifies in which library to create the module. The specified library_name must be a library name previously defined with the set_logical_design_libraries command. When the -in_library option is not specified, the module is created inside the default library, which is also specified using the set_logical_design_libraries command. The -in_library option is only permitted in the rtl context.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -vhdl_architecture *vhdl_architecture*

  An optional switch and string pair that specifies the name of the architecture to use for a created VHDL module. When -vhdl_architecture is not specified and -language is specified as vhdl, the created architecture is called struct. You cannot specify -vhdl_architecture unless -language is specified as vhdl. The -vhdl_architecture option is only permitted in the rtl context.

- -silent

  An optional switch that suppresses the error that is normally generated if module_name already exists. A null collection is returned when an error is suppressed.

Examples

Example 1

The following example creates a module called mytop and makes it the current module so that other modules can be instantiated in it.

```
set_context dft -no_rtl
set_system_mode insertion
create_module mytop
```

```
{mytop}
```

```
set_current_design mytop
```

Example 2

The following example creates a module called myblock and appends the newly created module object to a collection referenced by the Tcl variable called created_modules.

```
append_to_collection created_modules [create_module myblock]
```

```
{myblock}
```

Example 3

The following example creates a VHDL module inside library lib1 with the default vhdl architecture struct. Note how the get_modules command is used to report its attribute as report_attribute only looks for modules in the default library

```
create_module M1 -in_library lib1 -language vhdl
```

```
{M1}
```

```
report_attributes [get_modules M1 -in_library lib1]
```

```
Attribute Definition Report
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Attributes defined for object 'M1':
Name                Value   Inheritance
-----------------   ------  -----------
file_path_name              -
is_created          true    -
is_modified         false   -
is_unsaved          true    -
is_valid            true    -
language            vhdl    -
library_name        lib1    -
line_number         0       -
master_name         M1      -
name                M1      -
object_type         module  -
type                design  -
vhdl_architecture   struc   -
```

Related Topics

copy_module

create_connections

create_instance

create_port

get_modules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_net

Context: all contexts

Mode: insertion

Creates a net inside an instance of a design module in the current design.

Usage

create_net ***hier_net_name***
[-below_instance *instance_object*]
[-ascending]
[-silent]

Description

This command returns a collection of created net objects. This command returns one object for each bit of a bus with the left bit first and the right bit last.

Arguments

- ***hier_net_name***

  A required string that specifies the path to the instance in which the net is created, and the leaf name of the net to be created in the module. If the net name is a bus, this command creates one net for each bus bit, beginning with the left-most bit of the bus and ending with the right-most. The specified hierarchical path in which the net is to be created must exist in the design.

- -below_instance *instance_object*

  An optional switch and value pair that specifies to create the net below the specified *instance_object*. The tool searches the specified *name_patterns* relative to each instance found in the *instance_object* list. If the *instance_object* is a null collection or a null string, the switch is ignored and instances are searched relative to the current design.

- -ascending

  An optional switch that specifies the range direction type for a VHDL bus port. By default, the range direction type is inferred as “downto” when the left index is larger than the right index, and as “to” when the left index is smaller than the right index. You use this switch to specify that the VHDL port notation is “to” when the left index is equal to the right index; otherwise, the tool uses the “downto” range direction type. An error is generated if you specify this option and the left index is larger than the right index, or if the object name is not a bus. The -ascending option is only permitted in the rtl context.

- -silent

  An optional argument that specifies the suppression of an error given one of the situations below. The command returns a null collection when an error is suppressed.

  - The hierarchical path of the net name specified does not exist in the design

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The net already exist in the design

Examples

The following example creates a two-bit bus net called A inside instance u1.

create_net u1/A[4:5]

```
{u1/A[4] u1/A[5]}
```

Related Topics

delete_pins

delete_nets

get_icl_pins

get_icl_ports

get_nets

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_open_socket_failure

Context: patterns -failure_mapping

Mode: setup

Creates emulated open-socket failure files to use for validating reverse mapping top-level failures to the core in Tessent Diagnosis.

Usage

create_open_socket_failure ***failure_filename*** [-pattern_count *count*] [-replace]

Description

An open-socket failure log is a file that would be created if the test was run without a chip in the socket. When you specify this command, the tool loops over the patterns and cores in the design to find the scan chains and top-level pins. A fail bit is output on each top-level pin for each chip-level measure cycle that is not filtered by -pattern_count.

For usage details, see "Validating Reverse Mapping Prior to Core-Level Layout-Aware Diagnosis" in the *Tessent Diagnosis User's Manual*.

Arguments

- failure_filename

  A required string that specifies the name of the output open-socket failure file.

  Note

  The command creates cycle-based failure files. For examples, see "Cycle-Based Failure File Examples" in the *Tessent Diagnosis User's Manual*.

- -pattern_count *count*

  An optional switch and integer that specifies the number of patterns to output from the beginning and end of each test suite. If the count is 0, then the entire pattern range is output.

  The default is 1 for hierarchical test patterns, which leads to failures being generated for the first and last pattern of each test suite.

  The default is -1 for SSN, which leads to failures being generated for all pattern cycles in the test suite.

  Note

  For SSN, we recommend that you do not use this optional switch. When you use this switch with SSN, it is possible to specify an integer count that is too low, which results in the inability to map failures correctly.

- -replace

  An optional switch that overwrites existing open-socket failure information with newly generated data.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_patterns

Context: dft -edt, patterns -scan

Mode: analysis

Performs automatic high-speed test generation and pattern compression.

Usage

create_patterns [-OVerride_user_settings | -NO_AUto]
[-COMpaction_effort {Low | High | Maximum | *integer*}] [-NO_Analyze]
[-COVerage_effort {Low | Medium | High}] [-NO_REDundancy_analysis]
[-NO_TERMINATE_ineffective_atpg] [-PATterns_per_pass *integer*]
[-NO_REPort_statistics]

Description

This command is the recommended way to produce high quality, efficient, test pattern sets in the shortest possible time.

This is a multiprocessing command for ATPG. For more information about multiprocessing, refer to "Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual.*

The create_patterns command uses the built-in expert system, ATPG Expert, to create patterns. ATPG Expert analyzes the design and DRC and may modify tool settings to optimize test patterns for coverage, pattern count, and run time. By default, ATPG Expert only optimizes settings that have not been explicitly specified; it does not change user-specified settings. If you want ATPG Expert to also optimize user-specified settings, use the -override_user_settings switch. Additionally, a statistics report is automatically displayed. The create_patterns command displays the following statistics:

- Number of patterns simulated
- Test coverage
- Number of faults detected
- Number of faults remaining in the fault list (undetected)
- Number of effective patterns
- Number of test patterns
- Number of redundant, untestable, and aborted faults

  Occasionally, the display includes a line with three or four numbers separated by slashes (/). These numbers are, from left to right, the cumulative number of redundant (RE), ATPG_untestable (AU), and aborted faults. Aborted faults are separated into ATPG aborts (AAB) and EDT aborts (EAB) depending on whether compression is on. The sum of AAB and EAB aborts equal the total number of aborts.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example transcript without compression (these numbers are shown in bold font):

```
//#patterns test     #faults #faults   #eff.   #test    process    RE/AU/AAB
//simulated coverage in list detected patterns patterns CPU time
//   ---      ---     ---      ---      ---      ---    5.79 sec   0/40868/57

//    64     74.10%  135463  413980      63      63    14.73 sec
```

Example transcript with compression (these numbers are shown in bold font):

```
//#patterns   test   #faults #faults   #eff.   #test    process    RE/AU/AAB/EAB
//simulated coverage in list detected patterns patterns CPU time
//   ---     ------    ---     ---      ---      ---    5.79 sec   0/40868/27/30
//    64     74.10%  135463  413980      63      63    14.73 sec
```

Note

At the start of create patterns, when the hardware has EDT and the EDT low-power controller is enabled, but set_power_control has not been used, the tool issues the following note. The note says “highest” because there may be multiple active EDT instances, and they may have different hardware thresholds.

```
// Note: Creating low-power patterns with <N>%
// shift switching threshold based on highest threshold of all active EDT
// low-power controllers or 15%, whichever is higher.
// To use a different low power shift threshold or disable low
// power shift, use the ‘set_power_control shift on
// -switching_threshold_percentage <int>’ command.
```

At the end of the create_patterns run, the tool automatically performs redundant fault identification, which is equivalent to running the identify_redundant_faults command. If, however, you set any ATPG limit (for example, test coverage limit, run time limit, or pattern limit), then the tool does not perform redundant fault identification.

You should use the create_patterns command, review the transcript for any suggested command or setting changes, and implement the ones that help you achieve your test goals. Be aware that some of the suggestions have trade-offs (for example, greater run time in exchange for fewer test patterns); the merits of the trade-offs depend on your particular test requirements.

Note

When you perform compression analysis, the final order of scan cells in each scan chain in the design is not known. The order of the scan cells used as deterministic bits in the test cube impacts ATPG results. If many of the deterministic bits are clustered in consecutive cycles, more EDT_ABORT faults may appear because the clustered deterministic bits may not be compressed. When this occurs, ATPG test coverage may drop.

During compression analysis, the tool assumes the scan chains are organized in a balanced way based on an existing scan cell ordering. However, in the real design, scan cells may be in an arbitrary position depending on which scan stitching tool was used. If the clustering of deterministic bits occurs during compression analysis but not in the real design or if the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

clustering occurs in the real design but not in compression analysis, a discrepancy in the ATPG results may occur.

For static fault models, the tool automatically calls “add_faults -all” when the create_patterns command is run and the fault list is empty.

For transition and UDFM with defect_delay fault models, the tool automatically calls “add_faults -clock_domains all” when the create_patterns command is run and the fault list is empty. The tool does this to ensure that at-speed test is only performed within groups of synchronous clocks.

Note that you can use the add_faults command to define the scope of the create_patterns command. For more information, refer to “Example 3” on page 611.

The CPU time the tool reports is the time local to the command. If the command has multiple phases, then the reported CPU time is with respect to the beginning of that phase.

Arguments

- -OVerride_user_settings

  An optional switch that enables ATPG Expert to override user-specified settings if necessary to create an optimal test pattern set.

- -NO_AUto

  An optional switch that disables the use of ATPG Expert to create patterns. During test generation, the default settings are used for those settings that have not been explicitly specified.

- -COMpaction_effort {Low | High | Maximum | *integer*}

  An optional switch that specifies how much effort to spend trying to reduce the pattern count:

  Low

  Specifies a faster run time but a higher pattern count than the High option.

  High

  Specifies a longer run time but a smaller pattern count than the low option. Use this option when generating production patterns to significantly reduce test time and test data volume. This option usually gives the best results and is the default, except when using the set_multiprocessing_options command. In that case, the command “set_multiprocessing_options -optimize_secondaries runtime” automatically sets “-compaction_effort low” unless changed when creating patterns.

  Maximum

  Enables maximum tool effort to reduce pattern count. This option significantly increases run time and should be used only when absolutely necessary. Integer value

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

10 or 100 should be used before using this option due to the significantly long run time.

integer

A positive integer value that specifies compaction effort as follows:

0 = Low

1 = High (usually give the best results)

n = The tool works *n* times harder than High

As a general guide, when *n*=10, the run time increase is small, and the pattern count can be slightly better. When *n*=100, pattern reduction can be similar to Maximum, while the run time can be lower than Maximum. Differences in design complexity and size can affect which integer value provides the trade-off between pattern counts and run time you want.

Note that you can significantly reduce the run time caused by higher integer values by using multiprocessing. For more information about multiprocessing, refer to "Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual.*

- -NO_Analyze

  An optional switch that skips the analysis of test generation problems. Normally, before test generation, create_patterns checks for bus contention and whether ATPG restrictions and constraints can be satisfied. If these checks reveal potential test generation problems, an error message displays and the command terminates. If you want the analysis to be ignored and to proceed with test generation, reissue the command with this switch.

  Note

  Because this switch prevents normal analysis, all faults can be targeted, and ATPG effort expanded, yet without generating patterns. The extra effort may also give the appearance that the tool is stuck. Use this switch only in special cases when the risks are acceptable.

- -COVerage_effort {Low | Medium | High}

  An optional switch and keyword that specifies the level of effort the tool uses to reduce the number of aborted faults.

  Low

  Use for a faster run time but with the potential for a slight loss of coverage.

  Medium

  Use for most cases. This is the default.

  High

  Use for runs with many aborted faults but with the potential for a significant increase in run time. For that reason, Siemens EDA recommends using the High setting as part of a top-up pattern generation flow.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -NO_REDundancy_analysis

  An optional switch that omits the redundancy identification analysis that follows pattern generation. When you use this switch, the patterns generate faster, but the test coverage calculations are less accurate.

- -NO_TERMINATE_ineffective_atpg

  An optional switch that disables a feature that terminates the ATPG session after the patterns are no longer efficient at detecting faults. Note that using this switch results in a longer run time and may generate many additional patterns to detect very few faults.

- -PATterns_per_pass *integer*

  An optional switch and positive integer pair that specifies how many patterns to create and simulate in each pass. The integer value must be between 1 and 64. By default, the tool creates and simulates 64 patterns in each pass. When the value is 1, the tool generates and simulates every pattern, and so on. The switch affects only the current create_patterns session.

  The potential benefit of using a smaller integer value rather than the default 64 is better pattern count because ATPG is less likely to add non-essential faults into a pattern. However, when the value is small, the overall run time is likely to increase due to excessive fault simulation. Furthermore, a small value can reduce the run-time efficiency of multithreading and distributed ATPG.

  A smaller value is appropriate if the expected pattern count is small. A typical scenario for using a small integer value is IDDQ pattern generation, where you should use "create_patterns -patterns_per_pass 1" with "set_atpg_limits -pattern_count *n*" for creating *n* IDDQ patterns. You can use the -compaction_effort option with -patterns_per_pass to maximize the effectiveness of each pattern, but at the expense of additional run time.

- -NO_REPort_statistics

  An optional switch that suppresses statistics reporting after the tool creates patterns. This is helpful for avoiding duplicate reports when you want to run the report_statistics command with non-default options immediately after running create_patterns.

Examples

Example 1

The following example creates, through ATPG Expert, an internally-stored set of optimally compact patterns for the entire netlist that can be saved using the write_patterns command. ATPG Expert automatically decides the best setup (for example, sequential depth) for the design.

```
set_system_mode analysis
create_patterns
```

Example 2

The following example also creates an internally-stored set of compact patterns; however, ATPG Expert has been disabled. In this case, you must specify which sequential depth the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

pattern is created. The -sequential 2 switch means there are up to two clock pulses between scan loads. The patterns can be saved using the write_patterns command.

```
set_system_mode analysis
set_pattern_type -sequential 2
create_patterns -no_auto
```

Example 3

The following example creates patterns only for the specific block specified by the add_faults command:

```
set_system_mode analysis
add_faults /blockA/blockB
create_patterns
```

Example 4

The following example creates, through ATPG Expert, an internally-stored set of optimally compact at-speed fault patterns for the entire netlist that can be saved using the write_patterns command. Notice that for at-speed fault models, if you are not using named capture procedures, and there are no user defined settings or you are not calling create_patterns with the -override switch, the tool automatically turns on hold_pi and mask all primary outputs. It also automatically sets clock restriction to "domain_clock" with any_interaction and compatible_clocks_between_loads ON.

```
set_system_mode analysis
set_fault_type transistion
rep_faults

//  Warning: No faults reported.

create_patterns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// | ---------------------------------------------------------------------------|
// |                    Analyzing the design                                    |
// |                                                                            |
// |  Current clock restriction setting:  Domain_clock (edge interaction)       |
// |  Calling: set_clock_restriction domain_clock -any_interaction              |
// |                -compatible_clocks_between_loads on                         |
// |                                                                            |
// |  Current split capture setting:          Off (optimal)                     |
// |                                                                            |
// |  Current clock off simulation setting:   On(optimal)                       |
// |                                                                            |
// |        Current holdpi setting:     Off                                     |
// |                        Calling:     set_transition_holdpi on                |
// |                                                                            |
// |  Current output masks setting:     Off                                     |
// |                        Calling:     set_output_masks on                     |
// |                                                                            |
// |   Current abort limit setting:     300 (Combinational) 100  (Sequential)   |
// | ---------------------------------------------------------------------------|
// |                                                                            |
// |               Current sequential depth:     2 (optimal)                    |
// |                                                                            |
// | --------------------------------------------------------------------------- |


// ---------------------------------------------------------------------
//  Simulation performed for #gates = 654  #faults = 2497
//  system mode = ATPG    pattern source = internal patterns
//  ---------------------------------------------------------------------
//  #patterns  test      #faults  #faults    # eff.    # test     process      RE/AU/AAB
//  simulated  coverage  in list  detected   patterns  patterns   CPU time
//  deterministic ATPG invoked with comb/seq abort limit = 300/100
//   ---        ------      ---       ---      ---       ---          0.30 sec    21/15/5
//   64         79.86%      533      1949       59        59          0.31 sec
//   ---        ------      ---       ---      ---       ---          0.54 sec    21/91/19
//   128        89.31%      200       257       54       113          0.54 sec
//   ---        ------      ---       ---      ---       ---          0.64 sec    21/173/29
//   190        92.80%       23        95       49       162          0.64 sec
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
              Statistics Report
              Transition Faults
---------------------------------------------
Fault Classes                      #faults
                                   (total)
----------------------------  ---------------
  FU (full)                          2758
  --------------------------  ---------------
  UO (unobserved)                   23 ( 0.83%)
  DS (det_simulation)             2301 (83.43%)
  DI (det_implication)             224 ( 8.12%)
  UU (unused)                        4 ( 0.15%)
  TI (tied)                         12 ( 0.44%)
  RE (redundant)                    21 ( 0.76%)
  AU (atpg_untestable)             173 ( 6.27%)
---------------------------------------------
Fault Sub-classes
  --------------------------
  AU (atpg_untestable)
    SEQ  (sequential_depth)         44 ( 1.60%)
    Unclassified                   129 ( 4.68%)
  UC+UO
    AAB  (atpg_abort)               23 ( 0.83%)
---------------------------------------------
Coverage
  --------------------------
  test_coverage                        92.80%
  fault_coverage                       91.55%
  atpg_effectiveness                   99.17%
---------------------------------------------
#test_patterns                            162
  #basic_patterns                           2
  #clock_sequential_patterns              160
#simulated_patterns                       190
CPU_time (secs)                           1.4
---------------------------------------------
```

Related Topics

add_faults

identify_redundant_faults

report_statistics

reset_state

set_abort_limit

set_atpg_limits

set_fault_type

set_pattern_type

simulate_patterns

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_patterns_specification

Context: All contexts

Modes: All modes

Prerequisite: The top level ICL module must exist and be elaborated.

Generates a pattern specification for the specified usage. The usage is either signoff or manufacturing. The tool supports IJTAG and MemoryBIST patterns.

Usage

create_patterns_specification [*usage*]
[-pattern_id *pattern_id*]
[-replace]
[-ignore *ignored_instrument_list*]
[-bsdl_files *bsdl_file*]
[-testbench_files *file_path_list*]
[-infer_loopbacks_during_icl_verify { on | off }]
[-infer_loopbacks_during_memory_bisr_verify { on | off }]
[-infer_loopbacks_during_boundary_scan_verify { on | off }]
[-bscan_sim_views { full | ijtag_graybox }]
[-separate_tap_and_bscan_patterns { on | off }]

Description

This command can be called in any context and any system mode as long as the ICL module associated with the top-level design is available and elaborated, or if a BSDL file is specified for a “BSDL-only” design flow. You can provide a top-level ICL module description by using first the read_icl command followed by the set_current_design command, or by performing ICL extraction using the extract_icl command. The context is automatically changed to patterns if called from a different context.

Note

When you add pseudo ports in the dft context (commands labeled “dft” or “all contexts”), the tool does not consider them when you use the create_patterns_specification command, because they do not exist in the patterns context.

Note

If you are using SSN and an IJTAG graybox model exists in the TSDB, the create_patterns_specification command uses the graybox model instead of the full view.

This command returns the name of the PatternsSpecification wrapper if it was just created in memory. This makes it easier to modify using the configuration data editing and introspection commands listed in Table 13-1 on page 3843 as shown in the example below.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The DftControlSettings and the ProcedureStep wrapper are auto filled by the command. Refer to the Patterns section for a description of where the DftControlSettings list comes from. Refer to the documentation of the tessent_default_procedure_name and tessent_default_procedure_parameters attributes for a description of their influence on the ProcedureStep inference.

There is a special case for the port associated with the 'scan_en' signal where, in addition to this port being added to the PatternsSpecification/AdvancedOptions/ConstantPortSettings, it is also added to the Patterns/AdvancedOptions/ConstantPortSettings but only for the JtagBscan patterns. In this AdvancedOptions/ConstantPortSettings wrapper, the port is added with the value "-" to have this port not set to a constant value. This way, the boundary scan tests can toggle this port during the tests ensuring proper coverage of the boundary scan cell.

Any port having the attribute 'function' set to 'power' or 'ground' is added to the AdvancedOptions/ConstantPortSetting wrapper by the create_dft_specification command but only if that port is not already listed in the 'tessent_power_port_list' or 'tessent_ground_port_list' attribute of the ICL.

Limitations When Using the -testbench_files Option

The created PatternsSpecification captures an accurate view for the current design and child blocks only for the following:

- All ATPG patterns with and without SSN.
- Retargeting patterns with SSN.

In the above, and for the child block instances that are not the target of a testbench, the tool uses their scan graybox view when the IJTAG graybox view is not available.

For testbenches generated for retargeting without SSN, the information used by create_patterns_specification in the testbench file may be inaccurate when the EDT channels do not follow the IJTAG network. In this case, you must modify the related Patterns/SimulationOptions/LowerPhysicalBlockInstances to load the child-blocks' correct views that allow all the signals to reach the retargeted instance being simulated.

The same limitation applies to IJTAG testbenches generated with "open_pattern_set ... close_pattern_set" followed by the "write_patterns -verilog -pattern_sets ..." command. It is also the case for testbenches generated with "write_patterns -verilog -vector_set ..." command.

Arguments

- *usage*

  Specifies the intended usage of the patterns. The permitted values are as follows:

  signoff — Indicates that simulation signoff patterns are requested. Signoff patterns are patterns that are used for simulation and verification of the design. They only include

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

a sub-set of patterns by default. With this usage, the ICL file is analyzed to obtain the list of instruments and BIST controllers present in the design. A set of Patterns wrapper is created per instrument type to fully verify their proper operation inside the design. The properties found inside the SignOffOptions wrapper are used to configure how the patterns are created. By default, only the instruments and the BIST controllers found in the top physical instance are simulated. You can set the "simulate_instruments_in_lower_physical_instances" property inside the SignOffOptions wrapper to also request that the instruments and the BIST controllers found in the lower physical instance be re-simulated from the current level. To help with simulation time, those simulation patterns are created so as to have one Patterns wrapper per instrument type/per lower physical instance that enables the simulation to be done using only the full view of one lower physical instance and an empty view for the rest. Refer to the description of the LowerPhysicalBlockInstances wrapper found inside the SimulationOptions wrapper as well as the description of the *design_name.format_interface* file.

In the signoff simulation, the memory BIST controllers (including ROM BIST controllers) are simulated using the MemoryBist/reduced_address_count property set to On. This assumes that the memory library description has been generated by a certified memory library generator and is correct by construction. If the memory library was created or modified manually, you must simulate the full address space on the memory BIST assembly module, located in the instruments directory, using the flow mentioned below. Because the top module name ends with "_mbist_assembly", the create_patterns_specification command knows to not set the MemoryBist/reduced_address_count property to "on" so as to simulate the full address space of the memory.

```
> set_context patterns
> read_icl path/design_name_design_id_mbist_assembly.icl
> set_current_design design_name_design_id_mbist_assembly
> create_patterns_specification
> process_patterns_specification
> run_testbench_simulations
> check_testbench_simulations
```

manufacturing — Indicates the manufacturing patterns are requested. Manufacturing patterns are patterns generated for the ATE. They are a complete set of patterns by default. With this usage, the ICL file is analyzed to obtain the list of instruments and BIST controllers present in the design. A set of Patterns wrappers is created per instrument type to properly exercise the various test features during manufacturing test. The properties found inside the ManufacturingOptions wrapper are used to configure how the patterns are created. Based on the max_async_clock_sources property value, the patterns are organized to optimize parallelism without exceeding the amount of independent frequency clocks your testcase can supply. Unlike with the original LV flow where you could only have 0 or an unlimited amount of asynchronous clocks, you can know and precisely specify the exact amount of asynchronous clocks available on your tester; the PatternsSpecification is optimized for that constraint.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -pattern_id *pattern_id*

  An optional switch and string pair that specifies the *pattern_id* to use in the created PatternsSpecification(*design_name*, *design_id*, *pattern_id*). The default value is the value of the usage command line option: "signoff" or "manufacturing." If the top-level ICL module was extracted using the "extract_icl -variant_icl_id" command, the tool appends the specified variant ICL id prefixed with an underscore to the pattern id.

  However, the default value depends on contextual information if you also specify a testbench file with the "-testbench_files" option:

  - The default value is the "scan_*current_mode*" if the context is "patterns -scan" or "patterns -scan_retargeting", where *current_mode* is the value returned by the get_current_mode command. The tool does not append the variant ICL id to the *current_mode* name, as the *current_mode* name is likely to already include it.
  - Otherwise, the default value is "testbench_*usage*", where *usage* is the value specified by the usage command. If the "extract_icl -variant_icl_id" command created the top-level ICL module, the tool also appends the specified variant ICL id prefixed with an underscore to the default pattern id.

- -replace

  With this option, the error that would normally be generated is ignored when running the command where the PatternsSpecification wrapper with the same *design_name* and *design_id* and the *pattern_id* already exists in memory. The enables the PatternsSpecification wrapper present in memory to be replaced with the new one.

- -ignore *ignored_instrument_list*

  An optional switch and string list that suppresses generation of the default wrapper for the specified instrument. The only valid instrument for ignored_instrument_list is "ssn", which suppresses generation of the default PatternsSpecification()/Patterns(SSN)/ SSNContinuityVerify(default) wrapper even if SSN is present in the hardware.

- -bsdl_files *bsdl_file_list*

  An optional switch and value pair that specifies the BSDL file list to be used in creating the patterns specification for a "BSDL-only" design flow.

  In addition to an absolute or relative path to the BSDL files, you can use the following two syntactical forms:

  - dft_inserted_design(<***module_name***>_<***design_id***>)/<***bsdl_leaf_file_name***>
  - instrument(<***module_name***>_<***design_id***>_<*instrument_type*>)/ <***bsdl_leaf_file_name***>

  In the second form, *instrument_type* is optional. The default value is bscan.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

If you do not specify this argument, the tool uses the BSDL files specified with the "read_icl -bsdl_file" or "read_bsdl *bsdl_file_path*" commands. See the "BSDL-Only Flow" section of the *Tessent BoundaryScan User's Manual* for more information.

- -testbench_files *file_path_list*

  An optional switch and string list that specifies a list of testbench files generated by the "write_patterns -verilog ..." command. Each entry of the *file_path_list* can include globbing characters (* for one or more characters, ? for a single character) for the filename.

  For example, "-testbench_files [list scan_testbenches/*.v ijtag_testbenches/*.v]" finds all the testbenches ending in ".v" under the *scan_testbenches* and *ijtag_testbenches* sub-directories.

  file_path_list — When you supply a valid list of testbenches as the file_path_list, the created PatternsSpecification only includes Patterns wrapper entries featuring the TestbenchVerify sub-wrapper. One Patterns/TestbenchVerify entry is created for each testbench file in the file_path_list. When processed with the process_patterns_specification command, this PatternsSpecification generates the required information to enable you to simulate those testbenches with the run_testbench_simulations command.

  The create_patterns_specification command uses the information automatically generated by the "write_patterns -verilog ..." command in the <*testbench_file_name*>*.info.dict* file. The tool uses the information in this file to populate the Patterns/SimulationOptions/LowerPhysicalBlockInstances for each Patterns wrapper in the created PatternsSpecification. This ensures that the simulator compiles and uses the correct design view for each sub-block of the design. For the current design, the view is automatically set in the Patterns/SimulationOptions/current_design_view entry of the created PatternsSpecification.

- -infer_loopbacks_during_icl_verify { on | off }

  An optional property that turns off the inference of Loadboard loopback pairs. The tool infers these loopbacks when there is one or more ScanInterface of type Host in the current design used by IJTAG. These loopbacks enable performing verification of the Host ScanInterface before connecting to a Client ScanInterface at the next level up.

- -infer_loopbacks_during_memory_bisr_verify { on | off }

  An optional property that turns off the inference of Loadboard loopback pairs. The tool infers these loopbacks when there is one or more BISR Chain Interface of type Host in the current design. These loopbacks enable performing verification of the BISR chains before connecting to an external BISR chain at the next level up.

- -infer_loopbacks_during_boundary_scan_verify { on | off }

  An optional property that turns off the inference of Loadboard loopback pairs. The tool infers these loopbacks when there is one or more Boundary Scan Chain Interface of type

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Host in the current design. These loopbacks enable performing verification of the Boundary Scan chains before connecting to external Boundary Scan chains at the next level up.

- -bscan_sim_views { full | ijtag_graybox }

  An optional switch and value pair to specify which design view to use for the boundary scan simulation. The tool analyzes the design and creates a list of all physical blocks containing the boundary scan chain.

  full — The tool selects the full view of those physical blocks to perform the boundary scan simulation. This is the default.

  ijtag_graybox — The tool selects the IJTAG graybox view of the required physical blocks. Use this option to speed up boundary scan simulations.

  Tip

  A tiling design flow can include a large amount of logic in the simulation. For shorter simulation time, use Tessent Boundary Scan to implement your embedded boundary scan segments and top-level boundary scan chains. Also, use the "-create_ijtag_graybox on" switch when running the extract_icl command. If you have custom boundary scan segments, you can set the preserve_in_graybox attribute before running the "extract_icl -create_ijtag_graybox on" command to force their inclusion in the IJTAG graybox model. The generated IJTAG graybox views contain all the logic needed to operate the boundary scan segments when the above conditions are met. This enables you to use the -bscan_sim_views switch to specify that the tool simulates boundary scan using the IJTAG graybox view for all physical blocks identified, significantly speeding up the simulation.

- -separate_tap_and_bscan_patterns { on | off }

  An optional switch and literal pair that enables you to create separate boundary scan patterns for the TAP and boundary scan chain. The tool creates two wrappers instead of one: one to run all tests for the TAP, and another wrapper to run all boundary scan chain tests. The default is off.

  This is useful if your boundary scan chain or pads need special initialization. The TAP patterns do not need this, so separating it avoids problems with the test_logic_reset run test resetting the chip.

  Note

  The tool silently ignores this switch when creating a PatternsSpecification from a tcd_bscan description in the EBScan flow.

Examples

Example 1

The following example calls the create_patterns_specification command and uses the configuration editing commands to insert in all memory BIST pattern wrappers a DftControlSetting wrapper above the first TestStep wrapper that includes a DEBUG macro definition.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
> set spec [create_patterns_specification]
> set mbist_pats [get_config_el Patterns(MemoryBist_*) -in $spec -silent]
> foreach_in_col mbist_pat $mbist_pats {
>   set first_test_step [get_config_value TestStep<0> -in $mbist_pat -obj]
>   read_config_data -before $first_test_step -from_string {
      SimulationOptions {
        SimulationMacros {
          DEBUG;
        }
      }
    }
}
```

Example 2

In the following example, the *my_dft_signal* DFT signal initializes the BoundaryScan chain to the correct mode for the BoundaryScan testing. If this is added to a pattern that does an iReset, such as the test_logic_reset_test, the initialization is lost for all subsequent tests. Separating the TAP and BoundaryScan chain tests with the -separate_tap_and_bscan_patterns option solves this problem.

ANALYSIS>setspec[create_patterns_specification-separate_tap_and_bscan_patternson]

```
//  sub-command: create_patterns_specification
-separate_tap_and_bscan_patterns on
//  Creating '/PatternsSpecification(top,rtl,signoff)'
//    Getting patterns specifications for the 'ijtag' instrument type
//    Getting patterns specifications for the 'jtag_bscan' instrument \
//      type /PatternsSpecification(top,rtl,signoff)
```

**SETUP> add_config_element -in_wrapper **
**$spec/Patterns(JtagBscanPatterns)/DftControlSettings my_dft_signal **
-value 1

SETUP> report_config_data $spec

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
PatternsSpecification(top,rtl,signoff) {
  Patterns(ICLNetwork) {
    ICLNetworkVerify(top) {
    }
  }
  Patterns(JtagTapPatterns) {
    tester_period : 100ns;
    TestStep(JtagBscanTestStep) {
      BoundaryScan {
        RunTest(test_logic_reset) {
        }
        RunTest(inst_reg) {
        }
        RunTest(bypass_reg) {
        }
      }
    }
  }
  Patterns(JtagBscanPatterns) {
    tester_period : 100ns;
    DftControlSettings {
      my_dft_signal : 1;
    }
    TestStep(JtagBscanTestStep) {
      BoundaryScan {
        RunTest(bscan_reg) {
        }
        RunTest(input) {
        }
        RunTest(sample) {
        }
        RunTest(disbaled_outputs) {
        }
        RunTest(highz_mode) {
        }
        RunTest(clamp) {
        }
        RunTest(output) {
        }
      }
    }
  }
}
```

Related Topics

apply_specification_defaults

process_patterns_specification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_pin

Context: all contexts

Mode: insertion

Creates a pin on an instance of a design module.

Usage

create_pin ***hierPinName***
[-on_instance *instance_object*] [-direction input | output | inout]
[-ascending] [-silent]

Description

If the "set_insertion_options -auto_uniquify_edited_modules" option is On, the module is first uniquified before creating a pin on the specified instance. If the -auto_uniquify_edited_modules option is Off, the pin is created on the module, and all instances of the module get the newly created pin.

This command returns a collection of the pin objects created; if hierPinName is a bus, this command returns a collection with one pin object for each bus bit, beginning with the left-most bit of the bus and ending with the right-most.

Arguments

- ***hierPinName***

  A required string that specifies the hierarchical path of the pin to be created concatenated to the leaf pin name. The specified hierarchical path of the pin to be created must exist in the design.

- -on_instance *instance_object*

  An optional switch and string pair that specifies a single instance for the newly created pin. The ***hierPinName*** is then a name relative to *instance_object*.

- -direction input | output | inout

  An optional switch and literal pair that specifies the direction of the newly created pin. An input pin is created by default when this option is omitted.

- -ascending

  An optional switch that specifies the range direction type for a VHDL bus port. By default, the range direction type is inferred as "downto" when the left index is larger than the right index, and as "to" when the left index is smaller than the right index. You use this switch to specify that the VHDL port notation is "to" when the left index is equal to the right index; otherwise, the tool uses the "downto" range direction type. An error is generated if you specify this option and the left index is larger than the right index, or if the object name is not a bus. The -ascending option is only permitted in the rtl context.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -silent

  An optional argument that specifies to not generate an error message if the specified hierPinName already exists or when the hierarchical part of the specified name is not an existing instance within the current design. When an error is suppressed, the command returns an empty collection.

Examples

Example 1

The following example creates a 5-bit bus input pin called A on the instance u1. The command returns a collection of newly created pin objects.

create_pin u1/A[5:1]

```
{u1/A[5] u1/A[4] u1/A[3] u1/A[2] u1/A[1]}
```

Example 2

The following example introspects the direction attribute of the four newly created pin objects.

get_attribute_value_list [create_pin u1/C[3:0] -direction output] -name direction

```
{output output output output}
```

Example 3

The following example creates a pin on a specific instance.

create_pin newPin -on_instance i1/i2

```
i1/i2/newPin
```

You can use the -on_instance switch to create pins on an instance where you already have a collection.

set instance [get_instances i1/i2]
create_pin newPin -on_instance $instance

Related Topics

delete_pins

get_icl_ports

get_icl_pins

get_nets

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_port

Context: dft

Mode: insertion

Creates a port on the specified design module.

Usage

create_port ***port_name*** [-on_module *obj_spec*] [-direction input | output | inout] [-ascending] [-allow_existing] [-silent]

Description

By default, this command creates a port on the design module in the current design, If you specify the -on_module argument, this command creates a port on the module specified in obj_spec.

This command returns a collection of the port objects created; if port_name is a bus, this command returns a collection with one port object for each bus bit, beginning with the left-most bit of the bus and ending with the right-most.

If the specified port already exists the tool generates an error. You can specify the -silent switch to instruct the tool not to generate an error.

Arguments

- ***port_name***

  A required value that specifies the name of the port to be created. Bus ports must be created with a single create_port command where port_name includes the required bus range.

- -on_module *obj_spec*

  An optional switch and value pair that specifies the module to which the port is to be added. The obj_spec can be the name of a module or a module object returned by the get_module command. If the -on_module switch is not specified, this command adds the port to the current design, which is the module specified by the set_current_design command. You cannot create a port on a module that was read in from a file and that is not instantiated inside the current design. It is possible to create ports on modules created with the create_module command.

- -direction input | output | inout

  An optional switch and literal pair that specify the direction of the newly created port. An input port is created when this option is omitted.

- -ascending

  An optional switch that specifies the range direction type for a VHDL bus port. By default, the range direction type is inferred as “downto” when the left index is larger than the right index, and as “to” when the left index is smaller than the right index. You use this switch to specify that the VHDL port notation is “to” when the left index is equal to the right index;

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

otherwise, the tool uses the “downto” range direction type. An error is generated if you specify this option and the left index is larger than the right index, or if the object name is not a bus. The -ascending option is only permitted in the rtl context.

- -allow_existing

  An optional switch specifying that if a port with a given name exists, the tool returns that port instead of issuing an error.

- -silent

  An optional switch that specifies to not generate an error message if the specified port_name already exists. When an error is suppressed, the command returns an empty collection.

Examples

Example 1

The following example creates a 5-bit bus input port called A on the module that is the current design. The command returns a collection of the newly created port objects.

create_port A[5:1]

```
{A[5] A[4] A[3] A[2] A[1]}
```

Example 2

The following example creates a scalar output port called B on module ModB.

create_port B -on_module ModB -direction output

```
{B}
```

Example 3

The following example creates a 4 -bit bus output port called C on the current design and uses the get_attribute_value_list command to query the direction attribute of the newly created port objects.

get_attribute_value_list [create_port C[3:0]] -name direction

```
{input input input input}
```

Related Topics

delete_ports

get_icl_ports

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_reversible_patterns

Context: patterns -scan, patterns -scan_diagnosis

Mode: analysis

Generates reversible test patterns.

Usage

create_reversible_patterns **-shift_direction_port** ***shift_direction_port***
-output ***output_pattern_file*** **-pattern_format** {**STIL** | **WGL**}
[-begin_pattern *integer*] [-end_pattern *integer*]
[-test_setup_off] [-chain_mask *chain_mask_file_location*]
-reverse_scan_config ***reverse_scan_config_file_location***

Description

Use this command to generate reversible test patterns.

Arguments

- **-shift_direction_port** ***shift_direction_port***

  A required switch and string that specifies the name of the port used to change the direction of the reversible chain.

- **-output** ***output_pattern_file***

  A required switch and string that specifies the name of the output pattern file.

- **-pattern_format** {**STIL** | **WGL**}

  A required switch and string that specifies the format of the output pattern file. You can choose either STIL and WGL.

- -begin_pattern *integer*

  An optional switch and integer that specifies the beginning of the chain pattern set range to be extended into reversible patterns.

- -end_pattern *integer*

  An optional switch and integer that specifies the end of the chain pattern set range to be extended into reversible patterns.

- -test_setup_off

  An optional switch that specifies to exclude test setup chain patterns from the reversible pattern set.

- -chain_mask *chain_mask_file_location*

  An optional switch and string that specifies the location of a text file that lists the chains to mask.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-reverse_scan_config** ***reverse_scan_config_file_location***

  A required switch and string that specifies the location of input scan path report (.spr) file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_scan_chain_family

Context: dft -scan

Mode: analysis

Creates a scan_chain_family object, which controls how one or more scan chains are allocated/connected during distribution.

Usage

create_scan_chain_family ***family_name***
{ -include_elements ***obj_spec*** | -ordered_elements ***obj_spec*** }
[-edt_instances *obj_spec* |
{-ssh_instance *obj_spec* [-ssh_chain_groups *ssh_chain_group_list*]} |
{-si_connections *obj_spec* -so_connections *obj_spec* }]
[-si_associated_ports *obj_spec* -so_associated_ports *obj_spec*]
[-attribute_set_for_chain_assembly { *attribute_list* }]
[-single_attribute_set_chains { on | off }]
[-single_class_chains { on | off }]
[-single_clock_domain_chains { on | off }]
[-single_clock_edge_chains { on | off }]
[-single_power_domain_chains { on | off }]
[-single_cluster_chains { on | off }]
[-single_wrapper_type_chains { On | Off }]
[-port_index_start_value *integer*]
[-port_scalar_index_modifier *integer*]
[-chain_length { *integer* | unlimited }]
[-chain_count { *integer* | unlimited }]
[-si_port_format *format*]
[-so_port_format *format*]
[-si_timing { leading_edge | trailing_edge | any_edge }]
[-so_timing { leading_edge | trailing_edge | any_edge }]
[-lockup_cell_type { latch | flop }]
[-si_lockup_cell_type { latch | flop }]
[-so_lockup_cell_type { latch | flop }]
[-unused_chain_connection_handling { auto | insert_pipelining_flop |
tie_off_chain_output | dont_touch }]
[-si_pipelining { wrapper_chains_only | all_chains | off }]
[-si_pipelining_type {non_scan_flop | holding_scan_flop}]

Description

Use this command to create a scan_chain_family object, which controls how one or more scan chains are allocated/connected during distribution. This is an optional/advanced command that enables you to fine-tune how sub-populations of a mode get treated.

Refer to Chain Port Sharing and Scan Chain Family in the *Tessent Scan and ATPG User's Manual* for usage examples.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***family_name***

  A required string that specifies a scan chain family name.

- -include_elements ***obj_spec***

  A required switch (if -ordered_elements is not used) and value pair that specifies a list of mandatory scan elements to include in this chain family. It has no default value.

- -ordered_elements ***obj_spec***

  A required switch (if -include_elements is not used) and value pair that specifies an ordered list of mandatory scan elements to include in this chain family. It has no default value.

- -edt_instances *obj_spec*

  A switch and value pair that specifies EDT instances to connect the scan chains to. When core descriptions for the specified instances are detected, then the tool automatically sets the -si/ so_connections to the instances scan pins.

- -ssh_instance *obj_spec* [-ssh_chain_groups *ssh_chain_group_list*]

  Optional string and literal pair to enable you to identify a single ScanHost node instance for the add_scan_mode command. When the tool detects a core description for the specified instance, it automatically sets the -si/so_connections to the scan pins of the instance.

  -ssh_chain_groups *ssh_chain_group_list* — specify one or more subsets of the ScanHost connections using the SSH chain group ID(s). All ScanHost connections are selected when no SSH chain group ID is specified.

- -si/so_connections *obj_spec*

  A switch and value pair that specifies connections to existing SI/SO pins or ports. The specified value can be a Tcl list of one or more port/pin names, a collection of one or more pin/port objects, or a port/pin Verilog name (of a certain width). The specified values corresponds to existing top-level input port(s) or output pin(s) for *-si_connections* and existing top-level output port(s) or input pin(s) for *-so_connections*.

- -si_associated_ports *obj_spec* -so_associated_ports *obj_spec*

  Optional string and literal pairs that specify existing top-level port(s) associated to the internal connections identified using -si/so_connections. These are used in the .tcd file to describe the scan ports of the chains at the core boundary.

- -attribute_set_for_chain_assembly { *attribute_list* }

  An optional switch used to specify one or more string attributes to control scan chain assembly. When you use this switch and specify several string attributes, the tool creates clusters for every value combination of the specified attributes. For example, assume that

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

you have defined the following quadrant and testpoint_type name attributes and that all scan elements are attributed accordingly:

```
>register_attribute \
-name quadrant \
-value_type string \
-object_types scan_element \
-enum { NE NW SE SW }

>register_attribute \
-name testpoint_type \
-value_type string \
-object_types scan_element \
-enum { control observation }
```

Scan elements that are not test points are not assigned a “testpoint_type” attribute value, whereas all scan elements are assigned a “quadrant” attribute value. This implies that each scan element has one of the following attribute value combinations:

Table 3-20. Attribute Value Combinations

“quadrant” attribute value	“testpoint_type” attribute value
NE	control
NE	observation
NE	--
NW	control
NW	observation
NW	--
SE	control
SE	observation
SE	--
SW	control
SW	observation
SW	--

Accordingly, you can specify a scan chain family that creates separate scan chains for each of the above 12 attribute value combinations as follows:

```
>create_scan_chain_family foo \
-attribute_set_for_chain_assembly { quadrant testpoint_type }
```

- -single_attribute_set_chains { on | off }

  An optional switch and literal pair that restricts a scan chain to a single set of attribute values. The default is on.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -single_class_chains { On | Off }

  An optional string and literal pair that controls whether to mix wrapper and non-wrapper cells in a single chain. By default, the tool keeps wrapper cells separate from non-wrapper chains for internal and external type of scan modes. However, for unwrapped mode they can be mixed.

- -single_clock_domain_chains { On | Off }

  An optional string and literal pair that specifies whether to restrict a scan chain to a single clock domain. The default is off.

- -single_clock_edge_chains { On | Off }]

  An optional string and literal pair that specifies whether to restrict a scan chain to cells having a single edge. The default is off.

- -single_power_domain_chains { On | Off }

  An optional string and literal pair that specifies to restrict a scan chain to a single power domain using the built-in attribute "power_domain_island". The default is off. For attribute details, refer to the table "Built-in Attributes on Scan Element (SE) Object Type" in the "Scan Element Object Type" on page 3813.

- -single_cluster_chains { On | Off }

  An optional string and literal pair that specifies to restrict a scan chain to a single cluster using the built-in attribute "cluster_name". The default is on. For usage examples, refer to Scan Mode and Scan Insertion for Unwrapped Core sections in the *Tessent Scan and ATPG User's Manual*. For attribute details, refer to the table "Built in Attributes on Scan Element (SE) Object Type" in the Scan Element Object Type section of the *Tessent Shell Reference Manual*.

- -single_wrapper_type_chains { On | Off }

  An optional switch and literal pair that specifies that "-single_class_chains ON" controls whether to mix input wrapper and output wrapper cells in the same chain. The default is off.

- -port_index_start_value *integer*

  An optional switch and integer that specifies the starting port index. The starting index is 0 by default, but can be overridden by this option.

- -port_scalar_index_modifier *integer*

  An optional switch and integer that specifies the increment used between each chain port indices, the default is 1.

- -chain_length {*integer* | unlimited }

  An optional integer that specifies the target length of scan chains for a particular chain family. By default, the no limit (unlimited) is imposed on the chain length. The minimum scan chain length is 1.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -chain_count { *integer* | unlimited }

  An optional integer that specifies the number of scan chains that should be constructed for the specified chain family. By default, the chain count is set to the number of connections specified with the -si/so_connections or by the number of pins of the EDT controller(s) specified with the -edt_instances option, otherwise if the latter options were not used, then no limit (unlimited) is imposed on the chain count.

- -si/so_port_format *format*

  An optional string that specifies the port format. The default si/so_port_format is set to "ts_%s_si/so[%d]" when there are more than one scan mode, otherwise it defaults to "ts_si/so[%d]". The mode placeholder "%s" gets replaced by the chain mode name, whereas the index placeholder "%d" gets substituted by the chain index. A precision number may be used when specifying the index placeholder, for example the format "lsi_si%2d" would cause port names such as "lsi_si00", "lsi_si01" to be created. Note that leading zeros are inserted automatically when needed making "%2d" implicitly equivalent to "%02d".

- -si_timing { <u>leading_edge</u> | trailing_edge | any_edge }

  An optional string and literal pair that provides the ability to control the capture edge of the first scan cell in a chain. The tool inserts a lockup (re-timing) cell between the scan chain input pin and the first scan cell in the chain to control the capture edge of the first cell as leading or trailing edge. No lockup cell is inserted if "any_edge" is provided as the argument. The default edge type is leading_edge.

- -so_timing { leading_edge | <u>trailing_edge</u> | any_edge }

  An optional string and literal pair that provides the ability to control the change edge of the last scan cell in a chain. The tool inserts a lockup (re-timing) cell between the scan chain output pin and the last scan cell in the chain to control the change edge of the last cell as leading or trailing edge. No lockup cell is inserted if "any_edge" is provided as the argument. The default edge type is "trailing_edge".

- -lockup_cell_type { <u>latch</u> | flop }

  An optional switch and string that controls the type of lockup (re-timing) cell to use between timing-constrained scan elements of new scan chains. The default cell type is latch

- -si_lockup_cell_type { <u>latch</u> | flop }

  An optional switch and string that controls the type of lockup (re-timing) cell to use between the scan chain input pin and the first scan cell. The default cell type is latch

- -so_lockup_cell_type { <u>latch</u> | flop }

  An optional switch and string that controls the type of lockup (re-timing) cell to use between the scan chain output pin and the last scan cell. The default cell type is latch.

Note

By default, the tool adds a pipeline flop in front of the wrapper chains to facilitate launching a transition from the first scan cells in the wrapper chains.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -unused_chain_connection_handling {auto | insert_pipelining_flop | tie_off_chain_output | dont_touch}

  An optional switch and literal pair that selects how to handle unused scan in and scan out connections. Options are the following:

  auto — selects "insert_pipelining_flop" for EDT controllers with bypass chains, "tie_off_chain_output" for all other EDT controllers, and "dont_touch" for other connections. This option is the default.

  insert_pipelining_flop — inserts scan chains composed of a single pipelining flop.

  tie_off_chain_output — ties off the so_connection or the EDT compactor input when you specify an EDT instance.

  dont_touch — do not touch the unused scan chain connections.

- -si_pipelining { wrapper_chains_only | all_chains | off }

  An optional switch and literal that controls whether the tool inserts a pipelining flop at the beginning of each chain if the first scan element of the chain does not have the attribute "no_si_pipelining" set and is not an OCC segment. When set to all_chains, the tool adds a pipelining flop to all scan chains. When set to wrapper_chains_only, the tool adds pipeline flops only at the beginnings of chains that begin with a wrapper flop in internal or external mode. The default is wrapper_chains_only when you have specified "set_scan_insertion_options -se_pipeline_count 0" (the default value for this switch), but the default changes to all_chains when you have specified "set_scan_insertion_options -se_pipeline_count 1".

- -si_pipelining_type {non_scan_flop | holding_scan_flop}

  An optional switch and string that controls whether to implement pipelining using a simple non-scan data flop or a holding flop constructed using a scan flop with its output connected to its data input. The default is holding_scan_flop.

Related Topics

add_scan_mode

delete_scan_chain_families

get_scan_chain_families

set_scan_insertion_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_scan_segment

Context: dft -scan

Mode: analysis

Creates a spec scan segment that enables you to force sequences of scan elements to be stitched together within scan chains.

Usage

create_scan_segment **spec_segment_name**
{***-include_elements <obj_spec>*** | ***-ordered_elements <obj_spec>*** }
[*-head_elements <obj_spec>*]
[*-tail_elements <obj_spec>*]

Description

You can use this command to create a spec segment, which is a virtual segment that does not exist physically but which you can specify logically. It is a non-dividable scan element of type "segment" that has its "exists" attribute set to false (in contrast, the attribute is set to true for the existing scan segment). Another distinction from an existing scan segment is that the scan elements inside the spec segment are not scan connected, the tool does this during scan insertion.

You can define spec segments during the system mode analysis by specifying usable scan element leaves and segments. Scan elements included inside a spec segment get the status "usable_indirectly".

The tool treats spec segments in the same manner as existing segments during wrapper analysis. Accordingly, if a spec segment contains elements of mixed classes, then the tool automatically promotes any of its "core" elements to "wrapper" elements.

Note

If you create or delete spec segments after analyze_wrapper_chains and/or analyze_scan_chains, it triggers the reset of the rest of the wrapper analysis and/or distribution model.

Arguments

- **spec_segment_name**

  A required string that specifies the name of the spec segment to create.

- ***-include_elements <obj_spec>***

  An optional switch that specifies a list of required scan elements to include in the segment. The tool manages the ordering for the segment and may reorder the elements during scan insertion to minimize the use of retiming elements.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ***-ordered_elements <obj_spec>***

  An optional switch that specifies an ordered list of required scan elements to include in the segment.

- -head_elements <obj_spec>

  An optional switch that specifies a specific subset of ordered elements at the beginning of the segment.

- -tail_elements <obj_spec>

  An optional switch that specifies a specific subset of ordered elements at the end of the segment.

Examples

Example 1

This example creates a spec segment with three flops ordered in a specific manner.

```
ANALYSIS> create_scan_segment my_ordered_segment  \
    -ordered_elements { /df_03 /df_02 /df_01 }
```

Example 2

This example creates a spec segment that begins with three flops ordered in a specific manner, followed by the flops found below instance B_i (which are ordered by the tool to minimize the insertion of retiming cells).

```
ANALYSIS> create_scan_segment my_complex_segment \
   -head_elements { /df_04 /df_05 /df_06 } \
   -include_elements [get_scan_elements -below_instances B_i]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_silicon_insight_setup_specification

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Generates a default Tessent SiliconInsight setup specification.

Usage

create_silicon_insight_setup_specification [-setup *mode_name*]

Description

Generates a default Tessent SiliconInsight setup specification wrapper called SiliconInsightSetupSpec and saves it to the current working directory in a file named *SiliconInsightSetupSpec.cfg*.

The command returns the name of created specification. For example:

```
set si_spec [create_silicon_insight_setup_specification]
set_config_value [get_config_elements selected_setup \
    -in_wrapper $si_spec] Sid(signalyzerH4)
```

The setup specification contains Tessent SiliconInsight-specific data that defines the pin map, testers, adaptor types, and other data that are required to prepare Tessent SiliconInsight to communicate with a DUT, simulator process, or ATE.

By default, the tool generates a specification that includes all possible setups and with No_Tester(batch_cdp) as the current setup. (This mode enables generating a CDP only.) Specify the -setup option to generate the default setup specification only for the mode you specify.

For information about the Tessent SiliconInsight setup specification, refer to "Prepare the Design Under Test" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

Arguments

- -setup *mode_name*

  An optional switch and string pair that specifies to create a default setup specification for the specified Tessent SiliconInsight Desktop mode of operation. For *mode_name*, you can specify:

  jtagkey

  flyswatter2

  olimex_arm_usb_ocd

  olimex_arm_usb_ocd_h

  signalyzer

  signalyzerSHA40

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

signalyzerH2

signalyzerH4

signalyzerSP

opalKellyXem6310

simdut

Examples

The following example creates a setup specification to use the Tin Can Tools Flyswatter2 adaptor as the Tessent SiliconInsight Desktop connection to the performance board.

create_silicon_insight_setup_specification -setup flyswatter2

The tool generates SiliconInsightSetupSpecification with the default properties for this adaptor, which you can edit as needed. For details about the SiliconInsightSetupSpecification configuration data syntax, refer to "SiliconInsightSetupSpecification" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

```
SiliconInsightSetupSpecification {
   selected_setup : Sid(flyswatter2);
   Protocol {
      SvfTapPins {
        TRST : trst;
        TDI : tdi;
        TMS : tms;
        TDO : tdo;
        TCK : tck;
      }
   }
   Sid(flyswatter2) {
      cdp_directory : top.cdp;
      flywatter2 {
         PinMap {
            TDI : tdi;
            TMS : tms;
            TDO : tdo;
            TCK : tck;
         }
      }
   }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_ssn_continuity_patterns

Context: patterns -ijtag

Mode: analysis

Creates an SSN continuity pattern to test the internal path of the ScanHost from input to output.

Usage

create_ssn_continuity_patterns [-include_ssh_icl_instances *included_ssh_instances*]
[-ssn_datapath_ids *ssn_datapath_ids*]
[-custom_pattern_sequence *custom_pattern_sequence*]
[-verify_datapaths_simultaneously {on | off}]

Description

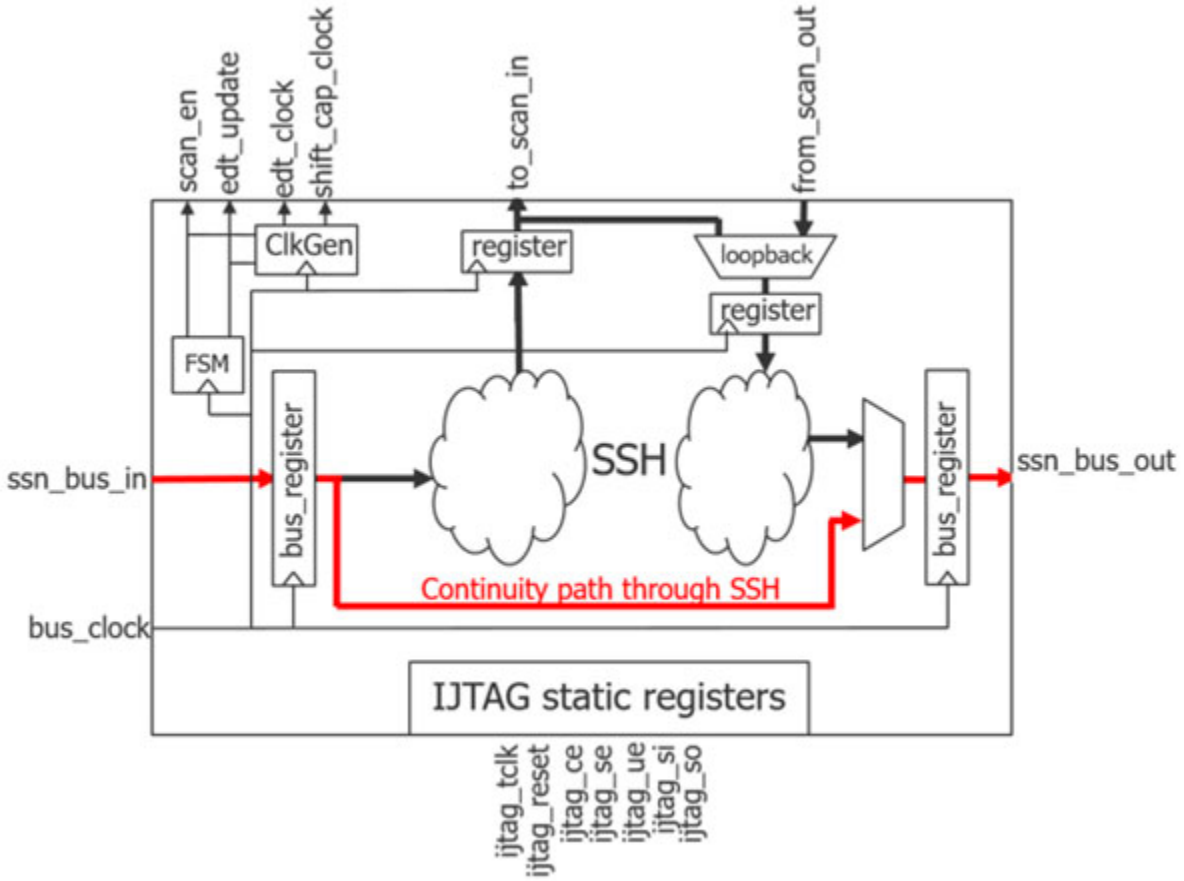
The continuity pattern verifies the SSN datapath. SSN datapaths from ssn_bus_in to ssn_bus_out are verified at the chip level. A new run of create_ssn_continuity_patterns is required after you configure each datapath. You can also run the SSN continuity pattern at the block level.

The continuity pattern detects deviations in the lengths of the datapaths and accidental shuffling of the datapath bits. It is not SSN packetized scan data.

Figure 3-35 shows the path through the ScanHost node.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 3-35. SSN Continuity Pattern - Internal SSH Paths Coverage



To create the continuity patterns with an existing process_patterns_specification framework, use the create_ssn_continuity_patterns command between open_pattern_set and close_pattern_set. This method stores the results of the pattern generation in an IJTAG pattern set. Use the write_patterns command to save those patterns.

At the chip level, you can use iWrite and iApply to configure the SSN Multiplexer node and change the SSN datapath configuration for the continuity pattern.

Note

Use iCall instead of iWrite if an iProc encapsulates the datapath configuration.

Arguments

- -include_ssh_icl_instances *included_ssh_instances*

  By default, the create_ssn_continuity_patterns command uses all accessible ScanHost instances. Using this optional switch restricts the pattern generation to the datapaths that are required to access the instances you specify with *included_ssh_instances*. The instances you specify in *included_ssh_instances* must be valid ScanHost instances by instrument type and subtype.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -ssn_datapath_ids *ssn_datapath_ids*

  By default, this optional switch uses all accessible SSN datapaths. Using this switch restricts the pattern generation to the datapath IDs you specify in *ssn_datapath_ids*. The ICL must specify the datapath IDs as values of the tessent_ssn_datapath_ids attribute on the primary datapath bus inputs and outputs.

- -custom_pattern_sequence *custom_pattern_sequence*

  By default, the continuity pattern data is a random bit sequence. Using this optional switch enables you to specify that pattern data sequence. If the sequence you define in *custom_pattern_sequence* is shorter than the active SSN datapath, the sequence repeats from the beginning.

- -verify_datapaths_simultaneously <u>on</u> | off

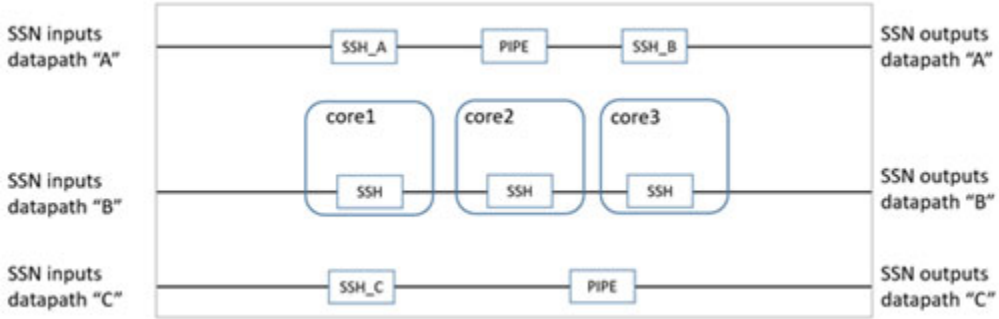
  Set this optional switch to "off" to test SSN datapaths sequentially. By default, all SSN datapaths are tested simultaneously.

Examples

Example 1

The following example shows how you can use the -include_ssh_icl_instances option to limit the test pattern generation to a subset of the SSH instances. Figure 3-36 shows an example design with multiple datapaths.

Figure 3-36. SSN With Multiple Datapaths




```
set_context patterns -ijtag
…
open_pattern_set ssn_verification1 -usage ssn
  create_ssn_continuity_patterns -include_ssh_icl_instances \
    {core1.SSH core3.SSH SSH_C}
close_pattern_set
write_patterns SSNContinuity.stil -pattern_sets ssn_verification1 -stil -replace
```

The generated test vectors contain stimuli and responses for the ports of the datapaths "B" and "C" only because this is the minimal set of datapaths that contains all of the specified SSH instances. The SSH instance core2.SSH is also included in the test because it is on datapath "B" and hence cannot be avoided.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example shows how you can use the -ssn_datapath_ids and -verify_datapaths_simultaneously options to test multiple datapaths one after another. Refer to Figure 3-36.

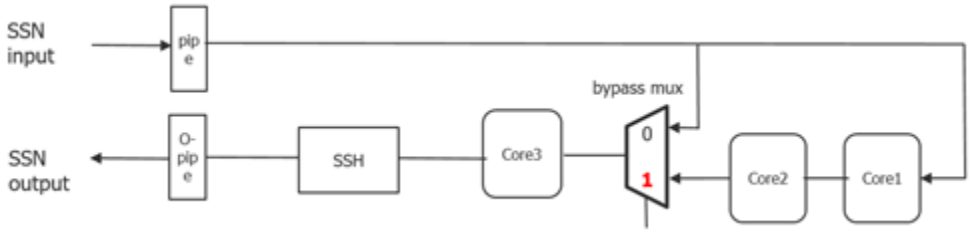
```
set_context patterns -ijtag
…
open_pattern_set ssn_verification2 -usage ssn
  create_ssn_continuity_patterns -ssn_datapath_ids {A B} \
    -verify_datapaths_simultaneously off
close_pattern_set
write_patterns SSNContinuity.stil -pattern_sets ssn_verification2 -stil -replace
```

The -ssn_datapath_ids option instructs the tool to limit the pattern generation to the specified datapaths “A” and “B.” Setting -verify_datapaths_simultaneously to “off” forces the tool to test those datapaths in series.

Example 3

The following chip-level example shows how you can call create_ssn_continuity_patterns twice to generate continuity patterns for the default SSN datapath configuration and an alternative configuration.

Figure 3-37. Example Design With Two SSN Datapath Configurations




```
set_context patterns -ijtag
…
open_pattern_set ssn_continuity_default -usage ssn
  # Select the default datapath (path 0 of bypass_mux)
  create_ssn_continuity_patterns
  # Select the secondary datapath (path 1 of bypass_mux)
  iCall <bypass_mux_inst>.setup select_secondary_bus 0b1
  iApply
  create_ssn_continuity_patterns
close_pattern_set
write_patterns SSNContinuity.stil -pattern_sets ssn_continuity_default \
  -stil -replace
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 4
Command Dictionary (D-F)

The Command Dictionary describes the application commands for Tessent Shell. For quick reference, the commands appear alphabetically with each beginning on a separate page.

The command usage line syntax conventions are common to all products documented in this manual. See "Conventions for Command Line Syntax" on page 97.

Command Descriptions **646**
deassert_icl_sib 653
defer_schematic_updates 654
delete_atpg_constraints 655
delete_atpg_functions 657
delete_bist_capture_ranges 659
delete_black_boxes 660
delete_bsdl 662
delete_bsdl_extraction_option 663
delete_bsdl_extraction_user_instruction 665
delete_capture_procedures 666
delete_cdp_test 668
delete_cell_constraints 669
delete_cell_library 671
delete_cell_models 672
delete_chain_masks 674
delete_clocks 675
delete_config_element 676
delete_config_messages 678
delete_config_tabs 680
delete_connections 681
delete_core_descriptions 683
delete_core_instances 685
delete_design 688
delete_design_partitions 689
delete_dfm 690
delete_dft_clock_enables 692
delete_dft_clock_muxes 694
delete_dft_control_points 696
delete_dft_modal_connections 698
delete_dft_signals 701
delete_edt_blocks 703
delete_edt_configurations 706

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_false_paths 708
delete_fault_sites 711
delete_faults 715
delete_flat_model 727
delete_gate_to_rtl_mapping 728
delete_icl_modules 729
delete_icl_ports 730
delete_icl_scan_interfaces 731
delete_icl_ssn_datapaths 732
delete_iddq_exceptions 733
delete_ijtag_logical_connection 734
delete_input_constraints 736
delete_instances 739
delete_iprocs 741
delete_layout_core_information 743
delete_layout_verification 744
delete_lfsr_connections 746
delete_lfsr_taps 747
delete_lfsrs 748
delete_lists 749
delete_loadboard_loopback_pairs 750
delete_misrs 752
delete_multicycle_paths 753
delete_nets 756
delete_nofaults 758
delete_nonscan_instances 761
delete_notest_points 763
delete_output_masks 766
delete_pade_logical_connection 768
delete_pattern_trigger 770
delete_patterns 771
delete_pins 773
delete_ports 775
delete_primary_inputs 777
delete_primary_outputs 779
delete_processors 781
delete_read_controls 784
delete_register_value 785
delete_rtl_to_gates_mapping 786
delete_scan_chain_families 788
delete_scan_chains 789
delete_scan_elements 790
delete_scan_groups 791
delete_scan_instances 792
delete_scan_modes 794

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_schematic_callout 795
delete_schematic_objects 796
delete_schematic_path 798
delete_simdut_fault. 799
delete_simulation_contexts. 800
delete_simulation_forces 802
delete_synchronous_clock_group. 804
delete_test_points 807
delete_tied_signals 809
delete_wave_generator_pins. 811
delete_write_controls 812
diagnose_failures 813
display_diagnosis_report 818
display_message 819
display_specification 821
dofile 823
echo. 825
elaborate_icl_modules 827
encrypt_layout 828
execute_cdp_test. 830
execute_gpib_command 836
execute_tester_command 839
exit 841
expand_compressed_patterns 842
extract_fault_sites 843
extract_icl 850
extract_inter_cell_data 855
extract_sdc 862
filter_collection. 864
find_design_names 866
foreach_in_collection 872
format_dictionary 875

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Command Descriptions

This section describes the Tessent Shell commands alphabetically, D through F. The beginning of each command description shows the contexts and modes that support the command and provides the following information:

Context: Lists each context and sub-context that supports the command. “All contexts” means that the command is supported in all available contexts. “Unspecified” means that you do not have to set a context to use the command.

Mode: Lists each mode that supports the command. “All modes” means that the command is supported in all available modes. In some cases, a particular mode supports the command only within certain contexts, and this is noted in parentheses after the mode type.

Table 4-1. Commands D Through F

Command	Description
deassert_icl_sib	Context: patterns -silicon_insight -ijtag
defer_schematic_updates	Postpones schematic updates until after the scope introduced by the command ends.
delete_atpg_constraints	Removes state restrictions from the specified pins.
delete_atpg_functions	Removes the specified function definitions.
delete_bist_capture_ranges	Specifies that the tool no longer uses a particular named capture procedure when generating BIST patterns.
delete_black_boxes	Undoes the effect of the add_black_boxes command.
delete_bsdl	Deletes stored BSDL file data for multiple die in a 2.5D design.
delete_bsdl_extraction_option	Deletes values for various BSDL attributes previously set with the set_bsdl_extraction_options command.
delete_bsdl_extraction_user_instruction	Deletes the specified user instructions previously added with the add_bsdl_extraction_user_instruction.
delete_capture_procedures	Removes the specified capture procedures from internal memory.
delete_cdp_test	Removes the specified test from the CDP.
delete_cell_constraints	Removes constraints placed on scan cells.
delete_cell_library	Removes all current cell libraries, the current design including the flat model, and all tool data that is design-specific.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 4-1. Commands D Through F (cont.)

Command	Description
delete_cell_models	Specifies the name of the DFT library cell that the tool is to remove from the active list of cells that the user can access when adding test points or that the tool can access when inserting test logic.
delete_chain_masks	Removes chain masks from the specified scan chains.
delete_clocks	Removes clocks from the clock list.
delete_config_element	Deletes one or more configuration elements. Only elements that can be added can be deleted.
delete_config_messages	Deletes error, warning or info message that were added to configuration elements using the add_config_message command.
delete_config_tabs	Deletes one or more configuration tree tabs from the Config Data Browser window.
delete_connections	Disconnects net, pin, or port objects.
delete_core_descriptions	Removes core descriptions that have been read in using the read_core_descriptions command from memory.
delete_core_instances	Removes an instance binding created with the add_core_instances command.
delete_design	Removes all design modules including the flat model and all tool data that is design-specific.
delete_design_partitions	Deletes partition definitions from a design.
delete_dfm	Deletes a DFM rule from the open LDB.
delete_dft_clock_enables	Deletes one or all previously added DFT clock enable signals.
delete_dft_clock_muxes	Deletes one or all previously added DFT clock multiplexers.
delete_dft_control_points	Deletes one or all previously added DFT control points.
delete_dft_modal_connections	You use the delete_dft_modal_connections command to delete modal connection specifications previously added by the add_dft_modal_connections command.
delete_dft_signals	Deletes a DFT signal that was previously added using the add_dft_signals command.
delete_edt_blocks	Removes the data for the specified EDT block(s) from the tool's internal database.
delete_edt_configurations	Deletes specified compression configurations from all blocks in a design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 4-1. Commands D Through F (cont.)

Command	Description
delete_false_paths	Removes definitions of false paths.
delete_fault_sites	Removes bridge entries, delay paths, user-defined fault sites, and any associated faults from the current fault list. For UDFMs, all faults are removed.
delete_faults	Removes faults from the current fault list.
delete_flat_model	Deletes a flat model that was previously created either by going to system mode analysis or by using the create_flat_model command.
delete_gate_to_rtl_mapping	Removes the specified rule for post-synthesis/post-layout to RTL instance name-mapping for DEF files.
delete_icl_modules	Deletes the specified ICL modules from memory.
delete_icl_ports	Undoes the effect of the add_icl_ports command on a specified list of top level ports.
delete_icl_scan_interfaces	Deletes the specified scan interfaces previously added by add_icl_scan_interfaces from the new ICL top module that is created during ICL extraction.
delete_icl_ssn_datapaths	Deletes the SSN datapath specifications added with the add_icl_ssn_datapaths and set_icl_ssn_datapath_ports commands.
delete_iddq_exceptions	Deletes exceptions to Iddq restrictions added with the add_iddq_exceptions command.
delete_ijtag_logical_connection	Removes logical connections previously added to a design. You must specify either the source pin/port or the destination pin/port or both source and destination pins/ports.
delete_input_constraints	Removes previously applied constraints from primary input pins.
delete_instances	Deletes the specified instance objects from the current design.
delete_iprocs	Deletes the specified list of iProcs attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by module_name.
delete_layout_core_information	In hierarchical layout-aware diagnosis, deletes all core instance data or the core instance data from a specified design.
delete_layout_verification	Reports or deletes stored layout verification information from the current layout database.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 4-1. Commands D Through F (cont.)

Command	Description
delete_lfsr_connections	Removes connections between the specified primary pins and Linear Feedback Shift Registers (LFSRs).
delete_lfsr_taps	Removes the tap positions from a Linear Feedback Shift Register (LFSR).
delete_lfsrs	Removes the specified Linear Feedback Shift Registers (LFSRs).
delete_lists	Removes pins from the pin list the tool monitors and reports on during simulation.
delete_loadboard_loopback_pairs	Deletes any loopback connection associated with the specified ports.
delete_misrs	Deletes MISRs you specify with the add_misrs command.
delete_multicycle_paths	Removes definitions of multicycle paths previously read in using the read_sdc command.
delete_nets	Removes net objects inside an instance of a design module.
delete_nofaults	Removes the nofault settings from either the specified pin or instance/module pathnames.
delete_nonscan_instances	Resets any user-specified is_non_scannable attribute on the specified objects.
delete_notest_points	Enables test logic insertion in the specified regions.
delete_output_masks	Removes the masking of the specified primary output pins.
delete_pade_logical_connection	Removes logical connections previously added to a design by the add_pade_logical_connection command. You must specify either the source pin/port or the destination pin/port or both source and destination pins/ports.
delete_pattern_trigger	Deletes pattern triggers set with the add_pattern_trigger command.
delete_patterns	Deletes the current pattern set.
delete_pins	Removes pin objects from a design module instance.
delete_ports	Removes port objects from a design module.
delete_primary_inputs	Removes the specified primary inputs from the existing flat model or, if a flat model does not yet exist, from the flattened result.
delete_primary_outputs	Removes the specified primary outputs from the existing flat model or, if a flat model does not yet exist, from the flattened result.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 4-1. Commands D Through F (cont.)

Command	Description
delete_processors	Removes processor definitions previously specified with the add_processors command.
delete_read_controls	Removes the read control line definitions from the specified primary input pins.
delete_register_value	Deletes all or a specified register value variable specified with the add_register_value command.
delete_rtl_to_gates_mapping	Removes name-mapping rules for RTL to post-synthesis/ post-layout name mapping.
delete_scan_chain_families	Removes scan chain families.
delete_scan_chains	Removes the specified scan chain definitions from the scan chain list.
delete_scan_elements	Removes scan elements (such as chain or segment) of a given name.
delete_scan_groups	Removes the specified scan chain group definitions from the scan chain group list.
delete_scan_instances	Removes the specified, sequential instances from the user-identified scan instance list.
delete_scan_modes	Removes one or more scan modes.
delete_schematic_callout	Deletes informational callouts from a Tessent Visualizer schematic window.
delete_schematic_objects	Deletes design objects from a Tessent Visualizer schematic window.
delete_schematic_path	Delete the specified path from a Tessent Visualizer schematic window.
delete_simdut_fault	Removes the injected stuck-at fault at the specified signal or all the injected faults.
delete_simulation_contexts	Deletes one or more user-defined simulation contexts.
delete_simulation_forces	Removes forces from one or more gate_pin objects.
delete_synchronous_clock_group	Removes one or more user-defined synchronous clock groups.
delete_test_points	Removes the specified test point definitions.
delete_tied_signals	Removes the assigned (tied) value from the specified floating nets or pins.
delete_wave_generator_pins	Deletes provided pin objects, or all objects, from the Tessent Visualizer Wave Generator tab.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 4-1. Commands D Through F (cont.)

Command	Description
delete_write_controls	Removes the write control line definitions from the specified primary input pins.
diagnose_failures	Diagnoses the failures in the specified failure file.
display_diagnosis_report	Opens a diagnosis report in which you can click symptoms and suspect locations to add related gates to the Tessent Visualizer Hierarchical Schematic window.
display_message	Sends a message to the specified message stream.
display_specification	Displays the DftSpecification(*design_name*,*id*) configuration data in a tab of the Config Data Browser window.
dofile	Runs the commands contained within the specified file.
echo	Issues a user-defined string to the transcript.
elaborate_icl_modules	Elaborates the specified ICL modules.
encrypt_layout	Encrypts a layout database (LDB) for purposes of data transfer.
execute_cdp_test	Runs the specified test using the currently connected tester.
execute_gpib_command	Enables you to control and program GPIB equipment.
execute_tester_command	Enables you to verify and debug the wiring connection between an adaptor used for Tessent SiliconInsight and the design under test.
exit	Terminates the application tool program.
expand_compressed_patterns	Expands compressed 1-hot patterns for better internal failing flop diagnosis.
extract_fault_sites	Extracts interconnect bridges and opens from a layout database and writes out the data in UDFM file format.
extract_icl	Checks the ICL connectivity rules between IJTAG instances, extracts the top-level ICL module, creates an SDC file that can be used for synthesis and static timing analysis (STA), and extracts an IJTAG graybox model.
extract_inter_cell_data	Extracts neighborhood cell combinations and writes the combinations out in UDFM file format.
extract_sdc	Creates an SDC file that can be used for synthesis and static timing analysis (STA).
filter_collection	Filters a previously created collection, resulting in a new collection.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 4-1. Commands D Through F (cont.)

Command	Description
find_design_names	Displays design object hierarchical names matched by an input string, which may include asterisk (*) or question mark (?) wildcard characters.
foreach_in_collection	Iterates over each element in a collection.
format_dictionary	Generates a Tcl dictionary in a human-readable format.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

deassert_icl_sib

Context: patterns -silicon_insight -ijtag

Mode: setup

Configures the SIB to pick the client input.

Usage

deassert_icl_sib ***sib_name***

Description

Use this command when you are configuring a scan path in the IJTAG network. The specified SIB is held at 0 in the scan path.

Use this command when manually debugging IJTAG networks as described in "Diagnosing and Debugging IJTAG Networks" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

Arguments

- ***sib_name***

  A required variable the specifies the name of the SIB.

Examples

deassert_icl_sib sib_l4

Related Topics

analyze_icl_suspect

assert_icl_sib

create_icl_flush_pattern

create_icl_setup_pattern

get_icl_network

get_icl_scan_mux

isolate_icl_node

report_icl_network

set_icl_network

set_icl_scan_mux

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

defer_schematic_updates

Context: all contexts

Modes: all modes

Postpones schematic updates until after the scope introduced by the command ends.

Usage

defer_schematic_updates ***script*** [-silent] [-highlight_added_objects]

Description

Defers execution of the given commands until the scope returns to that within which the defer_schematic_updates command was issued.

Arguments

- ***script***

  One or more Tcl commands enclosed in braces ({}).

- *-silent*

  An optional switch that suppresses display of the commands run.

- *-highlight_added_objects*

  An optional switch that highlights any added objects in orange.

Examples

The following example adds a design object and a design connection to the Hierarchical Schematic window. This addition occurs after the scope returns to the original scope from which you ran the defer_schematic_updates command.

```
defer_schematic_updates {
  add_schematic_objects bsr_i1 -disp hier
  add_schematic_connections cour_i/out[3] -to tap_i/in[3] -disp hier
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_atpg_constraints

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Removes state restrictions from the specified pins.

Usage

delete_atpg_constraints {***pin_pathname*** | ***gate_ID#*** | {***function_name*** | {**-Cell** ***cell_name*** {***pin_name***…}}}… [-Instance {*object_expression*…}] [-MODUle {*module_name* …}]}… | **-All**

Description

Define pin restrictions by using the add_atpg_constraints command.

If you change ATPG constraints, create patterns with those changed constraints, and then compress patterns, all patterns not meeting the new constraints are rejected. This can cause the good patterns created with the old constraints to be rejected; therefore, use the delete_atpg_constraints command to remove all ATPG constraints before compressing the pattern set.

Arguments

- ***pin_pathname***

  A repeatable string specifying the pathname of the pin from which to remove any ATPG pin constraints.

- ***gate_ID#***

  A repeatable integer specifying the gate identification number of the gate from which to remove any ATPG pin constraints.

- ***function_name***

  A repeatable string specifying the name of a function created with the add_atpg_functions command and from which to remove any ATPG pin constraints.

- **-Cell** ***cell_name*** {***pin_name***}

  A repeatable switch and string pair that removes ATPG constraints from the specified pin on the DFT library cell. You can repeat the *pin_name* argument if there are multiple pins or nets on a cell that have ATPG constraints you need to remove.

- -Instance *object_expression*

  An optional switch and repeatable string pair that removes ATPG constraints inside the specified list of instances. You can use regular expressions, which may include any number of embedded asterisk (*) and question mark (?) wildcard characters. You can only use this switch when using the -Cell option or when a function name is specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -MODUle *module_name*

  An optional switch and repeatable string pair that removes the ATPG constraints inside all instances of the specified module.

- **-All**

  A switch that removes all current, user-defined ATPG constraints from all objects.

Examples

Example 1

The following example creates two user-defined ATPG pin constraints, runs the ATPG process, removes all ATPG constraints, and then compresses the pattern set:

```
set_system_mode analysis
add_atpg_functions and_b_in and /i$144/q /i$141/q /i$142/q
add_atpg_constraints 0 /i$135/q
add_atpg_constraints 1 and_b_in
create_patterns
delete_atpg_constraints -all
compress_patterns
```

Example 2

The following Tessent FastScan example shows how to use wildcards.

```
add_atpg_constraints 1 -cell mux21 Y -ins uALU/*/ix184
report_atpg_constraints
```

```
1 dynamic /uALU/u3/ix184/Y (4400) .
1 dynamic /uALU/u2/ix184/Y (4406) .
1 dynamic /uALU/u1/ix184/Y (4412) .
```

```
delete_atpg_constraints -cell mux21 Y -ins uALU/u1/*
report_atpg_constraints
```

```
1 dynamic /uALU/u3/ix184/Y (4400) .
1 dynamic /uALU/u2/ix184/Y (4406) .
```

Related Topics

add_atpg_constraints

add_atpg_functions

report_atpg_constraints

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_atpg_functions

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Removes the specified function definitions.

Usage

delete_atpg_functions ***function_name***... | **-All**

Description

The delete_atpg_functions command lets you delete_atpg_functions defined with the add_atpg_functions command. You cannot remove an ATPG function if an ATPG constraint is currently using that function. If you attempt to remove an in-use function, the tool generates an error. Therefore, if you need to delete an in-use ATPG function, you must first remove all the associated ATPG constraints using the delete_atpg_constraints command; then you can remove the ATPG function.

You can display a list of the current ATPG functions that the tool is using as ATPG constraints by using the report_atpg_constraints command.

Arguments

- ***function_name***

  A repeatable string that specifies the names of the ATPG functions that you want to delete.

- **-All**

  A switch that removes all ATPG function definitions.

Examples

The following example creates two user-defined ATPG functions, one user-defined ATPG constraint, displays the currently-in-use ATPG constraints, and then removes one of the inactive ATPG functions:

```
add_atpg_functions and_b_in And /i$144/q /i$141/q /i$142/q
add_atpg_functions select_b_in select /i$144/q /i$142/q
add_atpg_constraints 0 /i$135/q
report_atpg_constraints
```

```
0 /$135/Q (23)
```

```
delete_atpg_functions and_b_in
```

Related Topics

add_atpg_functions

delete_atpg_constraints

report_atpg_constraints

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_atpg_functions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_bist_capture_ranges

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies that the tool no longer uses a particular named capture procedure when generating BIST patterns.

Usage

delete_bist_capture_ranges ***capture_proc_name*** | **-All**

Description

Specifies that the tool no longer uses a particular named capture procedure when generating BIST patterns.

Arguments

- ***capture_proc_name***

  A required string that specifies the name of the capture procedure.

- **-All**

  A switch that specifies that the tool no longer use any named capture procedures when generating BIST patterns.

Related Topics

add_bist_capture_range

report_bist_capture_ranges

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_black_boxes

Context: all contexts

Mode: setup

Undoes the effect of the add_black_boxes command.

Usage

delete_black_boxes **-Instances** [*ins_pathnames*] | **-Module**s [*module_names*] | **-All**

Description

Undoes the effect of the add_black_boxes command.

For a module that was originally modeled, removing the effect of the add_black_boxes command reinstates the original model. Specified tied values are no longer set on the output pins. For a module that was empty or undefined in the input netlist, the output pins revert to the default tied value of X.

Note

The tool releases the flat model if one exists at the time you issue this command.

Arguments

- **-Instances** [*ins_pathnames*]

  A switch that specifies for the tool to undo the effect of the add_black_boxes command on all instances specified by the *ins_pathnames* argument. The *ins_pathnames* argument can be a Tcl list of one or more instances, or a collection of one or more instances. If you do not specify the *ins_pathname*, the tool undoes the effect of the add_black_boxes command on all instance-based blackboxes.

- **-Modules** [*module_names*]

  A required switch that specifies for the tool to undo the effect of the add_black_boxes command on all instances of the Verilog module or Tessent Cell library model specified by the *module_names* argument. The *module_names* argument can be a Tcl list of one or more modules, or a collection of one or more modules. If you do not specify the *module_names* argument, the tool undoes the effect of the add_black_boxes command on all module-based blackboxes.

- **-All**

  A required switch that specifies for the tool to undo the effect of the add_black_boxes command on all blackboxes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example adds the black box for module “core” then undoes all blackboxes that were defined.

```
add_black_boxes -modules core 1
delete_black_boxes -all
```

Example 2

The following example deletes blackboxes for the collection of modules returned by the get_modules command. The get_modules command returns only those modules that have been marked by the user with the user-defined attribute mark_black_box.

```
delete_black_boxes -modules [get_modules -filter mark_black_box]
```

Related Topics

add_black_boxes

report_black_boxes

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_bsdl

Deletes stored BSDL file data for multiple die in a 2.5D design.

Usage

delete_bsdl {**-all** | **-entity** ***entity***} [-for_instances *for_instances*]

Description

Deletes BSDL file data from memory that was stored there by the read_bsdl command.

Arguments

- -all

  Deletes the data for all BSDL entities.

- -entity *entity*

  Deletes the data for the specified BSDL entity.

- -for_instances *for_instances*

  A switch and Tcl collection or list of instance names for which to delete the BSDL data. If it is not specified, the tool deletes all entries under the entity specified with the -entity option.

Examples

The following example deletes the BSDL data for one instance of a die in a 2.5D design.

```
report_bsdl
//
// // ------ -------- ----------------------------------------
// // Entity Instance BSDL File
// // ------ -------- ----------------------------------------
// // diee   i1       tsdb_outdir/inst/diee.inst/diee_base.bsdl
// //        i2       tsdb_outdir/inst/diee.inst/diee_iop0.bsdl
// //        i3       tsdb_outdir/inst/diee.inst/diee_iop1.bsdl
// //
delete_bsdl -entity diee -for_instances
i2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_bsdl_extraction_option

Context: dft (with no subcontext)

Mode: setup

Deletes values for various BSDL attributes previously set with the set_bsdl_extraction_options command.

Usage

```
delete_bsdl_extraction_option {-all | [-pulse_min_duration]
    [-train_minimum_count]
    [-train_max_duration]
    [-manufacturer_id_code]
    [-part_number_code]
    [-version_code]
    [-bonding_configuration_enables]}
```

Description

Deletes values for various BSDL attributes previously set with the set_bsdl_extraction_options command for the BSDL extraction.

Arguments

- -all

  An optional switch that deletes the values for minimum pulse duration, minimum train count, and maximum train duration.

- -pulse_min_duration

  An optional switch that deletes the value for minimum pulse duration. This is the minimum time to use between the rising and the next falling edge during the AC tests.

- -train_minimum_count

  An optional switch that deletes the value for the minimum number of pulses the train test must have.

- -train_max_duration

  An optional switch that deletes the value for the maximum time the train test can take.

- -manufacturer_id_code

  An optional switch that deletes the value of the manufacturer ID code section of the 32-bit device ID.

- -part_number_code

  An optional switch that deletes the value of the part number code section of the 32-bit device ID.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -version_code

  An optional switch that deletes the value of the version code section of the 32-bit device ID.
- -bonding_configuration_enables

  An optional switch that deletes the value of the bonding configuration enable pins/ports.

Related Topics

get_bsdl_extraction_option

report_bsdl_extraction_options

set_bsdl_extraction_options

Prerequisites for BSDL Extraction

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_bsdl_extraction_user_instruction

Context: dft (with no subcontext)

Mode: setup

Deletes the specified user instructions previously added with the add_bsdl_extraction_user_instruction.

Usage

delete_bsdl_extraction_user_instruction **-all** | **-name** *name*

Description

Deletes the specified user instructions previously added with the add_bsdl_extraction_user_instruction. One of **-all** or **-name** is required.

Arguments

- -all

  Deletes all added user instructions.

- -name *name*

  Deletes the specified user instruction.

Related Topics

add_bsdl_extraction_user_instruction

report_bsdl_extraction_user_instructions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_capture_procedures

Context: dft -edt, patterns -scan

Mode: analysis

Removes the specified capture procedures from internal memory.

Usage

delete_capture_procedures {**-All** | **-Created_capture_procedures** | **-Loaded_capture_procedures** | **-Disabled** | ***procedure_name***...}

Description

The delete_capture_procedures command deletes all or specific named capture procedures from the internal tool memory. A named capture procedure used by the current pattern set cannot be deleted.

Arguments

- **-All**

  A switch that deletes all the named capture procedures.

- **-Created_capture_procedures**

  A switch that deletes all the named capture procedures automatically created by the tool.

- **-LOaded_capture_procedures**

  A switch that deletes all the named capture procedures loaded using the read_procfile command or from procedure files associated with the scan group definition.

- **-DIsabled**

  A switch that deletes all the disabled named capture procedures.

- ***procedure_name***

  A repeatable string that specifies a named capture procedure to delete.

Examples

Example 1

The following example deletes all capture procedures.

```
delete_capture_procedures -All
```

Example 2

The following example removes the capture procedure named user_capture_proc1.

```
delete_capture_procedures user_capture_proc1
```

Related Topics

create_capture_procedures

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_capture_procedures

set_capture_procedures

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_cdp_test

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Removes the specified test from the CDP.

Usage

delete_cdp_test ***test_name***

Arguments

- ***test_name***

  A required string that specifies the name of a CDP test.

Examples

The following example removes the test1 CDP test:

```
delete_cdp_test test1
```

Related Topics

add_cdp_test

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_cell_constraints

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Removes constraints placed on scan cells.

Usage

delete_cell_constraints **-All** | ***pin_pathname*** | {***chain_name cell_position***} | {**-Drc C6**} | {**-CHain** ***chain_name***} | {**-CLock** ***clock_name***}

Description

The delete_cell_constraints command deletes the constraints placed on scan cells using the add_cell_constraints command. You can specify a scan cell by using either a pin pathname or a position in a scan chain.

Arguments

- **-All**

  A switch that, in Setup mode, specifies to delete all constraints from all scan cells. In analysis mode, it specifies to delete all cell constraints that were added in non-Setup modes.

- ***pin_pathname***

  A string that specifies the name of an output pin that directly connects to a scan memory element (you can only specify output pins of buffers and inverters). Valid in Setup mode only.

- ***chain_name cell_position***

  A string and integer pair that specifies the name of a currently-defined scan chain and the position of the cell in the scan chain. The *cell_position* is an integer where 0 is the scan cell closest to the scan-out pin. Valid in Setup mode only.

- **-Drc C6**

  A switch and literal pair that specifies to delete all cell constraints added by “add_cell_constraints -drc c6”. Valid in analysis mode only.

- **-CHain** ***chain_name***

  A switch and string pair that specifies to delete all cell constraints added by the -Chain argument to the add_cell_constraints command. Valid in analysis mode only.

- **-CLock** ***clock_name***

  A switch and string pair that specifies to delete all cell constraints added by the -Clock argument to the add_cell_constraints command. Valid in analysis mode only.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example deletes an incorrectly added cell constraint placed on a scan cell:

```
add_clocks 1 clock1
add_scan_groups group1 proc.g1
add_scan_chains chain1 group1 scanin1 scanout1
add_scan_chains chain2 group1 scanin2 scanout2
add_cell_constraints chain1 5 c0
add_cell_constraints chain2 3 c1
delete_cell_constraints chain2 3
add_cell_constraints chain2 4 c1
report_cell_constraints
```

Related Topics

add_cell_constraints

report_cell_constraints

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_cell_library

Context: unspecified, all contexts

Mode: setup

Removes all current cell libraries, the current design including the flat model, and all tool data that is design-specific.

Usage

delete_cell_library

Description

This command also clears any Tessent Visualizer schematic or instance browser tabs.

When this command is deleting a flat model, and that flat model was saved with a list of enabled and disabled commands, then the restriction on the disabled commands is removed. In other words, all commands permitted in the tool are enabled for the next design.

Arguments

None

Examples

The following example removes all current cell libraries:

```
delete_cell_library
```

Related Topics

delete_design

read_cell_library

read_flat_model

read_verilog

set_current_design

set_design_macros

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_cell_models

Context: dft -scan, dft -test_points

Mode: setup, analysis

Specifies the name of the DFT library cell that the tool is to remove from the active list of cells that the user can access when adding test points or that the tool can access when inserting test logic.

Usage

delete_cell_models ***dftlib_model***… | {**-Type** {**INV** | **And** | **Buf** | **OR** | **NAnd** | **NOr** | **Xor** | **Mux** | **Scancell** | **DFf** | **LATCH**}} | **-All**

Description

Specifies the name of the DFT library cell that the tool is to remove from the active list of cells that the user can access when adding test points or that the tool can access when inserting test logic.

You originally added the cells to the active list with either the add_cell_models command or with the *cell_type* library attribute. If you remove a cell model from the active list, you only remove the cell from that list and do not change the DFT library.

If you accidentally delete a DFT library cell from the active list with the delete_cell_models command, you can add the specified cell back into the active list with the add_cell_models command.

Arguments

- ***dftlib_model***

  A repeatable string that specifies the names of the particular DFT library models that you want the tool to remove from the active list.

- **-Type INV** | **And** | **Buf** | **OR** | **NAnd** | **NOr** | **Xor** | **Mux** | **Scancell** | **DFf** | **LATCH**

  A switch and argument pair that specifies the cell model type of all DFT library models that you want the tool to remove from the active list. The valid cell model types are as follows:

  INV — A literal that specifies a one-input inverter gate.

  And — A literal that specifies an AND gate.

  Buf — A literal that specifies a one-input buffer gate.

  OR — A literal that specifies an OR gate.

  NAnd — A literal that specifies a NAND gate.

  NOr — A literal that specifies a NOR gate.

  Xor — A literal that specifies an exclusive OR gate.

  Mux — A literal that specifies a 2-1 multiplexer.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Scancell — A literal that specifies a mux-scan D flip-flop.

DFf — A literal that specifies a D flip-flop.

LATCH — A literal that specifies a D latch.

- **-All**

  A switch that removes all cell models from the active list, including those tagged in the DFT library with the *cell_type* attribute. This switch does not change the contents of the DFT library, only the active list within the tool.

Examples

The following example removes a DFT library model from the active list:

```
add_clocks 0 clk
set_test_logic -set on -reset on
set_system_mode analysis
add_cell_models and2 -type and
add_cell_models or2 -type or
add_cell_models mux21h -type mux s a b
add_cell_models nor2 -type nor
delete_cell_models or2
insert_test_logic
```

Related Topics

add_cell_models

report_cell_models

set_test_logic

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_chain_masks

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup and analysis when EDT is off

Mode: setup only when EDT or LBIST is on

Removes chain masks from the specified scan chains.

Usage

delete_chain_masks ***chain_name***... | **-INStance *instance_name* -Block_chain_index_list *block_chain_index_list*** | **-All**

Description

The delete_chain_masks command deletes chain masks from the specified scan chains defined with the add_chain_masks command. You can delete the definitions of specific chain name or all chains.

Arguments

- ***chain_name***

  A repeatable string that specifies the names of the chain mask definitions that you want to delete.

- **-INStance *instance_name* -Block_chain_index_list *block_chain_index_list***...

  Required set of switches and values that specifies a core instance object_spec and the list of chain indices starting with “1” on the EDT controller.

- **-All**

  A switch that deletes all chain mask definitions.

Examples

The following example defines two chain masks then deletes all chain masks:

```
add_chain_masks chain2 chain3
delete_chain_masks -all
```

Related Topics

add_chain_masks

report_chain_masks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_clocks

Context: all contexts

Mode: setup

Removes clocks from the clock list.

Usage

delete_clocks ***pin_or_label_name***... | **-All**

Description

The delete_clocks command removes clocks from the clock list that you can specify by pin name or label name. If you delete an equivalence pin, the command also deletes all of the equivalent pins from the clock list.

If you want to change one or more options of a defined clock, you can use the set_clock_options command to modify the clock.

Arguments

- ***pin_or_label_name***

  A repeatable string that specifies the pin name or label name to delete from the clock list.

- **-All**

  A switch that deletes all clocks from the clock list.

Examples

The following example deletes an incorrectly added clock from the clock list:

```
add_clocks 1 clock1
add_clocks 1 clock2
delete_clocks clock1
```

Related Topics

add_clocks

report_clocks

set_clock_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_config_element

Context: unspecified, all contexts

Mode: all modes

Deletes one or more configuration elements. Only elements that can be added can be deleted.

Usage

delete_config_element ***config_object*** | {***config_name_list*** [-in_wrapper *wrapper_object_spec*]}

Description

Singular elements such as a wrapper declared with Repeatable set to single in the metadata and properties cannot be deleted as they always exist as soon as their parent wrapper exist.

Arguments

- ***config_object***

  A required collection containing one or more configuration elements. When using a collection of object, the -in_wrapper option cannot be used as the objects contains the full path of the object

- ***config_name_list***

  A required value that specifies a Tcl list containing names of one or more configuration elements. When the -in_wrapper option is specified, the names are relative to the specified wrapper element.

- -in_wrapper *wrapper_object_spec*

  An optional value pair that is used to specify the parent wrapper in which the configuration elements are to be deleted. The *wrapper_object_spec* value is the name of a wrapper or a collection containing a wrapper element. The names in the ***config_name_list*** are relative to the specified *wrapper_object_spec*.

Examples

The following example deletes elements using the full name of elements, using a relative name with respect to a parent wrapper and using a collection of objects.

```
delete_config_element {
   DftSpecification(mod1,rtl)/IjtagNetwork/HostScanInterface(1)
   DftSpecification(mod1,rtl)/IjtagNetwork/HostScanInterface(2) }

delete_config_element {HostScanInterface(1) HostScanInterface(2)} \
   -in_wrapper DftSpecification(mod1,rtl)/IjtagNetwork

> delete_config_element [get_config_elements HostScanInterface \
                    -in DftSpecification(mod1,rtl)/IjtagNetwork
```

Related Topics

add_config_element

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_config_elements

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_config_messages

Context: all contexts

Mode: all modes

Deletes error, warning or info message that were added to configuration elements using the add_config_message command.

Usage

delete_config_messages ***config_object_spec***
[-hierarchical] [-all | [-error] [-warning] [-info]] [-silent]

Description

Deletes error, warning or info message that were added to configuration elements using the add_config_message command.

Arguments

- ***config_object_spec***

  A required value that specifies a configuration object on which to delete the messages. The config_object_spec value is the name of a configuration element or a collection containing a single configuration element object.

- -hierarchical

  An optional Boolean option that specifies that the config message should be deleted on the specified config_object_spec and all elements below it. The -hierarchical option is ignored when the config_object_spec is a property.

- -all

  An optional Boolean option that specifies that all message types are to be deleted. The use of the -all option is mutually exclusive to the specification of the -error, -warning, and -info options. When none of the four options is specified, -all is assumed.

- -error

  An optional Boolean option that specifies that the messages to be deleted are of type error. The use of the -error option is mutually exclusive to the specification of the -all, -warning, and -info options. When none of the -error, -warning and -info options is specified, -all is assumed.

- -warning

  An optional Boolean option that specifies that the messages to be deleted are of type warning. The use of the -error option is mutually exclusive to the specification of the -error, -warning, and -info options. When none of the -error, -warning and -info options is specified, -all is assumed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -info

  An optional Boolean option that specifies that the messages to be deleted are of type info. The use of the -error option is mutually exclusive to the specification of the -all option. When none of the -error, -warning and -info options is specified, -all is assumed.

- -silent

  An optional Boolean option that disables the warning that is normally generated if the specified configuration element does not contain any messages of the specified type.

Examples

The following example deletes all message on and below the wrapper DftSpecification(moda,rtl). The addition and deletion of messages on the DftSpecification and DefaultsSpecification/DftSpecification wrappers is automatically handled by the process_dft_specification command.

```
delete_config_messages DftSpecification(moda,rtl) -hierarchical
```

Related Topics

add_config_message

get_config_messages

report_config_messages

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_config_tabs

Context: all contexts

Mode: all modes

Deletes one or more configuration tree tabs from the Config Data Browser window.

Usage

delete_config_tabs [[-partition *partition*] *wrapper*] [-silent]

Description

Deletes configuration tree tabs from the Config Data Browser window that were previously added using the add_config_tab or display_specification command.

When no config_object_spec value is specified, all existing tabs are deleted.

Arguments

- -partition *partition wrapper*

  An optional switch and value pair that specifies the partition and wrapper that the tool removes from the Config Data Browser. When this option is not specified, the tool uses the default partition.

- -silent

  An optional value that disables the warning that is normally generated when *config_object_spec* contains a configuration element that is not a root wrapper displayed in an existing tab, or when no configuration tab exists at all.

Examples

The first example deletes the tab in which the DftSpecification(mod1.rtl) wrapper is displayed. The second example deletes all tabs and suppresses the warning that is generated if there are no tabs to delete.

```
delete_config_tabs -partition tcd /Core(RAM12x8)
delete_config_tabs -silent
```

Related Topics

add_config_tab

display_specification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_connections

Context: all contexts

Mode: insertion

Disconnects net, pin, or port objects.

Usage

delete_connections ***obj_spec*** [-within] [-silent]

Description

If the disconnected object is a pin, the pin is left without a connection in the instance independent of whether the pin is an input, output, or inout pin. If the object is a port, if the associated net of the same name is connected to any other object, the tool creates a uniquified net and uses it in all of the connections the original net made. This results in input and inout ports with no fanout, and output and inout ports with no fan-in. If the object is a net that is also a pin, the tool creates a uniquified net and uses it in all of the connections the original net made. This results in input and inout pins with no fanout, and output and inout ports with no fan-in.

You must be careful when using the delete_connections command inside a non-uniquified design. You must only do the move or disconnection once per module and once per generate loop count. To perform this robustly, see how the parent_instance and the leaf_name_hash attributes are used in Section 3 of "Example 5" on page 982 in the get_dft_cell command description.

Tip

If you use the delete_connections command on instances generated by a "generate loop" block prior to uniquification, it applies to all instances. Uniquify instances in the generate loop to use the delete_connections command on specific instances.

Arguments

- ***obj_spec***

  The obj_spec must be a Tcl list of one or more net, pin, or port names, or a collection of one or more net, pin or port names created by the get_nets, get_pins, or get_ports commands.

  If an object within obj_spec does not exist and the -silent option is specified, the object is ignored. If an object within obj_spec does not exist and the -silent option is not specified, the tool generates an error.

- -within

  An optional argument that specifies to disconnect a pin inside the boundary of a hierarchical instance. That is, the tool disconnects the pin from the net to which it is attached inside the instance. If you do not use this argument, the tool disconnects the pin on the outside of the instance. This argument requires that obj_spec is a pin on a hierarchical instance.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -silent

  An optional argument that specifies to ignore specified objects that do not exist and to not generate error messages.

Examples

Example 1

The following example disconnects the net or constant that may be connected to input pin u1/A.

delete_connections u1/A

```
{}
```

Example 2

The following example disconnects the net or constant that may be connected to the pin within the generated collection.

delete_connections [get_pins ABC*/TE]

```
{}
```

Example 3

The following example disconnects the internal fanout of the input pin ABC/in1. It does that by renaming the net ABC/in1 to ABC/in1_ts1. The pin ABC/in1 ends up with no fanout.

delete_connections [get_nets ABC/in1]

```
{ABC/in1_ts1}
```

Example 4

The following example disconnects the internal fan-in of the output port out1. It does that by renaming the net out1 to out1_ts1. The port out1 ends up with no fan-in.

delete_connections out1

```
{out_ts1}
```

Related Topics

create_connections

get_icl_pins

get_icl_ports

get_nets

move_connections

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_core_descriptions

Context: all contexts

Mode: setup

Removes core descriptions that have been read in using the read_core_descriptions command from memory.

Usage

delete_core_descriptions {**-all** | **-cores** ***core_name_list***} [-modes *mode_name_list*]

Description

Removes core descriptions that have been read in using the read_core_descriptions command from memory.

You cannot delete a core description that is instantiated in another core description unless you specify to delete the parent core description also, by using the -all switch for example.

If a corresponding core instance exists for the core description, this command deletes the core instance along with the core description.

If you specify to delete multiple descriptions, and deleting one or more core descriptions fails, the command continues deleting the remaining core descriptions.

You can delete a top-level core description only in a flow without a top-level netlist. For information about such a flow, see "Retargeting Patterns Without a Top-Level Netlist" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- **-all**

  A switch that specifies to delete all core description files in the current design.

- **-cores** ***core_name_list***

  A required switch and Tcl list that specifies the core names of the core descriptions to be deleted.

- -modes *mode_name_list*

  An optional switch that specifies the mode of the core to delete.

Examples

The following example reads in the core descriptions from the top_core_des file and then reports the defined core descriptions. The top-level core description is then deleted from memory. The second_level_core becomes the top of the design. The report_core_descriptions command then reports the defined core descriptions after the top-level core description is deleted.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_context patterns -scan_retargeting
read_core_descriptions top_core_des
report_core_descriptions

```
Name                  Mode Name    Mode Type
-------------------   -----------  -----------
* top                  retargeting  retargeting
  second_level_core    retargeting  retargeting
  core1                mode_M1      internal
                       mode_M3      internal
  core2                mode_M2      internal
  core3                mode_M2      internal
                       mode_M3      internal
```

delete_core_descriptions -core top
set_current_design second_level_core
report_core_descriptions

```
Name                  Mode Name    Mode Type
-------------------   -----------  -----------
* second_level_core   retargeting  retargeting
  core1                mode_M1      internal
                       mode_M3      internal
  core2                mode_M2      internal
  core3                mode_M2      internal
                       mode_M3      internal
```

Related Topics

add_core_instances

delete_core_instances

read_core_descriptions

report_core_instances

report_core_descriptions

write_core_description

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_core_instances

Context: patterns -scan, patterns -scan_retargeting

Mode: setup

Removes an instance binding created with the add_core_instances command.

Usage

delete_core_instances
{**-all** | **-cores** ***core_names*** | **-modules** ***module_objects*** | **-instances** ***instance_objects*** | **-current_design**}
[-modes *mode_names*] [-silent]

Description

Removes an instance binding created with the add_core_instances command.

You can use this command to retarget patterns for different configurations sequentially in the same tool session. After completing one configuration and before switching to the next configuration, you use the delete_core_instances command to delete the old core instances so that the core instances used in the next session can be added.

Arguments

- **-all**

  A required switch that specifies to remove all bindings that have been added with the add_core_instances command.

- **-cores** ***core_names***

  A required switch and Tcl list that specifies the core names of the core descriptions from which all instance bindings should be removed.

- **-modules** ***module_objects***

  A required switch and string pair that specifies a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects. This command removes the instance bindings for each instantiation of those modules in the design.

- **-instances** ***instance_objects***

  A required switch and string pair that specifies a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects. This command removes the instance bindings for each instance in the list.

- **-current_design**

  A required switch that specifies to delete the core instance with respect to the current design. This switch is mutually exclusive with the -core, -modules, and -instances switches.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -modes *mode_names*

  An optional switch and string pair that specifies the list of modes to delete for each specified core instance.

- -silent

  An optional switch that specifies to suppress warning messages.

Examples

Example 1

The following example adds the same instance twice. The second attempt fails with a warning, deletes it, and re-adds it successfully.

add_core_instances -instances core_1 -cores my_core

```
//  Added core instance 'core_1'.
```

add_core_instances -instances core_1 -cores my_core

```
//  Warning: Instance 'core_1' has already been added. Skipping.
```

delete_core_instances -instances core_1
add_core_instances -instances core_1 -cores my_core

```
//  Added core instance 'core_1'.
```

Example 2

The following example adds two instances of the same module and then deletes both of them.

add_core_instances -instances {core_1 core_2} -cores my_core

```
//  Added core instance 'core_1'.
//  Added core instance 'core_2'.
```

delete_core_instances -instances {core_1 core_2}

Example 3

The following example adds two instances and then attempts to delete the two instances as well as one other instance that has not been added. The -silent usage suppresses the warning messages that would be printed for the three non-existing core instances.

add_core_instances -instances {core_1 core_2} -cores my_core
delete_core_instances -instances {core_1 core_2 core_3}

```
//  Warning: Instance 'core_3' of module 'my_core_2' has not been added as
//  a core instance. Skipping.
```

delete_core_instances -instances {core_1 core_2 core_3} -silent

Example 4

The following example adds two instances of the same module and then deletes both of them through the module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_core_instances -instances {core_1 core_2} -cores my_core

```
//  Added core instance 'core_1'.
//  Added core instance 'core_2'.
```

delete_core_instances -modules my_module

Related Topics

add_core_instances

read_core_descriptions

report_core_instances

write_core_description

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_design

Context: all contexts

Mode: setup

Removes all design modules including the flat model and all tool data that is design-specific.

Usage

delete_design

Description

Cell modules read in with the read_cell_library command are preserved. However, cell modules that are Verilog modules surrounded by `celldefine and loaded with the read_verilog command are also deleted.

This command also clears any Tessent Visualizer schematic or instance browser tabs. Note that this command does not delete cell libraries unless a flat model is deleted.

When the delete_design command is deleting a flat model, and that flat model was saved with a list of enabled and disabled commands, then the restriction on the disabled commands is removed. In other words, all commands permitted in the tool are enabled for the next design.

Arguments

None

Examples

The following example removes all design modules:

```
delete_design
```

Related Topics

delete_cell_library

read_verilog

set_current_design

set_design_macros

read_flat_model

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_design_partitions

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Deletes partition definitions from a design.

Usage

delete_design_partitions **-all** | ***partition_name...***

Description

Deletes a partition definition created by the add_design_partition command. Deleting the definition does not delete logic from the design.

Arguments

- **-all**

  A required switch that removes all partition definitions.

- ***partition_name***

  A required repeatable string that specifies the name of partition definitions to delete.

Examples

This example shows how to delete a partition named hot_spot_partition:

```
delete_design_partitions hot_spot_partition
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_dfm

Context: patterns -scan_diagnosis

Mode: analysis

Deletes a DFM rule from the open LDB.

Usage

delete_dfm [-rule *rule_name*] | [-vacuum]

Description

Deletes a DFM rule from the open LDB.

The -rule and -vacuum options are mutually exclusive.

Arguments

- -rule *rule_name*

  An optional switch that specifies a specific DFM rule to delete from the LDB. The rule name is one of the rule names as reported by the report_dfm_rules command. Enclose the rule name is quote marks.

  When not specified, the tool deletes all the DFM rules from the open LDB.

- -vacuum

  An optional switch that specifies to vacuum the LDB to reduce its size on disk. The tool re-creates the LDB, moving records around to squeeze out any holes left after deleting DFM rules. For large databases, this operation may take a long time.

Examples

Example 1

The following example shows the DFM rules before and after deleting one rule from the LDB.

report_dfm_rules

delete_dfm -rule "metal_cross_edge_route1"

```
-----------------------------------------------------------
  id   rule                                layer     type
-----------------------------------------------------------
  1    minimum_space_4_line_route1         route_1   BRIDGE
  2    metal_cross_edge_route1             route_1   OPEN
  3    low_density_min_space_route2        route_2   BRIDGE
  4    half_comb_route3                    route_2   OPEN
  5    metal_cross_edge_route1             route_3   OPEN
  6    corner_to_interior_metal4           route_4   BRIDGE
  7    divergent_line_with_jog_metal5      route_5   BRIDGE
  8    metal_cross_edge_route1             route_5   OPEN
  9    route_5_barbell                     route_5   OPEN
  10   route_2_between_vias                route_2   OPEN
-----------------------------------------------------------
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_dfm_rules

```
-----------------------------------------------------------
  id   rule                                layer     type
-----------------------------------------------------------
  1    minimum_space_4_line_route1         route_1   BRIDGE
  3    low_density_min_space_route2        route_2   BRIDGE
  4    half_comb_route3                    route_2   OPEN
  6    corner_to_interior_metal4           route_4   BRIDGE
  7    divergent_line_with_jog_metal5      route_5   BRIDGE
  9    route_5_barbell                     route_5   OPEN
  10   route_2_between_vias                route_2   OPEN
-----------------------------------------------------------
```

Example 2

The following example deletes multiple DFM rules. It shows you three ways you can specify deleting rules.

delete_dfm -rule "*rule_name1 rule_name2*"

or:

delete_dfm -rule "*rule_name1*" "*rule_name2*"

or:

delete_dfm -rule "*rule_name1*"

delete_dfm -rule "*rule_name2*"

Related Topics

import_dfm

report_dfm_rules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_dft_clock_enables

Context: dft (with no sub context)

Mode: setup

Prerequisites: The current design must be set with the set_current_design command.

Deletes one or all previously added DFT clock enable signals.

Usage

delete_dft_clock_enables ***port_pin_spec*** | **-all**

Description

All DFT clock enable signals are automatically deleted when elaborating a new design or re-elaborating the same design using the set_current_design command.

Arguments

- ***port_pin_spec***

  A required string that specifies the name of a port or pin on the current design, or a collection containing a port or pin on the current design, or a port on a sub-module. The pin or ports must have previously been the target of an add_dft_clock_enables command.

- **-all**

  A required Boolean switch that specifies that all pins and ports that were previously the target of an add_dft_clock_enables command are to be removed from the DFT clock enable signal list.

Examples

The following example adds three DFT clock enable signals, reports them, deletes one and reports them again, deletes them all and finishes with a last report. Notice that the clock enable signal defined on the port “FE” of the cell “cgand” remains even after deleting them all because this one was not defined with the add_dft_clock_enables command but instead was defined in the Tessent cell library.

add_dft_clock_enables en
add_dft_clock_enables blockb_i1/en2 -active_polarity 0
report_dft_clock_enables

```
// Dft clock enables
// =================
// --------------------  --------
//         Node          polarity
// --------------------  --------
// Port 'FE' of 'cgand'      1
// Port 'en'                 1
// Pin  'blockb_i1/en2'      0
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_dft_clock_enables en
report_dft_clock_enables

```
// Dft clock enables
// =================
// --------------------  --------
//         Node          polarity
// --------------------  --------
// Port 'FE' of 'cgand'      1
// Pin  'blockb_i1/en2'      0
//
```

delete_dft_clock_enables -all
report_dft_clock_enables

```
// Dft clock enables
// =================
// --------------------  --------
//         Node          polarity
// --------------------  --------
// Port 'FE' of 'cgand'      1
```

Related Topics

add_dft_clock_enables

read_cell_library

report_dft_clock_enables

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_dft_clock_muxes

Context: dft (with no sub context)

Mode: setup

Prerequisites: The current design must be set with the set_current_design command.

Deletes one or all previously added DFT clock multiplexers.

Usage

delete_dft_clock_muxes ***pin_port_spec*** | **-all**

Description

All DFT clock mux specifications are automatically deleted when elaborating a new design or re-elaborating the same design using the set_current_design command.

Arguments

- ***port_pin_spec***

  A required string that specifies the name of a port or pin on the current design, or a collection containing a port or pin on the current design. The pin or port must have previously been the target of an add_dft_clock_mux command.

- **-all**

  A required Boolean switch that specifies that all pins and ports that were previously the target of an add_dft_clock_mux command are to be removed from the DFT clock mux list.

Examples

The following example adds three DFT clock multiplexers, reports them, deletes one and reports them, deletes them all and then reports them again.

```
add_dft_clock_muxes clkb -test_clock_source mem1/clk -dft_signal_source_name all_test
register_static_dft_signal_names pll_bypass
add_dft_clock_muxes pll1/vco -test_clock_source pll1/ref \
   -dft_signal_source_name pll_bypass
add_dft_clock_muxes pll1/ref -test_clock_source clkb -dft_signal_source_name alt_ref
report_dft_clock_muxes
```

```
// Dft clock muxes
// ===============
// ----------  -----------------  -------------------
//    Node     Test clock source  Control source name
// ----------  -----------------  -------------------
// 'clkb'      mem1/clk           all_test
// 'pll1/vco'  pll1/ref           pll_bypass
// 'pll1/ref'  clkb               alt_ref
//
```

```
delete_dft_clock_muxes pll1/vco
report_dft_clock_muxes
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Dft clock muxes
// ===============
// ----------  -----------------  -------------------
//    Node     Test clock source  Control source name
// ----------  -----------------  -------------------
// 'clkb'      mem1/clk           all_test
// 'pll1/ref'  clkb               alt_ref
//
```

delete_dft_clock_muxes -all
report_dft_clock_muxes
// There are no defined dft clock muxes.

Related Topics

add_dft_clock_mux

report_dft_clock_muxes

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_dft_control_points

Context: dft (with no sub context)

Mode: setup

Prerequisites: The current design must be set with the set_current_design command.

Deletes one or all previously added DFT control points.

Usage

delete_dft_control_points ***pin_port_spec*** | **-all**

Description

All DFT control point specifications are automatically deleted when elaborating a new design or re-elaborating the same design using the set_current_design command.

Arguments

- ***port_pin_spec***

  A required string that specifies the name of a port or a pin on the current design or a collection containing a port or a pin on the current design. The pin or port must have previously been the target of an add_dft_control_points command.

- **-all**

  An optional Boolean switch that specifies that all pins and ports that were previously the target of an add_dft_control_points command are to be removed from the DFT control point list.

Examples

The following example adds three DFT control points, reports them, deletes one and reports them, and then deletes them all and reports them again.

add_dft_control_points portb -dft_signal_source_name all_test

register_static_dft_signal_names pll_bypass

add_dft_control_points pll1_mux/s -dft_signal_source_name pll_bypass

add_dft_control_points ref_mux/s -dft_signal_source_name alt_ref

report_dft_clock_mux

```
// Dft clock muxes
// ===============
// ------------ -------------------
//    Node      Control source name
// ------------ -------------------
// 'portb'      all_test
// 'pll1_mux/s' pll_bypass
// 'ref_mux/s'  alt_ref
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_dft_control_points pll1_mux/s

report_dft_control_points

```
// Dft clock muxes
// ===============
// ------------ -------------------
//    Node      Control source name
// ------------ -------------------
// 'portb'      all_test
// 'ref_mux/s'  alt_ref
//
```

delete_dft_control_points -all

report_dft_control_points

// There are no defined dft control points.

Related Topics

add_dft_control_points

report_dft_control_points

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_dft_modal_connections

Context: dft (with no sub context)

Mode: setup

Prerequisites: The current design must be set with the set_current_design command.

You use the delete_dft_modal_connections command to delete modal connection specifications previously added by the add_dft_modal_connections command.

Usage

delete_dft_modal_connections
{**-ports** ***port_spec*** | **-auxiliary_data_pins** ***pin_spec*** | **-all**}

Description

The delete_dft_modal_connections command is used to delete modal connection specifications previously added by the add_dft_modal_connections command. You can either delete all previously defined DFT modal connections using the -all switch or delete all connections to or from a set of ports or auxiliary data pins using the -ports or -auxiliary_data_pins switch. Use the report_dft_modal_connections command to see the list of ports or auxiliary data pins that can be referenced.

Arguments

- **-ports** ***port_spec***

  A switch used to specify a list of one or more port names or a collection containing one or more port objects. The ports must have been previously referenced using the "add_dft_modal_connections -ports" command.

- **-auxiliary_data_pins** ***pin_spec***

  A switch used to specify a list of one or more pin names or a collection containing one or more pin objects. The pins must have been previously referenced using the "add_dft_modal_connections -auxiliary_data_pins" command. An auxiliary data pin automatically inferred using the "add_dft_modal_connections -auxiliary_data_pins" command can also be referenced. The "Modes:" section of the report created with the report_dft_modal_connections command is useful to see all the port and auxiliary pins scheduled to be part of a modal connection.

- **-all**

  A Boolean switch used to specify that all previously specified DFT modal connections are to be deleted.

Examples

The following example shows the result of the report_dft_modal_connections before and after a delete_dft_modal_connections command invocation. The starting configuration comes from Example 1 of the add_dft_modal_connections command description.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SETUP> report_dft_modal_connections

```
DFT Modal Connections
=====================

Modes
-----
edt_mode          : DFT signal with usage scan_mode(unwrapped)
  Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
  Output connection (1):
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
retargeting1_mode : DFT signal with usage scan_mode(retargeting)
  Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
  Output connection (1):
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'

retargeting2_mode : DFT signal with usage scan_mode(retargeting)
  Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
  Output connection (1):
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
retargeting3_mode : DFT signal with usage scan_mode(retargeting)
  Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
  Output connections (2):
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
    Port 'gpio[3]' through auxiliary output pin 'DEF_inst/gpio_3_AuxOut'

Input connections
-----------------
1) Port 'gpio[1]' through auxiliary input pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_1_AuxIn'
   edt_mode          :<Existing destination>
   retargeting1_mode :    corea_i1/edt_channels_in[1]
   retargeting2_mode :    corea_i2/edt_channels_in[1]
   retargeting3_mode :    corea_i1/edt_channels_in[1]
                     :    corea_i2/edt_channels_in[1]
2) Port 'gpio[0]' through auxiliary input pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_0_AuxIn'
   edt_mode          :    <Existing destination>
   retargeting1_mode :    corea_i1/edt_channels_in[0]
   retargeting2_mode :    corea_i2/edt_channels_in[0]
   retargeting3_mode :    corea_i1/edt_channels_in[0]
                     :    corea_i2/edt_channels_in[0]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Output connections
------------------
1) Port 'gpio[2]' through auxiliary output pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_2_AuxOut'
   edt_mode          :    <Existing source>
   retargeting1_mode :    corea_i1/edt_channels_out[0]
   retargeting2_mode :    corea_i2/edt_channels_out[0]
   retargeting3_mode :    corea_i1/edt_channels_out[0]

2) Port 'gpio[3]' through auxiliary output pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_3_AuxOut'
   retargeting3_mode :    corea_i2/edt_channels_out[0]
```

SETUP> delete_dft_modal_connections -ports gpio[3:2]
SETUP> report_dft_modal_connections

```
DFT Modal Connections
=====================

Modes
-----
edt_mode          : DFT signal with usage scan_mode(unwrapped)
  Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
retargeting1_mode : DFT signal with usage scan_mode(retargeting)
  Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'

retargeting2_mode : DFT signal with usage scan_mode(retargeting)
  Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
retargeting3_mode : DFT signal with usage scan_mode(retargeting)
  Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'


Input connections
-----------------
1) Port 'gpio[1]' through auxiliary input pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_1_AuxIn'
   edt_mode          :<Existing destination>
   retargeting1_mode :    corea_i1/edt_channels_in[1]
   retargeting2_mode :    corea_i2/edt_channels_in[1]
   retargeting3_mode :    corea_i1/edt_channels_in[1]
                     :    corea_i2/edt_channels_in[1]
2) Port 'gpio[0]' through auxiliary input pin
'top_rtl1_tessent_bscan_logical_group_DEF_inst/gpio_0_AuxIn'
   edt_mode          :    <Existing destination>
   retargeting1_mode :    corea_i1/edt_channels_in[0]
   retargeting2_mode :    corea_i2/edt_channels_in[0]
   retargeting3_mode :    corea_i1/edt_channels_in[0]
                     :    corea_i2/edt_channels_in[0]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_dft_signals

Context: dft

Mode: setup

Deletes a DFT signal that was previously added using the add_dft_signals command.

Usage

delete_dft_signals {***name*** ...} | **-all_user_added** [-silent]

Description

A command used to delete a DFT signal that was previously added using the add_dft_signals command. Only DFT signals that were added and not yet created can be deleted. Once a DFT signal is inserted using the process_dft_specification command, the signal exists and can no longer be deleted.

Arguments

- ***name***

  A repeatable string defining one or more DFT signal names to delete. You can specify several names in one command line invocation. You can also supply several names in a single well formatted Tcl list.

- **-all_user_added** [-silent]

  A Boolean switch that specifies the deletion of all DFT signals that were previously added using the add_dft_signals command.

  The -silent optional argument specifies not to generate error and status messages when DFT signals cannot be found, or the deletion is not possible.

Examples

The following example shows how to delete one DFT signal at a time or all of them at one time:

add_dft_signals ltest_en int_ltest_en ext_ltest_en

get_dft_signals -list

```
ltest_en int_ltest_en ext_ltest_en
```

delete_dft_signals ltest_en

get_dft_signals -list

```
int_ltest_en ext_ltest_en
```

delete_dft_signals -all_user_added

get_dft_signals -list

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

add_dft_signals

get_dft_signal

process_dft_specification

report_dft_signals

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_edt_blocks

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Removes the data for the specified EDT block(s) from the tool's internal database.

Usage

delete_edt_blocks **-All** | ***block_name***

Description

The delete_edt_blocks command removes all data for the specified EDT block(s) from the tool's internal database, making the block(s) unavailable for use with the set_current_edt_block command.

If you delete the current EDT block, leaving just one remaining EDT block defined, the tool automatically sets the EDT context to the remaining block. The tool does not reset the EDT context, however, if there is more than one defined EDT block remaining; in this case, subsequent context-sensitive commands that define attributes for a given EDT block return an error until you set the EDT context using the add_edt_blocks or set_current_edt_block command.

You set or change the EDT context when you restrict the applicability of certain commands to a particular EDT block by issuing the add_edt_blocks or set_current_edt_block command. The EDT block on which context-sensitive commands currently operate is referred to as the current EDT block. The current EDT block remains the sole target of the context-sensitive commands until you delete its definition using the delete_edt_blocks command or change the context to a different EDT block using the add_edt_blocks or set_current_edt_block command. The add_scan_chains command, EDT-specific commands that define attributes for a given block, and some reporting commands are presently the only context-sensitive commands. Most commands like set_system_mode, report_statistics and create_patterns, are unaffected by the EDT context and always operate on the whole design.

Arguments

- **-All**

  A switch that specifies to remove the data for all EDT blocks from the tool's internal database.

- ***block_name***

  A string that specifies to remove the data for a particular EDT block from the tool's internal database.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example displays currently defined EDT block names, then removes the definition for the one representing the current EDT block.

report_edt_blocks

```
//  my_core1_block
//  my_core3_block   (current block)
```

delete_edt_blocks my_core3_block
report_edt_blocks

```
//  Block deleted. my_core1_block is now the current EDT block.
```

Example 2

The following example shows how the tool manages the current EDT context when there are more than two blocks defined.

report_edt_blocks

```
// my_core1_block
// my_core2_block
// my_core3_block
// my_core4_block   (current block)
```

delete_edt_blocks my_core3_block
report_edt_blocks

```
// my_core1_block
// my_core2_block
// my_core4_block   (current block)
```

delete_edt_blocks my_core4_block

```
// Block deleted. No EDT block now selected as current block.
```

set_edt_options -channels 4

```
// Error: No current EDT block defined.
```

set_current_edt_block my_core2_block
set_edt_options -channels 4
report_edt_configuration

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//
// EDT block "my_core2_block"
// --------------------------
// IP version:                    3
// External scan channels:        4
// Bypass logic:                  On
// Lockup cells:                  On
// Clocking:                      edge-sensitive
//
// Note: Only current EDT block "my_core2_block" is reported. Use "report
// edt configuration -all_blocks" to report the configurations of all
// blocks.
```

Related Topics

add_edt_blocks

report_edt_blocks

set_current_edt_block

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_edt_configurations

Context: dft -edt

Mode: setup

Prerequisites: One or more compression configurations must be defined with the add_edt_configurations command.

Deletes specified compression configurations from all blocks in a design.

Usage

delete_edt_configurations **-All** | ***config_name***

Description

In a multi-block design flow, all subsequently added blocks inherit the defined compression configurations. The compression configuration parameters are then set for each block context.

Arguments

- **-All**

  A required switch that deletes all the compression configurations from all the blocks of the design.

- ***config_name***

  A required string that specifies the name of a single compression configuration to delete from all the blocks of the design.

Examples

The following example reports the currently defined compression configurations in a design, deletes one, and reports the remaining compression configurations.

report_edt_configuration

```
// my_core1_block
// IP version:                  1
// Bypass logic:                On
// Lockup cells:                On
// Clocking:                    edge-sensitive
// Input channels:   [manufacturing_test : 10] [system_test : 4]
// Output channels:  [manufacturing_test : 5] [system_test : 2]
// Bypass chains: 5
```

delete_edt_configurations system_test
report_edt_configuration

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// my_core1_block
// IP version:                1
// Bypass logic:              On
// Lockup cells:              On
// Clocking:                  edge-sensitive
// Input channels: [manufacturing_test : 10]
// Output channels:[manufacturing_test : 5]
// Bypass chains: 5
```

Related Topics

add_edt_configurations

report_edt_configurations

set_current_edt_configuration

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_false_paths

Context: dft -edt, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Removes definitions of false paths.

Usage

delete_false_paths [-All] [-Verbose]
[-From *source_node…* | -FROM_Clock *clock_signal_name…*]
[-TO *sink_node…* | -TO_Clock *clock_signal_name…*]
[{-THrough *through_node…*}…]

Description

The delete_false_paths command deletes from the tool's internal database, false paths you defined using the add_false_paths command.

Arguments

Tip

When using the following arguments, you can include any number of asterisks (*) in pin or instance pathnames. The command treats an asterisk as a wildcard character, enabling you to use it to match many pathnames in the design.

- -All

  An optional switch that specifies to delete all previously defined false paths.

- -Verbose

  An optional switch that directs the tool to display the definition of each deleted false path as it is deleted.

- -From *source_node…*

  An optional switch and repeatable string pair that specifies the starting point of a false path. A valid starting point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the output pins of the instance to be starting points. For a clock PI, the tool considers the outputs of all the non-transparent latches, flip-flops and RAMs associated with the clock PI to be starting points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

  If a specified source node does not translate to an internal gate, the tool transcribes an error message and does not delete the false path definition.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -FROM_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the starting point of a false path by its associated clock pin pathname. The tool interprets any pin pathname you specify with this argument to be the source of a clock signal and considers the outputs of all non-transparent latches, flip-flops and RAMs associated with this clock signal to be starting points.

  If a specified clock signal name does not translate to an internal gate, the tool transcribes an error message and does not delete the false path definition.

- -TO *sink_node…*

  An optional switch and repeatable string pair that specifies the end point of a false path. A valid end point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the input pins of the instance to be end points. For a clock PI, the tool considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with the clock PI to be end points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

  If a specified sink node does not translate to an internal gate, the tool transcribes an error message and does not delete the false path definition.

  Note

  For two port latches with ports: Port1 = (D1, CLK1), Port2 = (D2, CLK2), and primary input clocks CLK1 and CLK2, the command “delete_false_paths... -to CLK2” would delete false paths that have an end point at D2.

- -TO_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the end point of a false path by its associated clock pin pathname. The tool interprets any pin pathname you specify with this argument to be the source of an internal clock signal and considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with this clock to be end points.

  If a specified clock signal name does not translate to an internal gate, the tool transcribes an error message and does not delete the false path definition.

- {-THrough *through_node…*}…

  An optional, repeatable switch and repeatable string pair that specifies circuit node(s) through which the false path must pass. A valid through node is a pin or instance pathname. Clock PIs are not supported as through nodes and the tool makes no attempt to determine if a pin pathname is that of an internal clock pin.

  When you use multiple -Through arguments, the tool considers their order left-to-right in the command string to be the order in which a signal would pass through the nodes on the specified false path. For example, the following false path specification:

```
delete_false_paths -from A -through B C -through D E -to F
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

specifies all paths that start from A, pass through either B or C, then pass through D or E, and end at F. If you do not use the -Through argument when specifying a path, the tool considers all paths from the specified starting points to the specified end points to be false paths.

If a through node does not translate to an internal gate, the tool transcribes an error message and does not delete the false path definition.

Examples

The following example displays current false path definitions, then deletes one of them, echoing its definition as it is deleted:

report_false_paths -all

```
// False Path -from /my_design/a /my_design/b -through /my_design/u1
// False Path -to /my_design/c
// False Path -through /my_design/u24 /my_design/u25
// Total reported paths = 3
```

delete_false_paths -verbose -to /my_design/c

```
// Deleting false paths ...
// False Path -to /my_design/c ( /my_design/c/Y (36) )
// Deleted 1 false paths.
```

Refer to the add_false_paths command for examples of signal matches for pathnames specified using the asterisk (*) character.

Related Topics

add_false_paths

delete_multicycle_paths

read_sdc

report_false_paths

report_multicycle_paths

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_fault_sites

Context: dft -edt, patterns -scan

Mode: analysis

Removes bridge entries, delay paths, user-defined fault sites, and any associated faults from the current fault list. For UDFMs, all faults are removed.

Usage

Bridge Faults Usage

delete_fault_sites {**-All** | {***bridge_name***… **-NAME**}}

Path Delay Faults Usage

delete_fault_sites {**-All**| **-Unsensitizable_paths** | ***path_name***…}

UDFM Faults Usage

delete_fault_sites {**-All** | ***filename***}

Description

Depending on the current fault model, deletes specified fault sites (bridge entries or delay paths) from the current fault list. The delete_fault_sites command removes faults loaded with the read_fault_sites command. When a fault site is removed from the current fault list, the associated patterns are also removed from the internal pattern set.

For more information on bridge faults and the bridge fault definition file, refer to "The Static Bridge Fault Model" in the *Tessent Scan and ATPG User's Manual*.

For more information on path delay faults and the path definition file, refer to "Path Delay Test Set Creation" in the *Tessent Scan and ATPG User's Manual*.

For more information on UDFMs, see "About User-Defined Fault Modeling" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- **-All**

  A switch that specifies to remove all fault sites from the current fault list and deletes all faults associated with the deleted fault sites.

- ***bridge_name***

  A repeatable string that specifies a bridge name.

- **-NAME**

  A switch that specifies the argument strings preceding the switch are bridge names. This switch is required just once, whether you enter one or multiple ***bridge_name*** strings.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-Unsensitizable_paths**

  A switch that directs the tool to perform an ATPG analysis of each path, and to delete those proved unsensitizable. For a list of critical paths identified by static timing analysis, for example, use this switch to identify and delete the unsensitizable paths.

  Note

  This ATPG process may be very lengthy. A progress message is displayed after every 100 paths analyzed, as well as at the end of the analysis.

- ***path_name***

  A repeatable string that specifies the name of an existing path that resides in the current path definition file and whose path delay faults you want to remove from the current path delay fault list.

- ***filename***

  A required string that specifies a UDFM filename. Use this option to delete all fault sites associated with a specified UDFM file.

Examples

Example 1

The following example reads the bridge information in the bridge fault definition file*./user/design/bridgefile*, then deletes two of the three valid entries added to the fault site list:

set_fault_type bridge
read_fault_sites /user/design/bridgefile

```
//
//  Load bridge entries from file "/user/design/bridgefile".
//
//  5 bridge entries were read from file "/user/design/bridgefile".
//  2 bridge entries were skipped due to unknown net name.
//  3 bridge entries were added to the fault site list.
```

report_fault_sites

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
BRIDGE "U7.BRIDGE" {
  NET1 = "/G5";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
}
BRIDGE "U5.BRIDGE" {
  NET1 = "/G4";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
}
BRIDGE "U0.BRIDGE" {
  NET1 = "/G11";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
}
```

delete_fault_sites U5.BRIDGE U7.BRIDGE -name
report_fault_sites

```
BRIDGE "U0.BRIDGE" {
  NET1 = "/G11";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
}
```

Example 2

The following path delay example reads the path information from the file, */user/design/pathfile*, and deletes one of the two paths:

set_fault_type path_delay
read_fault_sites /user/design/pathfile
report_fault_sites

```
PATH “path0” =
    PIN /I$6/Q + ;
    PIN /I$35/B0 + ;
    PIN /I$35/C0 + ;
    PIN /I$1/I$650/IN + ;
    PIN /I$1/I$650/OUT - ;
    PIN /A_EQ_B + ;
END ;
PATH “path1” =
    PIN /I$6/Q + ;
    PIN /I$35/B1 + ;
    PIN /I$35/C1 + ;
    PIN /I$1/I$649/IN + ;
    PIN /I$1/I$649/OUT - ;
    PIN /I$5/D - ;
END ;
```

delete_fault_sites path0
report_fault_sites

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
PATH "path1" =
    PIN /I$6/Q + ;
    PIN /I$35/B1 + ;
    PIN /I$35/C1 + ;
    PIN /I$1/I$649/IN + ;
    PIN /I$1/I$649/OUT - ;
    PIN /I$5/D - ;
END ;
```

Example 3

The following example deletes all fault sites associated with the *c090_std.udfm* UDFM model:

delete_fault_sites /home/design/c090_std/c090_std.udfm

Example 4

The following example deletes all fault sites from the current fault list:

delete_fault_sites -All

Related Topics

add_fault_sites

add_faults

analyze_fault

delete_faults

read_fault_sites

report_fault_sites

set_fault_type

write_fault_sites

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_faults

Context: dft -edt, dft -test_points, patterns -scan

Mode: analysis

Removes faults from the current fault list.

Usage

delete_faults ***fault_model_specific_options*** [*power_aware_options*] [-KEEP_Patterns]
[{> | >>} *file_pathname*]

Stuck/Transition/Toggle/Iddq Faults Usage

[-All | {-Class {*class_name* | *class_code*}…}
| {*object_expression*… [-PIn | -Instance | -MODule [-PIN *module_pin_pathname* ...]]}
| {-CLOCK_Domains {ALL | *clock_pathname*…} [-NO_EQUivalent_clocks]
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]}
| {-CAPture_procedures {ALL | *capture_procedure_name*…}}
| {[-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls]}]
[-Stuck_at {01 | 0 | 1}]
[-FRom *pin* ...][-THrough *pin* ...][-TO *pin* ...][-FROM_Clock *pin* ...][-TO_Clock *pin* ...]
[-STOP_at {SEQUential_elements | SCAN_cells | PORTs_only}]
[-fanout_off_path_stop {on | off }] [-fanin_off_path_stop { on | off }]
[-VERbose]

UDFM Fault Usage

{-All | {-Class {*class_name* | *class_code*} ...}
| {*object_expression* ... [-INstance]}
| {-CLOCK_Domains {ALL | *clock_pathname* ...} [-NO_EQUivalent_clocks]
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]}
| {[-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls]}
| -SCAN_Data
| {[-UDFM_type *name*] [-CELL*name*] [-MODule *name*] [-FAULT *name*]}]
[-FRom *pin* ...][-THrough *pin* ...][-TO *pin* ...][-FROM_Clock *pin* ...][-TO_Clock *pin* ...]
[-STOP_at {SEQUential_elements | SCAN_cells | PORTs_only}]
[-fanout_off_path_stop {on | off }] [-fanin_off_path_stop { on | off }]
[-VERBose]

Bridge Fault Usage

[-All | {-Class {*class_name* | *class_code*}…}
| {{*bridge_name*… -NAME} | {*net_pathname*… -SINGle} | {*instance_name*… -SINGle}}
| {-CLOCK_Domains {ALL | *clock_pathname*…} [-NO_EQUivalent_clocks]
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]}
| {-CAPture_procedures {ALL | *capture_proc_name*…}}
| {[-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls]}]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Path Delay Faults Usage

[-All | *name_of_delay_path*… | -Class {*class_name* | *class_code*}…]
[-Both | -Rise | -Fall] [-UNTestable]

Power-Aware Options (applicable only after you have loaded CPF/UPF power data)

[-ON_domains | -OFf_domains | {-POWer_domains {*domain_name* …}}]
[-ISolation_cells] [-LEvel_shifters] [-REtention_cells]

Description

The delete_faults command deletes faults you added using the add_faults or read_faults command from the fault list. The power-aware options remove faults based on power domains and power features after you have loaded a CPF or UPF file.

Alternatively, you can also delete a large number of faults using the -delete switch of the read_faults command.

When you run this command, the tool discards all patterns in the current test pattern set by default. To save the current test patterns, you must explicitly write them with the write_patterns command before using the delete_faults command or use the -keep_patterns switch.

By default, the delete_faults command ignores the constraint values implied in the data path when deleting faults in the data path. However, the tool still considers the constraint values for clock path faults.

You cannot use a single command to delete_faults for clock domains, capture procedures, and specific classes, objects, or paths. Use a separate command instance for each.

When you delete bridge faults, the currently loaded faults sites remain intact.

Arguments

- -KEEP_Patterns

  An optional switch that specifies that the tool preserves the current test patterns while deleting faults. The default behavior is for the tool to delete the current test patterns as faults are deleted.

- **-All**

  A required switch that deletes all faults in the current fault list.

- ***name_of_delay_path***

  A required, repeatable string that specifies the name of the delay path to delete from the current fault list. The name must correspond to the name of a delay path defined in the path definition file.

- **-Class** {***class_name*** | ***class_code***}…

  A required switch and repeatable literal pair that deletes one or more classes of faults. The table below lists the valid fault ***class_code***s and their associated fault ***class_name***s.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 4-2. Fault Class Codes and Class Names

Fault Class Codes	Fault Class Names	Fault Class Coverage
FU	Full	TE+UT
TE	TEstable	DT+PD+AU+UD
DT	DETEcted	DS+DI+DR
DS	DET_Simulation	
DI	DET_Implication	
DR	DET_Robust (Path Delay Testing only)	
DF	DET_Functional (Path Delay Testing only)	
PD	POSDET	PT+PU
PU	POSDET_Untestable	
PT	POSDET_Testable	
AU	Atpg_untestable	
UD	UNDetected	UC+UO
UC	UNControlled	
UO	UNObserved	
UT	UNTestable	UU+TI+BL+RE
UU	UNUsed	
TI	TIed	
BL	Blocked	
RE	Redundant	

- - Both | -Rise | -Fall

  An optional switch that specifies which faults to delete for each path already added via the read_fault_sites command. These switches are valid only for path delay faults.

  -Both — Deletes both the slow-to-rise and slow-to-fall faults. This is the default.

  -Rise — Deletes only the slow-to-rise faults.

  -Fall — Deletes only the slow-to-fall faults.

- ***object_expression***

  A required string representing a list of instances or pins associated with the faults you want to delete from the current fault list. In the case of UDFM, you can specify only instances. You can use any number of asterisk (*) and question mark (?) wildcard characters.

  Instance pathnames must be Tessent Cell library cell instances. Pin pathnames must be Tessent Cell library cell instance pins, also referred to as design level pins. If the object

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

expression specifies a pin within an instance of a Tessent Cell library model, the tool ignores it. By default, the tool matches pin pathnames first. If it does not find a pin pathname match, the tool next tries to match instance pathnames. Force the tool to match only pin pathnames or only instance pathnames by including the -Pin or -Instance switch after the ***object_expression***.

- -PIN

  An optional switch that uses the ***object_expression*** to match only pin pathnames and then delete the associated faults.

- -INstance

  An optional switch that uses the ***object_expression*** to match only instances and then delete the associated faults.

- -MODule [-PIN *module_pin_pathname* ...] (Stuck/Transition/Toggle/Iddq only)

  A switch that interprets the ***object_expression*** argument as a module pathname. All instances of the module are affected. You can use the asterisk (*) and question mark (?) wildcards for the ***object_expression*** argument, and the tool adds the faults for all matching modules or library models.

  When you follow the -Module switch with the optional -PIN switch, you can specify a list of module pin pathnames for the pin within the module. The tool automatically finds all the instances with the given object_expression and deletes faults for every pin specified in module_pin_pathname list.

- -Untestable

  An optional switch that deletes all identified, untestable path delay faults. Untestable faults are common when using random patterns. This includes faults that the tool cannot detect due either to constraints or the use of a single capture clock.

- ***bridge_name***

  A required, repeatable string that specifies the name of a bridge. The tool deletes the faults associated with the specified bridge from the current fault list.

- ***net_pathname***

  A required, repeatable string that specifies the name of a single net. The tool deletes the faults associated with the specified net from the current fault list.

- ***instance_name***

  A required, repeatable string that specifies the name of an instance. The tool deletes the faults associated with the specified instance from the current fault list.

- -NAME

  An optional switch specifying that all entered strings are bridge names.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -SINGle

  An optional switch specifying that all the entered strings are either a single net or instance. The tool searches for a net first. If it cannot find a corresponding net, the tool searches for a corresponding instance.

  If the string specifies a net, the tool deletes each bridge entry associated with the net.

  If the string specifies an instance that contains a non-dominant net included in a bridge entry, the tool deletes the associated faults.

- **-CLOCK_Domains** {**ALL** | ***clock_pathname…***}

  A required switch and literal or repeatable string pair that specifies a list of clocks that the tool uses to decide the faults to add to the fault list, given the following requirements:

  - **Static Faults** — The tool adds a fault to the fault list if it can be captured by any clock in the specified list of clocks or any of its equivalent clocks.
  - **Transition Faults** — The tool adds a fault to the fault list if one of the following is true:
    i. The launch and capture clock are the same clock from the specified list of clocks
    ii. The launch and capture clock are synchronous clocks from the specified list of clocks
    iii. The launch or capture clock from the specified are equivalent clocks

  When you use this switch, the tool ignores the constraint values implied in the data path when adding faults in the data path. However, the tool still considers the constraint value for the clock cone tracing to determine the state element clock domain.

  ALL — A literal that specifies all the clocks in the design.

  clock_pathname — A repeatable string that specifies a particular clock.

  The tool deletes faults potentially detectable by a clock that you specify with this switch, even if they might also be detectable by an unspecified clock.

  The tool takes user-defined non-race clocks and faults between synchronous clock groups into account when adding or deleting faults by clock domain.

- -NO_EQUivalent_clocks

  An optional switch that prevents the -Clock_domains switch from deleting faults in equivalent clock domains.

- -EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains

  An optional switch used with -Clock_domains to instruct the tool to exclude the inter-clock faults for synchronous clocks. When you specify this, the tool considers only faults within clock domains when adding or deleting faults by clock domain.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-CAPture_procedures** {**ALL** | ***capture_procedure_name…***}

  A required switch and literal or repeatable string pair that specifies a list of enabled named capture procedures and directs the tool to delete faults that are potentially detectable by any of the specified procedures. The argument choices are as follows:

  ALL — A literal that specifies all enabled named capture procedures.

  capture_procedure_name — A repeatable string that specifies a particular enabled named capture procedure.

  Refer to the set_capture_procedures command description for information about enabling or turning off named capture procedures.

- -FRom *pin…*

  An optional switch and repeatable string pair that sets the start pin or pins of the cone of logic the tool uses to select faults to delete. When specified, the tool performs forward cone tracing from the specified pin or pins until it encounters a stopping point. You can specify one or more instance names rather than pin names. When you specify an instance name with -from, the tool uses its output pins when defining the cone.

- -THrough *pin…*

  An optional switch and repeatable string pair that sets the through pin or pins of the cone of logic the tool uses to select faults to delete. When specified, the tool performs forward and backward cone tracing from the specified pin or pins until it encounters a stopping point in both directions. You can specify one or more instance names rather than pin names. When you specify an instance name with -through, the tool uses its output pins when defining the cone.

- -TO *pin…*

  An optional switch and repeatable string pair that sets the end pin or pins of the cone of logic the tool uses to select faults to delete. When specified, the tool performs backward cone tracing from the specified pin or pins until it encounters a stopping point. You can specify one or more instance names rather than pin names. When you specify an instance name with -to, the tool uses its input pins when defining the cone.

  Note

  When you use -from, -to, and -through with other switches, the selected faults are the intersection of the cones traced for all specified switches.

- -FROM_Clock *pin ...*

  An optional switch and repeatable string pair that sets the launch clock pin or pins of the cone of logic the tool uses to select faults to delete. When specified, the tool performs forward cone tracing from the output ports of the state elements launched by the specified clock pin or pins until it encounters a stopping point.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -TO_Clock *pin* ...

  An optional switch and repeatable string pair that sets the capture clock pin or pins of the cone of logic the tool uses to select faults to delete. When specified, the tool performs backward cone tracing from the data ports of the state elements captured by the specified clock pin or pins until it encounters a stopping point.

- -STOP_at {SEQUential_elements | SCAN_cells | PORTs_only}

  An optional switch and literal pair that sets the stop condition for tracing the cone of logic the tool uses to select faults to delete. By default, the traced logic cone includes only combinational logic, so cone tracing stops at any sequential elements. When you specify "scan_cells", the cone includes non-scan cells and tracing stops only at scan cells. When you specify "ports_only", the cone includes any sequential elements and tracing stops only at primary inputs and outputs.

- -fanout_off_path_stop {on | off}

  An optional switch and literal pair that specifies an additional stop condition when tracing the fan-in fault cone backward and reaching a multiple fanout gate G. When this switch is on, tracing stops at gate G unless the cone already includes all of the fanout gates of G. The default is off. You can use a separate command, set_fault_selection_options, to change the default of this stop condition.

- -fanin_off_path_stop {on | off}

  An optional switch and literal pair that specifies an additional stop condition when tracing the fanout fault cone forward and reaching a multiple fan-in gate G. When this switch is on, tracing stops at gate G unless the cone already includes all of the fan-in gates of G. The default is off. You can use a separate command, set_fault_selection_options, to change the default of this stop condition.

- -Stuck_at 01 | 1 | 0

  An optional switch and literal pair that specifies the stuck-at or transition faults to delete from the fault list, according to the following table:

	01 (default)	**0**	**1**
Stuck-At	stuck-at-0 stuck-at-1	stuck-at-0	stuck-at-1
Transition	slow-to-rise slow-to-fall	slow-to-rise	slow-to-fall

The following switches, which support stuck-at, transition, toggle, Iddq, UDFM, and bridge faults, delete faults at certain locations that are ATPG untestable in functional mode or should not be tested (to avoid over-testing). For bridge faults, the switches apply to any bridge that has at least one end in a specified location. Use the switches to fine-tune the fault population and improve the quality of the tool's coverage statistics. You should take care to ensure that you are familiar enough with the design to understand the effects of these switches on the design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -SCAN_Enable

  An optional switch that deletes faults that only fan out to the select lines of multiplexers in the scan path. For this switch, a multiplexer is either a MUX simulation primitive or a nonprimitive type multiplexer composed of AND and OR gates. In essence, this switch deletes all faults that are in the fan-in cone of local scan enable signals and are dominated by them.

- -CLOCK_Cones

  An optional switch that deletes all faults in the clock cone. The clock cone is the intersection of the fan-in of the sequential element clock ports and the fanout of the clock sources. This switch considers any sequential elements, such as flops, latches, RAMs, and ROMs, not just scan cells.

- -IO

  An optional switch that deletes faults that either only PIs control or only POs observe. This switch applies only to PIs that are either not defined as a clock, or defined as clock (or read/write control) but constrained to be off during capture.

- -ASYnchronous_controls

  An optional switch that deletes all faults that only fan out to Set/Reset ports of state elements and RAMs.

- -SCAN_Data

  An optional switch that deletes all bridge faults where at least one of the bridging nets is located on the scan data path.

- -VERbose

  An optional switch that outputs the instance or pin names instead of the summary. You can optionally redirect this output to a file using the > or >> redirection operators.

  By default when you specify wildcard characters for instance (-INstance) or pin (-PIN) names, the tool outputs all found instances or pins matching the wildcard characters. For example:

  ```
  ANALYSIS> delete_faults */irep0/sigdout -verbose
  === Found 1 design pin ===
  path_to_instance/irep0/sigdout
  ```

  If you use actual instance or pin names instead of wildcards, then this switch has no effect.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Udfm_Type *name*

  An optional switch and string pair that specifies the name of a UDFM type. Use this option to delete all faults associated with a specific UDFM type from the current fault list. The name string is case insensitive and can include any number of asterisk (*) and question mark (?) wildcard characters.

- -CEll *name*

  An optional switch and string pair that specifies a library cell. Use this option to delete all faults associated with a specified cell from the current fault list. The name string is case insensitive and can include any number of asterisk (*) and question mark (?) wildcard characters.

- -MODule *name* **(UDFM only)**

  An optional switch and string pair that specifies a module. Use this option to delete all faults associated with a specified module from the current fault list. The name string is case insensitive and can include any number of asterisk (*) and question mark (?) wildcard characters.

- -Class *class_type*

  An optional switch and repeatable literal pair that deletes the internal fault entries that belong to the class specified by the *class_type* argument. The *class_type* argument can be either the fault class code or a fault class name. For more information on fault classes, refer to "Fault Classes" in the *Tessent Scan and ATPG User's Manual*.

- -FAult *name*

  An optional switch and string pair that specifies a UDFM fault name. The name string is case insensitive and can include any number of asterisk (*) and question mark (?) wildcard characters.

Power-Aware Options (applicable only after you have loaded CPF/UPF power data)

- -ON_domains

  An optional switch that removes faults on all power-on domains.

- -OFF_domains

  An optional switch that removes faults on all power-off domains.

- -Power_domains {*domain_name* …}

  An optional switch and repeatable string that removes faults on the specified power domains.

- -Isolation_cells

  An optional switch that removes faults on all isolation cells.

- -Level_shifters

  An optional switch that removes faults on all level shifters.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -REtention_cells

  An optional switch that removes faults on all retention cells.

Examples

Example 1

The following example deletes a stuck-at-0 fault from the current fault list after adding all the faults to the circuit, but before creating patterns:

```
set_system_mode analysis
add_faults -all
delete_faults i_1006/i1 -stuck_at 0
create_patterns
```

Example 2

The following example deletes all stuck-at-0 AU faults. The reduction in AU faults in the statistics report corresponds to the increase in the reported coverages as a result of removing these faults from the calculation. (For information on the formulas used to calculate coverage, refer to "Testability Calculations" in the *Tessent Scan and ATPG User's Manual*.)

```
ATPG> report_statistics

                Statistics Report
                Stuck-at Faults
---------------------------------------------
Fault Classes                        #faults
                                     (total)
----------------------------  ---------------
  FU (full)                          136996
  --------------------------  ---------------
  UO (unobserved)                     84 ( 0.06%)
  DS (det_simulation)            112635 (82.22%)
  DI (det_implication)            10037 ( 7.33%)
  PU (posdet_untestable)            199 ( 0.15%)
  PT (posdet_testable)                7 ( 0.01%)
  UU (unused)                        62 ( 0.05%)
  TI (tied)                          96 ( 0.07%)
  BL (blocked)                        1 ( 0.00%)
  RE (redundant)                   9867 ( 7.20%)
  AU (atpg_untestable)             4008 ( 2.93%)
---------------------------------------------
Fault Sub-classes
  --------------------------
  AU (atpg_untestable)
    BB   (black_boxes)             2278 ( 1.66%)
    Unclassified                   1730 ( 1.26%)
  UC+UO
    AAB  (atpg_abort)                 84 ( 0.06%)
---------------------------------------------
Coverage
  --------------------------
  test_coverage                          96.70%
  fault_coverage                         89.62%
  atpg_effectiveness                     99.94%
...
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ATPG> delete_faults -class au -stuck_at 0

```
//  Note: 619 collapsed faults were deleted.
```

ATPG> report_statistics

```
              Statistics Report
               Stuck-at Faults
---------------------------------------------
Fault Classes                       #faults
                                    (total)
----------------------------  ---------------
  FU (full)                          135789
  --------------------------  ---------------
  UO (unobserved)                     84 ( 0.06%)
  DS (det_simulation)            112635 (82.95%)
  DI (det_implication)            10037 ( 7.39%)
  PU (posdet_untestable)            199 ( 0.15%)
  PT (posdet_testable)                7 ( 0.01%)
  UU (unused)                        62 ( 0.05%)
  TI (tied)                          96 ( 0.07%)
  BL (blocked)                        1 ( 0.00%)
  RE (redundant)                   9867 ( 7.27%)
  AU (atpg_untestable)             2801 ( 2.06%)
---------------------------------------------
Fault Sub-classes
  --------------------------
  AU (atpg_untestable)
    BB   (black_boxes)             1654 ( 1.22%)
    Unclassified                   1147 ( 0.84%)
  UC+UO
    AAB  (atpg_abort)                  84 ( 0.06%)
---------------------------------------------
Coverage
  --------------------------
  test_coverage                          97.62%
  fault_coverage                         90.42%
  atpg_effectiveness                     99.94%
...
```

Example 3

After adding all of the faults in the design, this example removes only the faults from the power domain named PD1:

add_faults -all
delete_faults -power_domain PD1

Example 4

The following example deletes all UDFM type intra_cell_bridges associated with all library cells that begin with “mux”:

delete_faults -udfm_type intra_cell_bridges -cell mux*

Related Topics

add_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_nofaults

read_fault_sites

read_faults

report_fault_sites

report_faults

report_testbench_simulation_options

set_fault_mode

set_fault_sampling

set_fault_selection_options

set_fault_type

write_fault_sites

write_faults

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_flat_model

Context: all contexts

Mode: setup

Deletes a flat model that was previously created either by going to system mode analysis or by using the create_flat_model command.

Usage

delete_flat_model

Description

Deletes a flat model that was previously created either by going to system mode analysis or by using the create_flat_model command.

You typically do not need this command because any command you use that invalidates the flat model automatically deletes the flat model at the same time. The tool also automatically deletes the flat model when you enter insertion mode.

In addition, the tool deletes the flat model when you set an attribute on a design module, instance, port, or pin that has the preserve boundary option enabled. For more information, refer to the "set_attribute_options -preserve_boundary_in_flat_model" description.

To free up used memory, you can use this command to delete the flat model while still in setup mode if you do not need it anymore.

You cannot use this command to delete a flat model that was read in with the read_flat_model command. In this case, you must use the delete_design command to delete the flat model.

Arguments

None

Return Values

None

Related Topics

create_flat_model

delete_design

read_flat_model

set_attribute_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_gate_to_rtl_mapping

Context: all contexts

Mode: setup

Removes the specified rule for post-synthesis/post-layout to RTL instance name-mapping for DEF files.

Usage

delete_gate_to_rtl_mapping **-instance_name_map_index** *instance_name_map_index*

Description

Deletes the specified DEF post-synthesis/post-layout instance to RTL instance name-mapping rule created with the add_gate_to_rtl_mapping command.

Arguments

- **-instance_name_map_index** *instance_name_map_index*

  A required switch and integer value pair that specifies the index of the name-mapping rule to be removed. The index values for existing name-mapping rules are reported with the report_gate_to_rtl_mapping command.

Examples

The following command removes the name-mapping reported for index 5 shown in the report_gate_to_rtl_mapping example:

```
delete_gate_to_rtl_mapping -instance_name_map_index 5
```

A subsequent report_gate_to_rtl_mapping command then reports:

```
report_gate_to_rtl_mapping
Index  Original instance name       Mapped instance name
-----  ---------------------------  ------------------------
1      memA                         /MEMORY.mem1
2      memB                         /MEMORY.mem2
3      memC                         /MEMORY.mem3
4      memD                         /MEMORY.mem4
5      memF                         /MEMORY.mem6
```

Related Topics

add_gate_to_rtl_mapping

report_gate_to_rtl_mapping

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_icl_modules

Context: all contexts

Mode: setup

Deletes the specified ICL modules from memory.

Usage

delete_icl_modules {***object_spec*** | **-all**} [-silent]

Description

If any of the specified ICL modules are instantiated below the current design, the ICL elaboration tree is destroyed and you must recreate it using a new set_current_design command.

Arguments

- ***object_spec***

  A value that specifies one or more ICL modules to delete. The value is a Tcl list of one or more ICL module names, or a collection of one or more ICL module objects as returned by the get_icl_modules command. When this argument is omitted, all ICL modules are deleted.

- **-all**

  A switch that specifies to delete all ICL modules in the system.

- -silent

  An optional argument that specifies to not generate error messages when an ICL module specified by ***object_spec*** does not exist.

Return Values

None

Examples

The first example deletes three ICL modules. The second example deletes all ICL modules. The third example deletes all ICL modules with a name starting with a, b, or c.

```
delete_icl_modules {mychip tap tdr}
delete_icl_modules -all
delete_icl_modules [get_icl_modules {[abc].*} -regexp]
```

Related Topics

get_icl_modules

read_icl

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_icl_ports

Context: patterns -ijtag, dft

Mode: setup, insertion

Undoes the effect of the add_icl_ports command on a specified list of top level ports.

Usage

delete_icl_ports { ***port_objects*** | **-all** } [-silent]

Description

Undoes the effect of the add_icl_ports command on a specified list of top level ports.

Arguments

- ***port_objects***

  A required string that specifies a Tcl list of one or more design port names or a collection of one or more design or icl ports.

  The ports in the collection must not have been used as arguments of the -source switch of an add_icl_port command before. (A port that is already in use as a “source port”, cannot be deleted by means of delete_icl_ports.)

- **-all**

  A required switch that triggers the deletion of all previous specifications of the add_icl_ports command.

- -silent

  An optional switch that specifies to suppress the warning messages that are normally generated when the specified port names do not exist or were never the target of previous add_icl_ports commands.

Examples

The following example declares then deletes ICL DataInPort D1, and ICL DataOutPorts O1 O2 and O3.

```
set_current_design top
add_icl_ports D1 -type data_in
add_icl_ports {O1 O2 O3} -type data_out
delete_icl_ports {O1 O2 O3}
```

Related Topics

add_icl_ports

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_icl_scan_interfaces

Context: patterns -ijtag, dft

Mode: setup, insertion (dft context only)

Deletes the specified scan interfaces previously added by add_icl_scan_interfaces from the new ICL top module that is created during ICL extraction.

Usage

delete_icl_scan_interfaces ***name_list*** | **-All**

Description

Deletes the specified scan interfaces previously added by add_icl_scan_interfaces from the new ICL top module that is created during ICL extraction.

You must set the current design with set_current_design before using the delete_icl_scan_interfaces command.

Arguments

- ***name_list***

  A list that specifies the names of scan interfaces that ICL extraction deletes from the new top ICL module.

- **-All**

  A switch that specifies that all scan interfaces are deleted from the new ICL top module.

Examples

The following example deletes the ScanInterfaces named TAP and Internal from the new ICL top module during ICL extraction:

```
delete_icl_scan_interfaces {TAP Internal}
```

Related Topics

add_icl_scan_interfaces

get_icl_scan_interface_list

get_icl_scan_interface_port_list

set_icl_scan_interface_ports

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_icl_ssn_datapaths

Context: all contexts

Mode: setup, insertion

Deletes the SSN datapath specifications added with the add_icl_ssn_datapaths and set_icl_ssn_datapath_ports commands.

Usage

delete_icl_ssn_datapaths ***name_list*** | -all

Description

The command delete_icl_ssn_datapaths is used to undo specifications that you have made with the add_icl_ssn_datapaths and set_icl_ssn_datapath_ports commands. You can either remove a single SSN datapath or a list of SSN datapaths, or all SSN datapath specifications. The specified names must have been introduced as SSN datapath identifiers. Deleting a SSN datapath also deletes the associated specifications done by means of the set_icl_ssn_datapath_ports command.

Arguments

- ***name_list***

  A Tcl list of names that are removed from the set of registered SSN datapath identifiers.

- -all

  A switch that instructs Tessent Shell to remove all SSN datapath specifications.

Examples

```
SETUP> add_icl_ssn_datapaths mypath
SETUP> delete_icl_ssn_datapaths mypath
SETUP> add_icl_ssn_datapaths mypath
SETUP> set_icl_ssn_datapath_ports -name mypath -clock_inputs bus_clock \
     -bus_data_inputs bus_in[11:0] -bus_data_outputs bus_out[11:0]
SETUP> extract_icl
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_iddq_exceptions

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Deletes exceptions to Iddq restrictions added with the add_iddq_exceptions command.

Usage

delete_iddq_exceptions {***original_string***… | **-All**}

Description

Deletes exceptions to Iddq restrictions added with the add_iddq_exceptions command.

For more information about IDDQ, see "IDDQ Test Set Creation" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- ***original_string***

  A required repeatable string that identifies a site to which you previously applied Iddq exception. The original string is reported by the report_iddq_exceptions command and must be one of the following:

  - A string originally input to an add_iddq_exceptions command
  - A string generated automatically by the "analyze_restrictions -Int_float_auto_except" command.

- **-All**

  A required switch that specifies the deletion of all Iddq restrictions.

Examples

The following example removes an Iddq exception:

```
delete_iddq_exceptions /cpu/alu/U2392/
```

Related Topics

analyze_restrictions

set_iddq_checks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_ijtag_logical_connection

Context: patterns -ijtag

Mode: setup

Removes logical connections previously added to a design. You must specify either the source pin/port or the destination pin/port or both source and destination pins/ports.

Usage

delete_ijtag_logical_connection
-from *source_pin_or_port_name* | **-to *destination_pin_or_port_name***

Description

You must specify the source pin/port or the destination pin/port or both with this command. If both the source and destination pins/ports are specified, the command removes the logical connection between them. If only the source pin/port is specified, then all logical connections that have that pin/port as the source are removed. If only the destination pin/port is specified, then all logical connections that have that pin/port as the destination are removed.

Arguments

- **-from *source_pin_or_port_name***

  A switch and value pair that specifies the source pin/port for the logical connection, which should be a top level module port or a hierarchical instance pin in the current top design module.

- **-to *destination_pin_or_port_name***

  A switch and value pair that specifies the destination pin/port for the logical connection, which should be a top level module port or a hierarchical instance pin in the current top design module.

Examples

The following example adds three logical connections in the current design, then deletes one logical connection.

add_ijtag_logical_connection -from /i2/in -to /i2/out

add_ijtag_logical_connection -from /i3/in -to /i3/out

add_ijtag_logical_connection -from /i4/in -to /i4/out

report_ijtag_logical_connections

```
//  IJTAG Logical Connections:
//  From Source  To Destination   Inversion
//  ===========  ==============   =========
//  /i2/in       /i2/out          no
//  /i3/in       /i3/out          no
//  /i4/in       /i4/out          no
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_ijtag_logical_connection -from /i4/in -to /i4/out

report_ijtag_logical_connections

```
//  IJTAG Logical Connections:
//  From Source  To Destination   Inversion
//  ===========  ==============   =========
//  /i2/in       /i2/out          no
//  /i3/in       /i3/out          no
```

Related Topics

add_ijtag_logical_connection

report_ijtag_logical_connections

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_input_constraints

Context: all contexts

Mode: setup

Removes previously applied constraints from primary input pins.

Usage

Context: dft-scan and dft -test_points

delete_input_constraints
[-ALl | *primary_input_pin_name…* | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs | -EQuivalent *equivalent_pin…*] [*constraint_switch*]

Context: patterns -scan

delete_input_constraints
[-ALl | *primary_input_pin_name…* | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs | -EQuivalent *equivalent_pin…*] [*constraint_switch*] [-HOld] [-NO_Z] [-SLow_pad [-CELL *model_name*]]

Context: patterns -ijtag

delete_input_constraints
[-ALl | *primary_input_pin_name…* | -BIDI_Only | -BIDI_Exclude| -EQuivalent *equivalent_pin…*] [*constraint_switch*]

Description

The delete_input_constraints command removes constraints you previously applied to primary input pins using the add_input_constraints command.

Arguments

- -ALl

  An optional switch that specifies to remove all previously applied input constraints, including pin equivalents. This is the default. Constraints for power and ground pins that are added via attributes are not deleted. To remove only pin equivalents, use the -Equivalent switch.

- *primary_input_pin_name*

  An optional, repeatable string that specifies the name of a primary input pin from which to remove the constraint.

- -BIDI_Only

  An optional switch that specifies to remove previously applied constraints only from bidirectional pins.

- -BIDI_Exclude

  An optional switch that specifies to remove previously applied constraints only from primary input pins that are not bidirectional.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -SCAN_INputs

  An optional switch that specifies to remove previously applied constraints only from scan input pins.

- -EQuivalent *equivalent_pin*

  An optional switch and repeatable string that specify the name of a primary input pin from which to remove a previously defined equivalence specification.

- *constraint_switch*

  An optional switch that specifies a particular constraint to remove. The choices, from which you can specify only one, are as follows:

  -C0 — Constant 0

  -C1 — Constant 1

  -CX — Constant X (unknown)

  -CZ — Constant Z (high-impedance)

  -CT0 — Constant TIE0

  -CT1 — Constant TIE1

  -CTZ — Constant TIEZ

  Tip

  For additional information about the constraints associated with the preceding switches, refer to the same switches under the add_input_constraints command.

  -C_All — All C-type constraints

  -C_Default — All C-type constraints added using any of the following switches with the add_input_constraints command: -All, -Bidi_only, -Bidi_exclude, -Scan_inputs. This does not include C-type constraints added by specifying the primary input pin name explicitly.

  -C_Specified — C-type constraints added by specifying the primary input pin name explicitly.

- -HOld

  An optional switch that specifies to remove the “hold” restriction, if applicable, from the specified pins. Refer to the description of the -Hold argument to the add_input_constraints command for more information.

- -NO_Z

  An optional switch that specifies to remove the “no-Z” restriction, if applicable, from the specified pins. Refer to the description of the -No_z argument to the add_input_constraints command for more information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -SLow_pad

  An optional switch that specifies to remove the “slow-pad” restriction, if applicable, from the specified pins. Refer to the description of the -Slow_pad argument to the add_input_constraints command for more information.

- -CELL *model_name*

  An optional switch and string pair that, together with the -Slow_pad switch, specifies to prefix the instance name of each instance of the Tessent Cell library model of type *model_name* to the *primary_input_pin_name,* and look up each resulting name as the pin directly driven by the primary input.

Examples

The following example deletes constraints from two primary inputs.

```
delete_input_constraints indata2 -c1
delete_input_constraints indata4 -c0
```

Related Topics

add_input_constraints

get_input_constraints

report_input_constraints

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_instances

Context: all contexts

Mode: insertion

Deletes the specified instance objects from the current design.

Usage

delete_instances ***obj_spec*** [-silent]

Description

The obj_spec can be a Tcl list of one or more hierarchical instance names, or a collection of one or more instances.

Arguments

- ***obj_spec***

  A required value that specifies one or more instances to delete.The value is a Tcl list of one or more hierarchical instance names, or a collection of one or more instances.

  Tip

  If your ***object_spec*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- -silent

  An optional argument that specifies to not generate an error messages when an instance in the obj_spec list does not exist.

Examples

Example 1

The following example deletes the two specified instances in the Tcl list:

```
delete_instances {u1/u2 u1/u4}
```

Example 2

The following example deletes the collection of instances returned by the get_instances command:

```
delete_instances [get_instances ABC* -hierarchical -of_module XYZ]
```

Related Topics

create_instance

get_common_parent_instance

rename_instance

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

replace_instances

uniquify_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_iprocs

Context: all contexts

Mode: all modes

Deletes the specified list of iProcs attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by module_name.

Usage

delete_iprocs ***iproc_list*** **[**-of_module *module_name]* [-silent]

Description

Deletes the specified list of iProcs attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by module_name.

All iProcs attached to the ICL module are deleted when iproc_list is not specified.

Arguments

- ***iproc_list***

  A Tcl list of iProc names.

- -of_module *module_name*

  A required switch and value pair that constrains the command to delete only instances of iProcs attached to the module specified by *module_name*.

- -silent

  An optional switch that specifies to suppress error messages if the specified iProcs or module do not exist.

Return Values

None

Examples

The following example deletes the test1 iProc from module mychip and then searches and deletes any remaining iProcs on mychip that begin with “p”.

```
delete_iprocs test1 -of_module mychip
delete_iprocs [lsearch -inline -all [get_iproc_list p*] -of_module mychip]
```

Related Topics

get_iproc_argument_default

get_iproc_argument_list

get_iproc_body

get_iproc_list

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iProc

iProcsForModule

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_layout_core_information

Context: patterns -scan_diagnosis

Mode: analysis

In hierarchical layout-aware diagnosis, deletes all core instance data or the core instance data from a specified design.

Usage

delete_layout_core_information {**-chip_design_name** ***name*** | **-all**}

Description

To use this command you must have a valid LDB that you previously opened using the open_layout command.

Refer to "Diagnosis for Hierarchical Design" in the *Tessent Diagnosis User's Manual* for more information.

Arguments

- **-chip_design_name** ***name***

  A switch and string pair that specifies to delete the core instance data from the specified design.

- **-all**

  A switch that specifies to delete all the core instance data.

Related Topics

add_layout_core_information

report_layout_core_information

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_layout_verification

Context: patterns -scan_diagnosis

Mode: analysis

Reports or deletes stored layout verification information from the current layout database.

Usage

delete_layout_verification {**-List** | **-Design** ***design_index_in_list***}

Description

Reports or deletes stored layout verification information from the current layout database.

Arguments

- **-Design** ***design_index_in_list***

  A required switch and integer pair that specifies the layout verification data the tool deletes. Use the -list switch to print index numbers for all stored design layout verification information.

- **-List**

  A required switch that directs the tool to list all the designs and corresponding layout rule violation information stored in the layout database. Each design is listed with its assigned Index number, which is used as an argument to the -design switch. The -list switch does not delete any layout verification information, use the -design switch to delete layout verification data from the database.

Examples

In the following example, the diagnosis lists the layout verification and design information stored in the open_layout database:

delete_layout_verification -list

```
//  Note: Layout verification information corresponding to the following
//        designs can be deleted from the layout database
//
//  Index    Stored Violation Count  Flat model file name
//  1.       611                     my_design.flat
//  2.       643                     other_design_version.flat
//
//   Note: The current design is my_design.flat (Index 1.)
//   Note: To delete any of the above, please use
//         delete_layout_verification -design <design index in the list>
//
```

In the following example, the diagnosis deletes the layout verification information for design index 1 (my_design.flat):

delete_layout_verification -design 1

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Note: Deleting layout verification information corresponding
//        to my_design.flat.
//  Warning: Since the layout verification information for the
//           current design has been deleted, the layout will be closed.
```

Related Topics

create_layout

open_layout

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_lfsr_connections

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Removes connections between the specified primary pins and Linear Feedback Shift Registers (LFSRs).

Usage

delete_lfsr_connections ***primary_pin***… | **-All**

Description

The delete_lfsr_connections command deletes the connections between the LFSRs and the primary pins specified with the add_lfsr_connections command. You can use the report_lfsr_connections command to display all the current connections between LFSRs and primary pins.

Arguments

- ***primary_pin***

  A repeatable string that lists the primary pins whose connections to LFSRs you want to delete.

- **-All**

  A switch that deletes all of the connections between LFSRs and primary pins.

Examples

The following example changes the definition of an LFSR connection by deleting it and then re-adding it with a new definition:

```
add_lfsrs lfsr1 prpg 5 15 -serial -in
add_lfsr_taps lfsr1 2 3 4
add_lfsr_connections scan_in.1 lfsr1 2
delete_lfsr_connections scan_in.1
add_lfsr_connections scan_in.2 lfsr1 2
```

Related Topics

add_lfsr_connections

report_lfsr_connections

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_lfsr_taps

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Removes the tap positions from a Linear Feedback Shift Register (LFSR).

Usage

delete_lfsr_taps ***lfsr_name*** {***tap_position***… | **-All**}

Description

The delete_lfsr_taps command deletes the specified LFSR tap positions added with the add_lfsr_taps command. To display the current tap positions of all defined LFSRs, use the report_lfsrs command.

Arguments

- ***lfsr_name***

  A string that specifies the reference name of the LFSR whose tap positions you want to delete.

- ***tap_position***

  A repeatable string that specifies the list of tap positions that you want to delete from the *lfsr_name*.

- **-All**

  A switch that deletes all of the tap positions from the *lfsr_name*.

Examples

The following example changes an LFSR tap position by deleting it and then adding a new tap position:

```
add_lfsrs lfsr1 prpg 5 15 -serial -in
add_lfsrs lfsr2 Prpg 5 13 -serial -in
add_lfsr_taps lfsr1 2 3 4
add_lfsr_taps lfsr2 1 3
delete_lfsr_taps lfsr1 3
add_lfsr_taps lfsr1 1
```

Related Topics

add_lfsr_taps

report_lfsrs

set_lfsrs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_lfsrs

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Removes the specified Linear Feedback Shift Registers (LFSRs).

Usage

delete_lfsrs ***lfsr_name***… | **-All**

Description

The delete_lfsrs command deletes LFSRs defined with the add_lfsrs command. You can use the report_lfsrs command to display a list of the current LFSRs with their current values and tap positions. When you delete an LFSR, the tool also deletes all its taps and pin connections.

Arguments

- ***lfsr_name***

  A repeatable string that specifies the reference names of the LFSRs that you want to remove.

- **-All**

  A switch that deletes all defined LFSRs.

Examples

The following example changes the definition of an LFSR by deleting it and then re-adding it with a new definition:

```
add_lfsrs lfsr1 prpg 5 15 -serial -in
add_lfsrs lfsr2 prpg 5 13 -serial -in
add_lfsrs lfsr3 prpg 5 11 -parallel -out
delete_lfsrs lfsr3
add_lfsrs lfsr3 prpg 5 11 -parallel -in
```

Related Topics

add_lfsrs

report_lfsrs

set_lfsrs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_lists

Context: dft -edt, patterns -scan

Mode: analysis

Removes pins from the pin list the tool monitors and reports on during simulation.

Usage

delete_lists ***pin_pathname***... | **-All**

Description

The delete_lists command removes from the monitored pin list, pins that you previously added to the list using the add_lists command. To review the current list of pins, use the report_lists command. To put additional pins on the list, use the add_lists command.

Arguments

- ***pin_pathname***

  A repeatable string that specifies the pins you want to remove from the monitored pin list.

- **-All**

  A switch that removes all currently listed pins from the monitored pin list.

Examples

The following example removes an extra added output pin from the monitored pin list:

add_lists /i_1006/o /i_1007/o /i_1008/o
report_lists

```
3 pins are currently monitored.
/i_1006/o
/i_1007/o
/i_1008/o
```

delete_lists /i_1007/o
report_lists

```
2 pins are currently monitored.
/i_1006/o
/i_1008/o
```

Related Topics

add_lists

report_lists

set_list_file

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_loadboard_loopback_pairs

Context: all contexts

Modes: setup

Prerequisites: Design must be elaborated prior to using this command.

Deletes any loopback connection associated with the specified ports.

Usage

delete_loadboard_loopback_pairs **-all** | ***port_object_spec***

Description

You use this command to delete the loopbacks associated with specified ports or all of the loopbacks that were created with the add_loadboard_loopback_pairs command.

Arguments

- **-all**

  A switch that specifies to delete all loopbacks.

- ***port_object_spec***

  A required string that lists the ports from which to delete associated loopbacks. You may specify one or both ports of the associated loopback pairs.

Examples

Example 1

This example shows one loopback pair being deleted using the ***port_object_spec***:

add_loadboard_loopback_pairs -inputs {din[4] ue} -outputs {dout[5] dout[0]}
report_loadboard_loopback_pairs

```
-----------------------------
Inputs  Outputs  AC Time to Z
------  -------  ------------
din[4]  dout[5]  N/A
ue      dout[0]  N/A
-----------------------------
```

delete_loadboard_loopback_pairs {dout[5] din[4]}
report_loadboard_loopback_pairs

```
-----------------------------
Inputs  Outputs  AC Time to Z
------  -------  ------------
ue      dout[0]  N/A
-----------------------------
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

This example shows how you delete all loopback pairs using the **-all** switch:

add_loadboard_loopback_pairs -inputs {din[4] ue} -outputs {dout[5] dout[0]}
report_loadboard_loopback_pairs

```
-----------------------------
Inputs  Outputs  AC Time to Z
------  -------  ------------
din[4]  dout[5]  N/A
ue      dout[0]  N/A
-----------------------------
```

delete_loadboard_loopback_pairs -all
report_loadboard_loopback_pairs

```
-----------------------------
Inputs  Outputs  AC Time to Z
------  -------  ------------
-----------------------------
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_misrs

Context: pattern -scan

Mode: setup, analysis

Deletes MISRs you specify with the add_misrs command.

Usage

delete_misrs {**-all** | ***misr_name***}

Description

Deletes MISRs you specify with the add_misrs command.

This command is used with the hybrid TK/LBIST flow. Refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- **-all**

  A required switch that specifies removing all MISRs.

- ***misr_name***

  A required repeatable string that specifies the name of the MISR.

Examples

This example deletes a previously added MISR called misr_0, and then re-adds the MISR as blk1_misr.

add_misrs misr_0 -size 24 -seed 000000 -taps 21 22 24

delete_misrs misr_0

add_misrs blk1_misr -size 24 -seed 0000 -taps 21 22 24

report_misrs -all

```
list of MISRs
blk1_misr  MISR  length=24  in_stream_taps=20,21,23
blk1_misr_pre_unload_value = 000000000000000000000000
```

Related Topics

add_misrs

report_misrs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_multicycle_paths

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Removes definitions of multicycle paths previously read in using the read_sdc command.

Usage

delete_multicycle_paths [-All] [-Verbose]
[-Cycles *integer* [-Start | -End]]
[-From *source_node…* | -FROM_Clock *clock_signal_name…*]
[-TO *sink_node…* | -TO_Clock *clock_signal_name…*]
[{-THrough *through_node…*}…]

Description

The delete_multicycle_paths command deletes from the tool's internal database, multicycle path definitions previously read in by the read_sdc command.

Arguments

Tip

When using the following arguments, you can include asterisks (*) in pin or instance pathnames. The command treats an asterisk as a wildcard character, enabling you to use it to match many pathnames in the design.

- -All

  An optional switch that specifies to delete all multicycle path definitions.

- -Verbose

  An optional switch that directs the tool to display the definition of each deleted multicycle path as it is deleted.

- -Cycles *integer* [-Start | -End]

  An optional switch and integer value that define the multicycle paths to delete when the path's cycle count is equal to the value specified by *integer*.

  [-Start | -End]

  These argument options for the -Cycles switch, specify the cycle count is associated with the clock of the start point (-from) or the end point (-to). The default, -End, is uses the end point clock (-to).

- -From *source_node…*

  An optional switch and repeatable string pair that specifies the starting point of a multicycle path. A valid starting point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the output pins of the instance to be starting points. For a clock PI, the tool considers the outputs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

of all the non-transparent latches, flip-flops and RAMs associated with the clock PI to be starting points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

If a specified source node does not translate to an internal gate, the tool transcribes an error message and does not delete the multicycle path definition.

- -FROM_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the starting point of a multicycle path by its associated clock pin pathname. The tool interprets any pin pathname you specify with this argument to be the source of a clock signal and considers the outputs of all non-transparent latches, flip-flops and RAMs associated with this clock signal to be starting points.

  If a specified clock signal name does not translate to an internal gate, the tool transcribes an error message and does not delete the multicycle path definition.

- -TO *sink_node…*

  An optional switch and repeatable string pair that specifies the end point of a multicycle path. A valid end point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the input pins of the instance to be end points. For a clock PI, the tool considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with the clock PI to be end points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

  If a specified sink node does not translate to an internal gate, the tool transcribes an error message and does not delete the multicycle path definition.

  Note

  For two port latches with ports: Port1 = (D1, CLK1), Port2 = (D2, CLK2), and primary input clocks CLK1 and CLK2, the command “delete_multicycle_paths... -to CLK2” would delete_multicycle_paths that have an end point at D2.

- -TO_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the end point of a multicycle path by its associated clock pin pathname. The tool interprets any pin pathname you specify with this argument to be the source of an internal clock signal and considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with this clock to be end points.

  If a specified clock signal name does not translate to an internal gate, the tool transcribes an error message and does not delete the multicycle path definition.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -THrough *through_node…*

  An optional, repeatable switch and repeatable string pair that specifies circuit node(s) through which the multicycle path must pass. A valid through node is a pin or instance pathname. Clock PIs are not supported as through nodes and the tool makes no attempt to determine if a pin pathname is that of an internal clock pin.

  When you use multiple -Through arguments, the tool considers their order left-to-right in the command string to be the order in which a signal would pass through the nodes on the specified multicycle path. For example, the following multicycle path specification:

```
delete multicycle paths -from G -through H I -through J
K -to L
```

  specifies all paths that start from G, pass through either H or I, then pass through J or K, and end at L. If you do not use the -Through argument when specifying a path, the tool considers all paths from the specified starting points to the specified end points to be false paths.

  If a specified through node does not translate to an internal gate, the tool transcribes an error message and does not delete the multicycle path definition.

Examples

The following example displays current multicycle path definitions, then deletes one of them, echoing its definition as it is deleted:

report multicycle paths -all

```
// Multicycle Path -from /my_design/a -through /my_design/u1 -cycles 2
// Multicycle Path -to /my_design/c -cycles 3
// Multicycle Path -through /my_design/u24 /my_design/u25 -cycles 2
// Total reported multicycle paths = 3
```

delete multicycle paths -verbose -to /my_design/c

```
// Multicycle Path -to /my_design/c (/my_design/c/Y (130)) -cycles 2
// Deleted 1 multicycle path.
```

Refer to the add_false_paths command for examples of signal matches for pathnames specified using the asterisk (*) wildcard character.

Related Topics

add_false_paths

delete_false_paths

read_sdc

report_false_paths

report_multicycle_paths

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_nets

Context: all contexts

Mode: insertion

Removes net objects inside an instance of a design module.

Usage

delete_nets ***obj_spec*** [-silent]

Description

You cannot delete a net that has a fan-in or fanout. You can delete only dangling nets.

You cannot delete a net that is also a pin of an instance. You can delete the corresponding input pin using the delete_pins command, but the net remains if it has objects in its fanout. You can delete the corresponding output pin using the delete_pins command, but the net remains if it has objects in its fan-in.

In general, you cannot delete a net that is driven by any RTL construct; however, an exception is that you can delete a dangling net driven by a Verilog assign statement. When you delete such a net, you also delete the assign statement.

Arguments

- ***obj_spec***

  A required value that specifies a Tcl list of one or more net names or a collection of one or more net objects as returned by the get_nets command. If the specified net object is part of a bus net, the entire bus is deleted.

  Tip

  If your ***object_spec*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- -silent

  An optional argument that specifies to ignore specified objects that do not exist and to not generate error messages.

Examples

Example 1

The following example creates a connection between two pins using a specific net name. Once created, the net cannot be deleted until it is no longer used in any connection.

```
create_connections INSTA/I1 INSTB/O1 -net_name mynet

get_nets mynet
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{mynet}
```

delete_net mynet

```
// Error: Cannot remove net 'mynet' in scope 'mychip' as it is being used
// in expression(s).
// Error: Error on 'delete_nets' at 'mynet'.
```

delete_connections { INSTA/I1 INSTB/O1}

delete_net mynet

get_nets mynet -silent

```
{}
```

Example 2

The following example checks for nets inside instance u1 with no fan-in, disconnects their fanout and deletes them.

```
foreach_in_collection net [get_nets u1/*] {
   if {[sizeof_collection [get_fanins $net]] == 0} {
      delect_connections [get_fanouts $net -stop_on pin]
      delete_nets $net
  }
}
```

Related Topics

create_net

delete_pins

get_icl_pins

get_icl_ports

get_nets

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_nofaults

Context: dft -edt, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup

Removes the nofault settings from either the specified pin or instance/module pathnames.

Usage

delete_nofaults {**-All** | ***modulename*** ... **-Module** [-PIN *module_pin_pathname* ...] | ***object_expression*** ... [-PIN | -Instance]} [-Stuck_at {01 | 0 | 1}] [-Class {System | User | Full}] [-VERBose] [(> | >>) *file_pathname*]

Description

The delete_nofaults command deletes nofault settings you added using the add_nofaults command.

You can optionally specify nofault settings that have a specific stuck-at or transition value. For the latter, you specify 0 for “slow-to-rise” and 1 for “slow-to-fall” transition faults. If you do not specify a stuck-at (transition) value when deleting a nofault setting, the command deletes both stuck-at (transition) nofault settings.

You can use the report_nofaults command to display all the current nofault settings.

You can also optionally specify nofault settings that have a specific class code: user-defined, system netlist, or both. If you do not specify a class code, then the command deletes the nofault setting from the user class.

Arguments

- **-All**

  A switch that deletes all nofault settings.

- ***modulename***

  A repeatable string that specifies the name(s) of the module(s) from which you want to delete nofault settings. You must include the -Module switch when you specify a module name.

- **-Module**

  A switch that specifies interpretation of the *modulename* argument as a module pathname. All instances of these modules are affected. You can use the asterisk (*) and question mark (?) wildcards for the ***modulename*** argument, and the tool deletes the nofault for all matching modules or library models.

  When you follow the -Module switch with the optional -PIN switch, you can specify a list of module pin pathnames for the pin within the module. The tool automatically finds all the instances with the given module name and deletes nofaults for every pin specified in module_pin_pathname list.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ***object_expression***

  A string representing a list of pathnames of instances or pins from which you want to delete nofault settings. You can use regular expressions, which may include any number of asterisk (*) and question mark (?) wildcard characters.

  Pin pathnames must be Tessent Cell library cell instance pins, also referred to as design level pins. If the object expression specifies a pin within an instance of a Tessent Cell library model, the tool ignores it. By default, pin pathnames are matched first. If a pin pathname match is not found, the tool next tries to match instance pathnames. You can force the tool to match only pin pathnames or only instance pathnames by including the -Pin or -Instance switch after the *object_expression*.

- -Pin

  An optional switch that specifies to use the preceding object expression to match only pin pathnames; the tool then deletes nofault settings from all the pins matched.

- -Instance

  An optional switch that specifies to use the preceding object expression to match only instance pathnames; the tool then deletes nofault settings from all boundary and internal pins of the instances matched.

- -Stuck_at 01 | 0 | 1

  An optional switch and literal pair that specifies the stuck-at or transition values from which you want to remove a nofault setting. The choices are as follows:

  01 — A literal specifying that for stuck-at faults the tool delete both “stuck-at-0” and “stuck-at-1” nofault settings; or for transition faults the tool delete both “slow-to-rise” and “slow-to-fall” nofault settings. This is the default.

  0 — A literal that deletes only the “stuck-at-0” nofault settings (“slow-to-rise” nofault settings for transition faults).

  1 — A literal that deletes only the “stuck-at-1” nofault settings (“slow-to-fall” nofault settings for transition faults).

- -Class User | System | Full

  An optional switch and literal pair that specifies the source (or class) of the nofault settings that you want to delete. The valid literals are as follows:

  User — A literal that deletes the user-entered nofault settings. This is the default.

  System — A literal that deletes netlist-based nofault settings.

  Full — A literal that deletes all the nofault settings in the user and system classes.

- -VERbose

  By default when you specify wildcard characters for instance (-INstance) or pin (-PIN) names, the tool outputs a summary message similar to the following:

```
// Note:  Adding faults for 330 fault sites.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When you specify the optional -VERbose switch, then the tool outputs the instance or pin names instead of the summary. You can optionally redirect this output to a file using the > or >> redirection operators.

If you use actual instance or pin names instead of wildcards, then this switch has no effect.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example deletes nofault settings from the pins of instance i_1007 and then adds all faults to the circuit, thereby enabling the tool to add faults to the pins of the i_1007 instance:

```
add_nofaults i_1006 i_1007 i_1008 -instance
delete_nofaults i_1007 -instance
set_system_mode analysis
add_faults -all
```

Related Topics

add_faults

add_synchronous_clock_group

delete_faults

report_faults

report_nofaults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_nonscan_instances

Context: dft, dft -scan, dft -test_points

Mode: setup

Resets any user-specified is_non_scannable attribute on the specified objects.

Usage

delete_nonscan_instances -INStances *instance_spec* | -Modules *module_spec* | -scan_elements *scan_element_spec* | -All | -unmapped_flop_cells | -library_models

Description

Resets any user-specified is_non_scannable attribute on the specified objects, thus stopping to exclude them from the scan insertion process.

The delete_nonscan_instances command resets the user-specified is_non_scannable attribute (see add_nonscan_instances) on the specified instances, all instances within the specified modules, the specified scan elements, or all instances. Please note that if the is_non_scannable attribute was set for any other reason(s), it remains set after this operation.

Arguments

- -INStances *instance_spec*

  An optional switch and value pair that specifies to delete the specified instances from the non-scan instance list. *instance_spec* can be a Tcl list of one or more instances or a collection of one or more instances. If the instance is hierarchical, then all instances beneath it are also deleted from the non-scan instance list.

- -Modules *module_spec*

  An optional switch and value pair that specifies to delete all instances within the specified modules from the non-scan instance list. *module_spec* can be a Tcl list of one or more modules or a collection of one or more modules. Note: module may specify the name of a Verilog module, or a Tessent library model.

- -scan_elements *scan_element_spec*

  An optional switch and value pair that specifies to delete the specified scan elements from the non-scan scan_element list. *scan_element_spec* can be a Tcl list of one or more scan elements or a collection of one or more scan elements.

- -All

  An optional switch that specifies to delete all instances from the non-scan instance list.

- -unmapped_flop_cells

  An optional switch that specifies to delete from the non-scan instance list every flop cell not already mapped to a scan cell in the design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -library_models

  An optional switch that specifies to delete from the non-scan instance list all nonscan_instances that were marked as such by using the -module switch followed by a Tessent library model name. This switch serves as a replacement for the obsolete "delete_(non)scan_models -all" command.

Examples

The following example deletes an extra sequential non-scan instance called i_1007.

```
add_nonscan_instances -instances i_1006 i_1007 i_1008
delete_nonscan_instances -instances i_1007
insert_test_logic
```

Related Topics

add_nonscan_instances

report_seq_transparent_procedures

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_notest_points

Context: dft -scan, dft -test_points

Mode: setup, analysis

Enables test logic insertion in the specified regions.

Usage

delete_notest_points {{***pin_pathname***… | ***instance_pathname…*** | ***instance_expression***} | **-ALL** | {**-Path** ***critical_pathname***} | **-ALL_Paths}**

Description

The delete_notest_points command is used to enable test logic insertion in regions of the design that were previously excluded from use because the add_notest_points command was used, or the underlying no_control_point/no_observe_point attributes were set directly. This command works by setting the no_control_point and no_observe_point attributes to false (which enables test logic insertion). This means a simple scenario such as preventing test points everywhere in the design except for one particular block can be done as follows:

```
add_notest_points /
delete_notest_points /block1
```

One obvious problem is that the no_control_point and no_observe_point attributes actually have three values (true, false, and unspecified). Simply setting the attribute for the top level instance to false when “delete_notest_points -all” is issued does not produce the required result. Instead, the -all switch actually sets the no_control_points and no_observe_points attributes to unspecified for all instances and pin_pathnames within the design. This also occurs when you use “delete_notest_points /”.

For more information, refer to Attribute Inheritance Behavior in Chapter 6.

Arguments

- ***pin_pathname***

  A repeatable string that specifies a list of pins on which you want to enable test logic insertion.

- **-All**

  A switch that enables test logic insertion in the entire netlist by setting the no_control_point and no_observe_point attributes to unspecified for all pin_pathnames, instance_pathnames, and instance_expressions.

- ***instance_pathname***

  A required repeatable string that enables test logic insertion on or inside the specified instance by setting the no_control_point and no_observe_point attributes to false for this instance.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The no_control_point and no_observe_point attribute for the gate-pins inside this instance are NOT affected in any way.

- ***instance_expression***

  A string representing a list of instances within the design. The string *instance_expression* is defined as:

  ```
  { string | string * } ...
  ```

  The asterisk (*) is a wildcard that enables you to match many instances in a design. Any expression that does not contain an asterisk (*) matches exactly zero or one instance.

  This argument does not support pathnames to objects below the instance level of a Tessent Cell library model. You can use a pathname expression to select several instances and the tool then enables test logic insertion for all the pins on those instances; but if the expression specifies a location below the instance level of a Tessent Cell library model, the tool reports an error message.

- **-Path *critical_pathname* (dft -test_points context only)**

  A required switch and name pair that specifies to delete the named critical path. You can list the names of the critical paths using the report_notest_points command with the -Paths switch. For more information on the format of the file, refer to "The Path Definition File" in the *Tessent Scan and ATPG User's Manual*.

- **-ALL_Paths**

  A required switch that specifies to delete all critical paths.

Examples

Example 1

The following example deletes an incorrect notest circuit point and corrects it with a new circuit point before performing testability analysis:

```
set_system_mode analysis
add_notest_points tr_i ts_i
delete_notest_points tr_i
add_notest_points tr_io
```

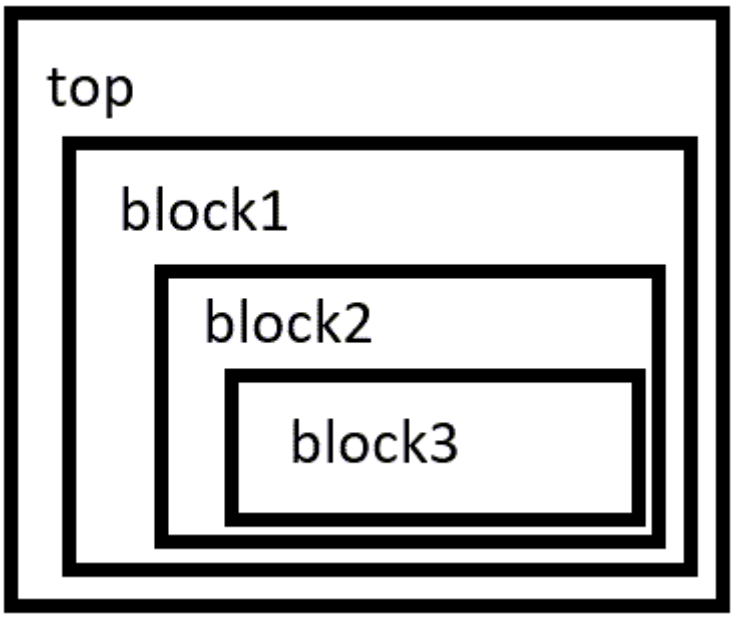
Example 2

You must be careful when using the add and delete_notest_point commands in conjunction with introspection. Figure 4-1 shows a hierarchical design used in the following example:

```
add_notest_points [get_instances -below_instance /block1]

delete_notest_points /block1/block2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 4-1. Hierarchical Test Logic Insertion Example



The add_notest_points command sets the no_control_point and no_observe_point attributes for block2 and block3. The delete_notest_points command sets the attribute for only block2 to false. The tool inserts test logic in the gates, pins, and nets inside block2. Any other module instantiated below block2, such as block3, remains out of bounds for the test logic insertion tool. This is because the add_notest_points command set the attributes for all the instances below /block1. If you want to insert test logic for block3, use the following command:

```
delete_notest_points [get_instances -below /block1/block2]
```

The get_instances command returns block3, which is the only instance inside block1 and block2.

Related Topics

add_output_masks

report_notest_points

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_output_masks

Context: dft, dft -edt, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Removes the masking of the specified primary output pins.

Usage

delete_output_masks ***primary_output***... | **-All**

Description

The tools use primary output pins as the observe points during the fault detection process. When you mask a primary output pin with the add_output_masks command, the tools mark that pin as an invalid primary output during the fault detection process.

You can set a default mask for all output and bidirectional pins using the set_output_masks command. You can add a hold value to a default mask with the add_output_masks command, or remove a hold value using the delete_output_masks command. To turn off the default masks for all output pins, you must use the "set_output_masks Off" command.

Note

The delete_output_masks command can only be used to delete pin constraints added in the same mode. Pin constraints added in setup mode cannot be deleted in analysis mode.

Arguments

- ***primary_output***

  A repeatable string that specifies the names of the primary output pins that you want to unmask.

- **-All**

  A switch that unmasks all primary outputs masked using the add_output_masks command.

Examples

The following example first incorrectly chooses two of the design's primary output pins to mask. The example then unmasks the one primary output that was inappropriate, masks the correct primary output, and then displays the complete list of currently-masked primary output pins no longer available as observation points:

```
add_output_masks q1 qb3
delete_output_masks q1
add_output_masks qb1
report_output_masks

TIEX       qb1
TIEX       qb3
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

add_output_masks

report_output_masks

set_output_masks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_pade_logical_connection

Context: dft

Mode: setup, analysis, insertion

Removes logical connections previously added to a design by the add_pade_logical_connection command. You must specify either the source pin/port or the destination pin/port or both source and destination pins/ports.

Usage

delete_pade_logical_connection
-from *source_pin_or_port_name* | **-to *destination_pin_or_port_name***

Description

You must specify the source pin/port or the destination pin/port or both with this command. If you specify both the source and destination pins/ports, the command removes the logical connection between them. If only the source pin/port is specified, the tool removes all logical connections that have that pin/port as the source. If only the destination pin/port is specified, then all logical connections that have that pin/port as the destination are removed.

Arguments

- **-from *source_pin_or_port_name***

  A switch and value pair that specifies the source pin/port for the logical connection, which should be a top-level module port or a hierarchical instance pin in the current top design module.

- **-to *destination_pin_or_port_name***

  A switch and value pair that specifies the destination pin/port for the logical connection, which should be a top-level module port or a hierarchical instance pin in the current top design module.

Examples

The following example adds three logical connections in the current design, then deletes one logical connection.

add_pade_logical_connection -from /i2/in -to /i2/out

add_pade_logical_connection -from /i3/in -to /i3/out

add_pade_logical_connection -from /i4/in -to /i4/out

report_pade_logical_connections

```
//  PADE Logical Connections:
//  From Source  To Destination   Inversion
//  ===========  ==============   =========
//  /i2/in       /i2/out          no
//  /i3/in       /i3/out          no
//  /i4/in       /i4/out          no
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_pade_logical_connection -from /i4/in -to /i4/out

report_pade_logical_connections

```
//  PADE Logical Connections:
//  From Source  To Destination   Inversion
//  ===========  ==============   =========
//  /i2/in       /i2/out          no
//  /i3/in       /i3/out          no
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_pattern_trigger

Context: patterns -silicon_insight

Mode: all modes

Deletes pattern triggers set with the add_pattern_trigger command.

Usage

delete_pattern_trigger ***pattern_name*** [-vector_address *vector_address*]

Arguments

- ***pattern_name***

  A required string that specifies the name of the pattern from which to delete the trigger.

- -vector_address *vector_address*

  An optional switch and nonnegative integer that specifies the vector address for the trigger to delete. If you do not specify an address, the command deletes all pattern triggers.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_patterns

Context: dft -edt, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis, patterns -ijtag

Mode: analysis

Deletes the current pattern set.

Usage

Context: patterns -scan

delete_patterns [-external | -internal] [-pattern_sets [chain] [scan]] [-keep_fault_status] [-force]

Context: patterns -ijtag

delete_patterns [-pattern_sets *pattern_set_list*] [-force]

Context: patterns -scan_retargeting

delete_patterns **-core *name*** [-mode *name*] [-silent]

Description

This command removes either the internal or external pattern set or both from the current session.

You can only delete generated pattern sets (that is, patterns generated by the tool and not read in) that have been written out using the write_patterns command, unless you use the -force switch. This restriction prevents you from deleting patterns that you have generated but have not yet written out to disk. In the -ijtag context, you can delete only patterns that have been closed.

Arguments

- -external | -internal **(patterns -scan context only)**

  Optional switches that specify which pattern source to remove. By default, the command removes both internal and external pattern sets.

- -pattern_sets [chain] [scan] **(patterns -scan only)**

  An optional switch and keyword pair that specifies which pattern type (scan or chain) to remove from the current session. By default, the tool removes both scan and chain patterns.

- -pattern_sets *pattern_set_list* **(-ijtag and patterns -scan_retargeting context only)**

  An optional switch and string that specifies a Tcl list of pattern set names that were created using the open_pattern_set command. When you do not use this switch, the command deletes all pattern sets.

- -keep_fault_status **(-scan context only)**

  Optional switch that keeps the current fault status when removing the internal patterns. When deleting the internal patterns, the status of existing faults is reset by default if you do not use this switch.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -force

  An optional switch that forces removing the internal patterns even when they are not stored previously. By default, the tool does not permit you to remove the unsaved internal patterns. However, the -force option enables you to delete the unsaved internal patterns without this safeguard. You can always remove the external patterns.

- **-core *name* (patterns -scan_retargeting only)**

  A required switch that specifies the cores from which pattern sets are to be deleted. If the core ***name*** does not exist, the patterns for all cores are deleted in the pattern retargetable phase.

- -mode name **(patterns -scan_retargeting only)**

  An optional switch that specifies which modes of the core the patterns should be deleted from. If -mode is not specified, all modes of the core are deleted.

- -silent **(patterns -scan_retargeting only)**

  An optional switch that specifies not to report patterns as they are deleted. By default, deleted patterns are reported.

Examples

Example 1

The following example removes all pattern sets currently in memory:

```
delete_patterns
```

Example 2 (-scan context)

The following example removes only the chain patterns from the internal pattern set. The -force option suppresses the error that would otherwise occur if you had not already saved the internal patterns using the write_patterns command.

```
delete_patterns -internal -pattern_sets chain -force
```

Example 3 (-ijtag context)

The following example removes only the patterns in the Tcl list named *pattern_list5*:

```
delete_patterns -pattern_sets patterns_list5
```

Related Topics

read_patterns

simulate_patterns

write_patterns

open_pattern_set

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_pins

Context: all contexts

Mode: insertion

Removes pin objects from a design module instance.

Usage

delete_pins ***obj_spec*** [-silent]

Description

If the pin to be deleted has an internal connection, the internal nets associated with the pin are kept; otherwise, they are also removed.

Note

Deleting one bit of a bus deletes the entire bus.

If the "set_insertion_options -auto_uniquify_edited_modules" option is On, the module is first uniquified before deleting a pin on the specified instance. If the -auto_uniquify_edited_modules option is Off, the pin is deleted on the module, and all instances of the module.

Arguments

- ***obj_spec***

  A required value that specifies a Tcl list of one or more pin names, or a collection of one or more pin objects as returned by the get_pins command. If the specified pin object is part of a bus net, the entire bus is deleted.

  Tip

  If your ***object_spec*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- -silent

  An optional argument that specifies to not generate an error message if an object within the specified obj_spec does not exist.

Examples

Example 1

The following example deletes the specified pins names. On instance u1, pin A is a scalar port, pin B is a bus pin with indices [3:0], and pin C is also a bus pin with indices [7:0]. All three pins are completely deleted even though the complete bus range was not specified for pins B and C.

```
get_pins {u1/A u1/B u1/C}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{u1/A u1/B[3] u1/B[2] u1/B[1] u1/B[0] u1/C[7] u1/C[6] u1/C[5] u1/C[4]u1/
C[3] u1/C[2] u1/C[1] u1/C[0]}
```

delete_pins {u1/A u1/B u1/C[1]}
get_pins {u1/A u1/B u1/C}

```
//  Warning: No result. Please check your filtering options.
{}
```

Example 2

The following example deletes the collection of pin objects returned by the get_pins command.

delete_pins [get_pins {u1/A u1/B u1/C[1]} -silent]

Related Topics

create_pin

get_icl_pins

get_icl_ports

get_nets

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_ports

Context: all contexts

Mode: insertion

Removes port objects from a design module.

Usage

delete_ports ***obj_spec1*** [-on_module *obj_spec2*] [-silent]

Description

If the port to be deleted has an internal connection, the internal nets associated to the port are kept; otherwise they are removed too.

If obj_spec1 is a collection of ports, those names refers to ports on specific modules and the specified -on_module option is ignored.

Note

Deleting one bit of a bus port deletes the entire bus port.

Arguments

- ***obj_spec1***

  A required value that specifies a Tcl list of one or more port names or a collection of one or more port objects.

- **-on_module *obj_spec2***

  An optional switch and value pair that specifies the module from which the ports are to be deleted. The *obj_spec2* value can be the name of a module or a module object returned by the command. If the -on_module switch is not specified, this command deletes the port from the current design, which is the module specified by the set_current_design command. You cannot delete a port on a module that was read in from a file and that is not instantiated inside the current design. It is possible to delete ports on modules created with the create_module command.

- -silent

  An optional argument that specifies to not generate an error message if an object within the specified obj_spec1 or obj_spec2 does not exist.

Examples

Example 1

The following example deletes the specified port names of module ModA. On ModA, port A is a scalar port, port B is a bus port with indices [3:0], and port C is also a bus port with indices [7:0]. All three ports are completely deleted even if the complete bus range was not specified for port B and C.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
delete_ports {A B C[1]} -on_module ModA
```

Example 2

The following example deletes the collection of port objects returned by the get_ports command. Notice that the -on_module option is not specified for the delete_ports command because the port objects returned by the get_ports command are already associated to a specific module.

```
delete_ports [get_ports {A B C[1]} -of_module ModA]
```

Related Topics

create_port

get_icl_ports

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_primary_inputs

Context: all contexts

Mode: setup

Removes the specified primary inputs from the existing flat model or, if a flat model does not yet exist, from the flattened result.

Usage

delete_primary_inputs {***net_pathname***... | ***primary_input_pin***... | **-All**}
[-Class {User | System | Full}]

Description

The delete_primary_inputs command deletes from a circuit the primary inputs that you specify. You can delete either the user class, system class, or full classes of primary inputs. If you do not specify a class, the tool deletes the primary inputs from the user class.

You can display a list of any class of primary inputs by using the report_primary_inputs command.

Note

Once you delete a primary input pin with this command, adding the pin back again with the add_primary_inputs command does not create the same pin. Bus information is lost and the pin is marked as user-added. Take care to use delete_primary_inputs only if you want to truly delete a pin from the design so it is not used.

Arguments

- ***net_pathname***

  A repeatable string that specifies the circuit connections that you want to delete. You can specify the class of primary inputs to delete with the -Class switch.

- ***primary_input_pin***...

  A repeatable string that specifies a list of primary input pins that you want to delete. You can specify the class of primary inputs to delete with the -Class switch.

- **-All**

  A switch that deletes all primary inputs. You can specify the class of primary inputs to delete with the -Class switch.

- -Class User | System | Full

  An optional switch and literal pair that specifies the class code of the designated primary input pins. The valid class code literal names are as follows:

  User — A literal specifying that you added the primary inputs using the add_primary_inputs command. This is the default class.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

System — A literal specifying that the primary inputs derive from the netlist.

Full — A literal specifying that the primary inputs consist of both user and system classes.

Note

In dft -scan context and dft -test_points context, the delete_primary_input -class system or -class full are not permitted.

Examples

The following example deletes an extra added primary input from the user class of primary inputs:

```
add_primary_inputs indata2 indata4 indata6
delete_primary_inputs indata4 -class user
```

Related Topics

add_primary_inputs

report_primary_outputs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_primary_outputs

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup

Removes the specified primary outputs from the existing flat model or, if a flat model does not yet exist, from the flattened result.

Usage

delete_primary_outputs {***net_pathname***... | ***primary_output_pin***... | **-All**}
[-Class {User | System | Full}]

Description

The delete_primary_outputs command deletes from a circuit the primary outputs that you specify. You can delete either the user class, system class, or full classes of primary outputs. If you do not specify a class, the tool deletes the primary outputs from the user class.

You can display a list of any class of primary outputs by using the report_primary_outputs command.

Note

Once you delete a primary output pin with this command, adding the pin back again with the add_primary_outputs command does not create the same pin. Bus information is lost and the pin is marked as user-added. Take care to use delete_primary_outputs only if you want to truly delete a pin from the design so it is not used.

Arguments

- ***net_pathname***

  A repeatable string that specifies the circuit connections that you want to delete. You can specify the class of primary outputs to delete with the -Class switch.

- ***primary_output_pin***

  A repeatable string that specifies a list of primary output pins that you want to delete. You can specify the class of primary outputs to delete with the -Class switch.

- **-All**

  A switch that deletes all primary outputs. You can specify the class of primary outputs to delete with the -Class switch.

- -Class User | System | Full

  An optional switch and literal pair that specifies the class code of the primary output pins that you specify. The valid literal names are as follows:

  User — A literal specifying that the list of primary outputs were added using the add_primary_outputs command. This is the default class.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

System — A literal specifying that the list of primary outputs derive from the netlist.

Full — A literal specifying that the list of primary outputs consists of both the user and system class.

Examples

The following example deletes a primary output from the system class of primary outputs:

```
delete_primary_outputs outdata1 -class system
```

Related Topics

add_primary_outputs

report_primary_outputs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_processors

Context: unspecified, dft -edt, patterns -scan

Mode: setup, analysis, insertion (dft -edt context only)

Removes processor definitions previously specified with the add_processors command.

Usage

delete_processors [*hostname*[:*cpus*]]…

Description

Removes processor definitions previously specified with the add_processors command.

The delete_processors command deletes secondary processor definitions you previously specified using the add_processors command. When you issue the command with no arguments, the tool deletes any additional local (primary) threads and discontinues all current secondary processes. If you specify one or more *hostname* arguments, the tool discontinues secondary processes only on the specified host machines. If you include :*cpus* with *hostname*, the tool discontinues only the specified number of secondary processes on that host machine. Without the :*cpus*, all current secondary processes on that host machine are discontinued. When specifying *hostname*, you must specify the network name of the machine, even if it was originally selected by an LSF, SGE, or custom job scheduler.

Be aware that the process run by an LSF, SGE, or custom job scheduler is a script rather than the actual secondary process. If you discontinue these secondary processes with the delete_processors command, the corresponding scripts also terminate.

Tip

Alternatively, you can end execution of the scripts by using the job scheduler commands (SGE's **qdel**, LSF's **bkill**, or whatever is used by the job scheduler invoked by the add_processors generic command), or a Linux "kill" command. There is a small delay before the secondary process is actually shut down to permit for the detection of the script termination and for the shutdown to be coordinated with the primary process. Secondary processes shut down in this manner appear as having terminated in the primary process's transcript. This is normal behavior. Detection of script terminations occurs only during the add_processors, delete_processors, and report_processors commands; you can also see these terminations when actively using the processes. You can force detection from the command line without other side effects by issuing the report_processors command with no arguments.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- *hostname*

  A repeatable string that specifies a host machine you previously defined with the add_processors command or received via a job scheduler. The hostname choices are as follows:

  machine_name — A string that specifies the host machine by its network name.

  LOCALHOST — A literal that specifies the host machine is the machine on which you invoked the tool.

  IPaddress — A string that specifies the host machine by its IP address.

- : *cpus*

  An optional colon (:) and positive integer pair that specifies the number of processes to delete on the particular host machine.

Examples

The following example begins by reporting the distributed ATPG status. It then discontinues all the secondary processes currently running on one host (nemo) and one of the secondary processes running on the other host (ahab):

report_processors

```
hosts/processors   arch      CPU(s)     %idle    free RAM    process size
----------------  -------  -----------  -----  ------------  ------------
ace   (primary)   x86      2 x 2.2 GHz    81%    8393.52 MB      20.07 MB
nemo           1  x86      2 x 2.0 GHz    98%     382.33 MB      18.78 MB
               2                                                 18.78 MB
               3                                                 18.78 MB
               4                                                 18.78 MB
ahab           1  x86      1 x 1.7 GHz    99%     334.98 MB      18.78 MB
               2                                                 18.78 MB
Primary process with 6 threads running.
```

delete_processors nemo ahab:1

```
//  Deleted host nemo:4
//  Deleted host ahab:1
//  Total processors: 1
//  Note: Freeing 1 'Tessent FastScan' license.
```

report_processors

```
hosts/processors   arch      CPU(s)     %idle    free RAM    process size
----------------  -------  -----------  -----  ------------  ------------
ace   (primary)   x86      2 x 2.2 GHz    81%    8393.52 MB      20.07 MB
ahab           1  x86      1 x 1.7 GHz    97%     229.83 MB      18.78 MB
Primary process with 1 thread running.
```

Related Topics

add_processors

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

compress_patterns

create_patterns

identify_redundant_faults

order_patterns

report_processors

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_read_controls

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup

Removes the read control line definitions from the specified primary input pins.

Usage

delete_read_controls ***primary_input_pin***... | **-All**

Description

The delete_read_controls command deletes read control lines defined with the add_read_controls command. You can delete the read control line definitions for specific pins or for all pins.

Arguments

- ***primary_input_pin***

  A repeatable string that specifies a list of primary input pins from which you want to delete any read control line definitions.

- **-All**

  A switch that deletes the read control line definitions for all primary input pins.

Examples

The following example deletes an incorrect read control line, then redefines that read control line with the correct off-state:

```
add_read_controls 0 r1 r2
delete_read_controls r1
add_read_controls 1 r1
set_system_mode analysis
```

Related Topics

add_read_controls

report_read_controls

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_register_value

Context: all contexts

Mode: setup

Deletes all or a specified register value variable specified with the add_register_value command.

Usage

delete_register_value {**-All** | ***value_name*** ...}

Description

Deletes all or a specified register value variable specified with the add_register_value command.

Refer to "Serial Register Load and Unload for LogicBIST and ATPG" in the *Tessent Shell User's Manual* for complete information.

Arguments

- **-All**

  A required switch that deletes all register value variables.

- ***value_name*** ...

  A required repeatable string that deletes a named ***value_name***.

Examples

The following example deletes all register value variables:

```
delete_register_value -all
```

Related Topics

add_register_value

report_register_value

set_register_value

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_rtl_to_gates_mapping

Context: all contexts

Mode: all modes

Removes name-mapping rules for RTL to post-synthesis/post-layout name mapping.

Usage

delete_rtl_to_gates_mapping [-instance_name_map_index *instance_name_map_index*] |
[-register_name_modifiers_index *register_name_modifiers_index*] |
[-port_name_map_index *port_name_map_index*] |
[-port_name_substring_map_index *port_name_substring_map_index*]

Description

Removes the name-mapping rules for RTL to post-synthesis/post-layout name mapping built into the tool or that you previously added using the add_rtl_to_gates_mapping command. Use the report_rtl_to_gates_mapping command to see the integer index value associated with each mapping.

Arguments

- **-instance_name_map_index *instance_name_map_index***

  An optional switch and string pair that specifies the instance renaming rule index to remove.

- **-register_name_modifiers_index *register_name_modifiers_index***

  An optional switch and string pair that specifies a mapping previously added by the add_rtl_to_gates_mapping -register_name_prefix or -register_name_suffix options to remove.

- **-port_name_map_index *port_name_map_index***

  An optional switch and string pair that specifies a mapping previously added by the add_rtl_to_gates_mapping -port_name_map_list option to remove.

- **-port_name_substring_map_index *port_name_substring_map_index***

  An optional switch and string pair that specifies a mapping previously added by the add_rtl_to_gates_mapping -port_name_substring_map_list option to remove.

Examples

The following example shows the result of report_rtl_to_gates_mapping before and after the delete_rtl_to_gates_mapping command was called. The register name prefix "rtlcreg_" which is used to create the built in mapping rule "rtlcreg_%name{[%d]}{.%name{[%d]}}" and the user-specified instance mapping “corea_i1/coreb_i1 -> corea_i1” rules are deleted by using their respective indexes from the report created by the report_rtl_to_gates_mapping command.

report_rtl_to_gates_mapping

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Index  Register Name Prefix  Register Name Suffix
   -----  --------------------  --------------------
    1                                          _reg
    2                rtlcreg_
    3
// Index Original instance name Mapped instance name
// ----- ---------------------- --------------------
// 1     corea_i1/coreb_i1      corea_i1
```

delete_rtl_to_gates_mapping -instance_name_map_index 1

delete_rtl_to_gates_mapping -register_name_modifiers_index 2

report_rtl_to_gates_mapping

```
// Index  Register Name Prefix  Register Name Suffix
   ----- ---------------------  --------------------
    1                                          _reg
    2
```

Related Topics

add_rtl_to_gates_mapping

report_rtl_to_gates_mapping

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_scan_chain_families

Context: dft -scan

Mode: analysis

Removes scan chain families.

Usage

delete_scan_chain_families {***obj_spec*** | **-all**} [-force]

Description

Use this command to remove scan chain families.

Arguments

- ***obj_spec***

  An required list that specifies a collection of scan chain families that should be deleted.

- **-all**

  An required switch that specifies that all scan chain families that have been added so far should be deleted.

Note

If included in one or more scan mode, you must first delete these modes using the "delete_scan_modes" command.

- -force

  An optional switch that deletes associated scan modes along with the specified scan chain families.

Related Topics

get_scan_chain_families

create_scan_chain_family

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_scan_chains

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup

Removes the specified scan chain definitions from the scan chain list.

Usage

delete_scan_chains ***chain_name***... | **-All**

Description

The delete_scan_chains command deletes scan chains defined with the add_scan_chains command. You can delete the definitions of specific scan chains or of all scan chains.

dft -scan Details

When you remove a scan chain definition, it is only the definition you are removing, not the scan chain itself.

Arguments

- ***chain_name***

  A repeatable string that specifies the names of the scan chain definitions that you want to delete.

- **-All**

  A switch that deletes all scan chain definitions.

Examples

The following example defines several scan chains, adding them to the scan chain list, then deletes one of the scan chains:

```
add_scan_chains chain1 group1 indata2 outdata4
add_scan_chains chain2 group1 indata3 outdata5
add_scan_chains chain3 group1 indata4 outdata6
delete_scan_chains chain2
```

Related Topics

add_scan_chains

report_scan_chains

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_scan_elements

Context: dft -scan, dft -test_points, dft -no_rtl

Mode: setup, analysis

Removes scan elements (such as chain or segment) of a given name.

Usage

delete_scan_elements {***obj_spec*** | **-all** }

Description

Use this command to remove scan elements (such as chain or segment) of a given name.

Arguments

- ***obj_spec***

  A required list that specifies a collection of scan elements that should be deleted.

- **-all**

  A required switch that specifies that all scan elements that have been added so far should be deleted.

Note

Only scan elements created by you in the current system mode can be deleted.

Related Topics

add_scan_segments

report_scan_segments

get_scan_elements

report_scan_elements

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_scan_groups

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup

Removes the specified scan chain group definitions from the scan chain group list.

Usage

delete_scan_groups ***group_name***... | **-All**

Description

The delete_scan_groups command deletes scan chain groups defined with the add_scan_groups command. You can delete the definitions of specific scan chain groups or of all scan chain groups.

When you delete a scan chain group, the tool also deletes all scan chains within the group.

Arguments

- ***group_name***

  A repeatable string that specifies the names of the scan chain group definitions that you want to delete.

- **-All**

  A switch that deletes all the scan chain group definitions.

Examples

The following example defines two scan chain groups, adding them to the scan chain group list, then deletes one of the scan chain groups:

```
add_scan_groups group1 scanfile1
add_scan_groups group2 scanfile2
delete_scan_groups group1
```

Related Topics

add_scan_groups

report_scan_groups

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_scan_instances

Context: dft -scan, dft -test_points

Mode: setup

Removes the specified, sequential instances from the user-identified scan instance list.

Usage

delete_scan_instances -INStances *instance_spec* | -Modules *module_spec* | -All | -library_models

Description

The delete_scan_instances command deletes sequential instances, library models or all instances within the specified module that were previously added to the scan instance list by using the add_scan_instances command. You can delete either a specific list of instance names or all instances.

You can also use this command to remove an exception status from an instance, that you added using the add_scan_instances command, which enabled the tool to convert the instance into a scan cell even though it violated the S4 rule. In this case, although the instance is converted back into a non-scan instance, the effects of its previous unknown state on downstream instances is not reset.

User-identified scan instances result from using the add_scan_instances command.

If you issue an insert_test_logic command after removing an instance from the user-identified scan list with the delete_scan_instances command, the tool then has the option of including it in the system-identified scan instance list.

Arguments

- -INStances *instance_spec*

  An optional switch and value pair that causes to tool to delete the specified instances from the user-identified scan instance list. *instance_spec* can be a Tcl list of one or more instances or a collection of one or more instances. If the instance is hierarchical, then all instances beneath it are also deleted from the user-identified scan instance list.

- -Modules *module_spec*

  An optional switch and value pair that causes to tool to delete all instances within the specified modules from the user-identified scan instance list. *module_spec* can be a Tcl list of one or more modules or a collection of one or more modules. Note: module may specify the name of a Verilog module, or a Tessent library model.

- -All

  An optional switch that specifies to delete all instances from the user-identified scan instance list. This switch does not affect the instances in the system-identified scan instance list.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -library_models

  An optional switch that causes the tool to delete all nonscan_instances that were marked as such by using the -module switch followed by a Tessent library model name. This switch serves as a replacement for the obsolete “delete_(non)scan_models -all” command.

Examples

The following example deletes an extra sequential scan instance that was defined to be treated as a scan cell; thus, the deleted instance is no longer included in the user-identified scan instance list:

```
add_scan_instances -instances i_1006 i_1007 i_1008
delete_scan_instances i_1007
```

Related Topics

add_scan_instances

report_seq_transparent_procedures

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_scan_modes

Context: dft -scan

Mode: analysis

Removes one or more scan modes.

Usage

delete_scan_modes {***obj_spec*** | **-all**}

Description

Use this command to remove one or more chain scan modes.

Arguments

- ***obj_spec***

  A required list of scan modes that should be deleted.

- **-all**

  A required switch that specifies that all scan modes that have been added so far should be deleted.

Related Topics

add_scan_mode

get_scan_modes

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_schematic_callout

Context: all contexts

Mode: all modes

Deletes informational callouts from a Tessent Visualizer schematic window.

Usage

delete_schematic_callout ***object_spec*** [-display *window_name*]

Description

Deletes informational callouts from objects in the Tessent Visualizer schematic window.

Arguments

- ***object_spec***

  A Tcl list or Tcl collection of one or more names of pin, port, pseudo_port, gate_pin, instance, or net objects.

  Tip

  If your ***object_spec*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- -display *window_name*

  An optional switch and literal that specifies the Tessent Visualizer schematic window from which to delete the callout. The valid options for the *window_name* argument are as follows:

  flat_schematic — A literal that specifies the Flat Schematic window.

  hierarchical_schematic — A literal that specifies the Hierarchical Schematic window.

  ijtag_network — A literal that specifies the IJTAG Network window.

  icl_schematic — A literal that specifies the ICL Schematic window.

  By default, the callout is deleted from the Flat Schematic window.

Examples

This example shows how to add a callout with the message "this is an instance" to a gate and another callout with the message "this is a pin" to a pin in the Hierarchical Schematic window. After these are added, the callout for the pin is deleted.

```
add_schematic_callout U_0/CSEEblock/rootcntr_reg[1] U_0/CSEEblock/rootcntr_reg[1]/Q \
-message "this is an instance" -display hierarchical_schematic

add_schematic_callout U_0/CSEEblock/rootcntr_reg[1]/Q -m "this is a pin" \
-display hierarchical_schematic

delete_schematic_callout U_0/CSEEblock/rootcntr_reg[1]/Q -display hierarchical_schematic
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_schematic_objects

Context: all contexts

Modes: all modes

Deletes design objects from a Tessent Visualizer schematic window.

Usage

delete_schematic_objects [***object_spec*** | ***-all***] [-display *window_name*]

Description

Delete objects from the specified schematic window.

Arguments

- ***object_spec***

  A Tcl list or Tcl collection of one or more names of pin, port, pseudo_port, gate_pin, instance, or net objects.

  Tip

  If your ***object_spec*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- ***-all***

  Remove all objects from the schematic window. Mutually exclusive with ***object_spec***

- -display *window_name*

  An optional switch and literal that specifies the Tessent Visualizer schematic window from which to delete the objects. The valid options for the *window_name* argument are as follows:

  flat_schematic — A literal that specifies the Flat Schematic window.

  hierarchical_schematic — A literal that specifies the Hierarchical Schematic window.

  ijtag_network — A literal that specifies the IJTAG Network window.

  icl_schematic — A literal that specifies the ICL Schematic window.

  By default, the objects are deleted from the Flat Schematic window.

Examples

Example 1

This example shows how to delete a collection of objects from the Hierarchical Schematic window:

```
delete_schematic_objects { bsr_i1 bsr_i2 } -display hierarchical_schematic
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

This example shows how to delete all objects from the Flat Schematic window:

```
delete_schematic_objects -all
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_schematic_path

Context: all contexts

Mode: all modes

Delete the specified path from a Tessent Visualizer schematic window.

Usage

delete_schematic_path ***object_spec*** [-display *window_name*]

Description

Deletes the specified path from the Tessent Visualizer Flat or Hierarchical Schematic window.

Arguments

- ***object_spec***

  A Tcl list or Tcl collection of one or more names of pin or gate_pin objects. Given lists of pins must represent existing connections in the hierarchical or flat data model.

  Tip

  If your ***object_spec*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- -display *window_name*

  An optional switch and literal that specifies the Tessent Visualizer schematic window from which to delete the path. The valid options for the *window_name* argument are as follows:

  flat_schematic — A literal that specifies the Flat Schematic window.

  hierarchical_schematic — A literal that specifies the Hierarchical Schematic window.

  ijtag_network — A literal that specifies the IJTAG Network window.

  icl_schematic — A literal that specifies the ICL Schematic window.

  By default, the path is deleted from the Flat Schematic window.

Examples

This example shows how to delete a path from the Hierarchical Schematic window:

```
delete_schematic_path { U_0/CSEEblock/rootcntr_reg[1]/Q U_0/CSEEblock/U50/A0 } \
-display hierarchical_window
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_simdut_fault

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Removes the injected stuck-at fault at the specified signal or all the injected faults.

Usage

delete_simdut_fault {**-signal** *signal_name* | **-all**}

Description

Removes the injected stuck-at fault at the specified signal or all the injected faults.

Arguments

- **-signal** *signal_name* | **- all**

  A required choice, as follows:

  -signal *signal_name*: deletes the specified injected stuck-at fault at the specified signal.

  -all: deletes all of the injected faults stored in the simulator.

Examples

The following example removes an injected stuck-at fault from the SIMDUT_TB.DUT_inst.memory00.q_a signal.

```
delete_simdut_fault -signal SIMDUT_TB.DUT_inst.memory00.q_a
```

Related Topics

add_simdut_fault

report_simdut_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_simulation_contexts

Context: all contexts

Mode: setup, analysis

Prerequisite: A flat model must currently exist

Deletes one or more user-defined simulation contexts.

Usage

delete_simulation_contexts {***context_name***... | **-all_user_defined**}

Description

Deletes one or more user-defined simulation contexts.

The delete_simulation_contexts command enables you to delete one or multiple or all user-defined simulation contexts.

Arguments

- ***context_name***

  A required string that defines the name of one or more simulation contexts. Note that you cannot use wildcard characters in the ***context_name***. You also cannot delete any of the predefined context names listed in Table 8-6 on page 2434.

- **-all_user_defined**

  A required switch that specifies all user-defined simulation contexts.

Examples

Example 1

The following example lists all of the user-defined simulation contexts and then deletes them:

```
SETUP> get_simulation_context_list  -user_defined {myContext1 myContext2 myContext3}

SETUP> delete_simulation_contexts -all_user_defined
SETUP> get_simulation_context_list -user_defined { }
```

Example 2

The following example lists all of the user-defined simulation contexts and then deletes two of them:

```
SETUP> get_simulation_context_list  -user_defined {myContext1 myContext2 myContext3}

SETUP> delete_simulation_contexts { myContext2 myContext3 }
SETUP> get_simulation_context_list -user_defined { myContext1 }
```

Related Topics

add_simulation_context

copy_simulation_context

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_simulation_forces

get_current_simulation_context

get_simulation_value_list

report_simulation_contexts

report_simulation_forces

set_current_simulation_context

simulate_clock_pulses

simulate_forces

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_simulation_forces

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist.

Removes forces from one or more gate_pin objects.

Usage

delete_simulation_forces {***obj_spec*** | **-all**}

Description

Removes forces from one or more gate_pin objects.

The delete_simulation_forces command removes simulation forces from the specified gate_pin objects. When this command removes a simulation force, the tool immediately propagates the values on the driving gate_pin objects to keep the circuit in a consistent state.

This command is permitted for user-defined contexts only.

Arguments

- ***obj_spec***

  A required value that specifies one or more gate_pin objects.

- **-all**

  A required switch that removes all simulation forces on all gate_pin objects within the current simulation context.

Examples

The following example shows the effect of adding a force and then removing all forces:

```
ANALYSIS> set_current_simulation_context myContext
ANALYSIS> add_simulation_forces [get_gate_pins I1] -value 1
ANALYSIS> report_simulation_forces

Value   Object
------  ------
1       I1

ANALYSIS> delete_simulation_forces -all
ANALYSIS> report_simulation_forces
```

Related Topics

add_simulation_context

add_simulation_forces

copy_simulation_context

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_simulation_contexts

get_simulation_context_list

get_simulation_value_list

report_simulation_contexts

report_simulation_forces

simulate_clock_pulses

simulate_forces

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_synchronous_clock_group

Context: dft, patterns

Mode: setup

Removes one or more user-defined synchronous clock groups.

Usage

delete_synchronous_clock_group { ***object_spec*** | **-all** } [-scope {capture | shift | all}]

Note

Synchronous clock groups have no impact on scan chain analysis, test point identification, and EDT IP insertion (in the dft -scan, dft -test_points and dft -edt sub-contexts).

Description

Removes one or more user-defined synchronous clock groups.

The delete_synchronous_clock_group command removes clock groups that you defined with the add_synchronous_clock_group command. For more information about synchronous clock groups, refer to the add_synchronous_clock_group command.

Arguments

- ***object_spec***

  A Tcl list or collection of port or pseudo_port objects that specify a current synchronous clock group. You must specify the complete original object_spec for the clock group you want to remove.

- **-all**

  A switch that specifies all synchronous clock groups.

- -scope {capture | shift | all}

  An optional switch and literal pair that specifies whether to delete capture clock groups, shift clock groups, or both. By default, capture clock groups are deleted.

Examples

The following example creates two synchronous clock groups and then removes one of them:

```
SETUP> report_clocks
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
User-defined Clocks (4):
========================

  Sync and Async Source Clocks
  ============================
   ----  ---------  -----------  --------
   Name  Off State  Constraints  Internal
   ----  ---------  -----------  --------
    I1       0                      No
    I2       0                      No
    I3       0                      No
    I4       0                      No
```

SETUP> add_synchronous_clock_group {I1 I2}
SETUP> report_synchronous_clock_groups

```
User-defined Synchronous Clock Group
====================================
//  No.  Clocks (Labels)   Scope
//  ---  ---------------  -------
//   1   I1               capture
//       I2
```

SETUP> add_synchronous_clock_group {I3 I4}
SETUP> report_synchronous_clock_groups

```
User-defined Synchronous Clock Groups
=====================================
//  No.  Clocks (Labels)   Scope
//  ---  ---------------  -------
//   1   I1               capture
//       I2
//   2   I3               capture
//       I4
```

SETUP> delete_synchronous_clock_group {I1}

```
//  Error: Specified synchronous clock group does not exist.
```

SETUP> delete_synchronous_clock_group {I2 I1}
SETUP> report_synchronous_clock_groups

```
User-defined Synchronous Clock Group
====================================
//  No.  Clocks (Labels)   Scope
//  ---  ---------------  -------
//   1   I3               capture
//       I4
```

The following example uses the same clocks to create a synchronous clock group for both capture and shift and then removes the capture group:

SETUP> delete_synchronous_clock_group -all -scope all
SETUP> add_synchronous_clock_group -scope all {I1 I2}
SETUP> report_synchronous_clock_groups

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
User-defined Synchronous Clock Group
=====================================
//  No.  Clocks (Labels)   Scope
//  ---  ---------------  -------
//   1   I1               capture
//       I2
//  ---  ---------------  -------
//   1   I1                shift
//       I2
```

SETUP> delete_synchronous_clock_group -all

```
//  All user-defined synchronous clock groups in the scope of capture are deleted.
```

SETUP> report_synchronous_clock_groups

```
User-defined Synchronous Clock Group
=====================================
//  No.  Clocks (Labels)  Scope
//  ---  ---------------  -----
//   1   I1               shift
//       I2
```

Related Topics

add_clocks

add_synchronous_clock_group

get_synchronous_clock_groups

report_clocks

report_synchronous_clock_groups

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_test_points

Context: dft -test_points

Mode: setup, analysis

Removes the specified test point definitions.

Usage

delete_test_points {*object_spec* | -location *pin/port_spec* | -all} [-silent]

Description

You can use this command to delete all test points, specific test points from a test_point object spec as returned from the get_test_points command, or specific test points according to their location. This command removes both user-defined and system-defined test points.

Arguments

- *object_spec*

  The value of object_spec can be a control point or observe point or a Tcl list or Tcl collection of one or more test points as returned from the get_test_points command.

- -location *pin/port_spec*

  A switch and string pair that specifies a pathname to the test points to delete. *pin/port spec* can be a Tcl list of one or more pins/ports, or a collection of one or more pin/port objects.

- -all

  A switch that removes the control and observe test points from all locations.

- -silent

  An optional switch that suppresses the error message normally generated when the command does not result in the deletion of any test points. The error message is intended to indicate a possible mistake when specifying patterns. This -silent option has no effect on any other error reporting.

  Note

  You can use the -silent switch to manage the case in which no test_points are deleted. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

Examples

Example 1

The following example removes specified test points:

```
delete_test_points -location {/I_6_16/cp /I_7_16/q}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The delete_test_points command only specifies the pathnames of the test points, not the type. This example includes both control and observe test points and deletes them by default.

Example 2

The following example removes all observe points returned by the get_test_point_type command:

```
delete_test_points [get_test_points -type OP]
```

Related Topics

add_input_constraints

add_tied_signals

analyze_test_points

insert_test_logic

report_test_logic

report_test_points

set_test_point_insertion_options

set_test_point_analysis_options

set_test_point_types

write_test_point_dofile

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_tied_signals

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup

Removes the assigned (tied) value from the specified floating nets or pins.

Usage

delete_tied_signals {***floating_object_name***... | **-All**} [-Pin]

Description

The delete_tied_signals command deletes the tied values assigned with the add_tied_signals command. You can display a list of tied signals using the report_tied_signals command.

Whenever you delete tied values from nets or pins, be sure to re-add any necessary values before performing another simulation. If you do not add required tied values to floating nets or pins, the tool displays a warning. The warning states that the design has floating nets or pins and assumes they are tied to the default value; you must set the default value using the set_tied_signals command.

Arguments

- ***floating_object_name***

  A repeatable string that specifies the names of the tied floating nets or pins whose tied values you want to delete.

  If you do not specify the -Pin option, the tool assumes ***floating_object_name*** is a net name. If you specify the -Pin option, it assumes the ***floating_object_name*** is a pin name.

- **-All**

  A switch that deletes the tied values from all tied floating nets or pins.

- -Pin

  A switch specifying that the ***floating_object_name*** argument that you provide is a floating pin name.

Examples

The following example deletes the tied value from the user-class tied net “vcc”; thereby leaving “vcc” as a floating net:

```
add_tied_signals 1 vcc vdd
delete_tied_signals vcc
```

Related Topics

add_tied_signals

report_tied_signals

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_tied_signals

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_wave_generator_pins

Context: all

Mode: all

Deletes provided pin objects, or all objects, from the Tessent Visualizer Wave Generator tab.

Usage

delete_wave_generator_pins *object_spec* | -all

Description

A flat model must be available to use this command.

Arguments

- *object_spec*

  A Tcl list or Tcl collection of one or more names of pin, port, pseudo_port, or gate_pin objects.

- -all

  An optional switch that specifies that all pins are deleted from the Wave Generator tab.

Examples

```
delete_wave_generator_pins {inst/clk inst/scan_en}
```

```
delete_wave_generator_pins -all
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_write_controls

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup

Removes the write control line definitions from the specified primary input pins.

Usage

delete_write_controls ***primary_input_pin***... | **-All**

Description

The delete_write_controls command deletes write control lines defined with the add_write_controls command. You can delete the write control line definitions for specific pins or for all pins.

Arguments

- ***primary_input_pin***

  A repeatable string that specifies a list of primary input pins from which you want to delete any write control line definitions.

- **-All**

  A switch that deletes the write control line definitions for all primary input pins.

Examples

The following example deletes an incorrect write control line, then re-adds that write control line with the correct off-state:

```
add_write_controls 0 w1 w2
delete_write_controls w1
add_write_controls 1 w1
set_system_mode analysis
```

Related Topics

add_write_controls

report_write_controls

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

diagnose_failures

Context: patterns -scan_diagnosis

Mode: analysis

Prerequisites: The read_patterns command must be set to a pattern source.

Diagnoses the failures in the specified failure file.

Usage

diagnose_failures {***failure_filename*** | **-Internal**} [-Output *report_filepath* | > *pipe_to_filename*] [-Csv *csv_filename*] [-Replace] [-Faulty_chain {*faulty_chain_name faulty_chain_type*}...]

Description

The diagnose_failures command performs a diagnosis of the failing scan and chain test patterns in the specified failure file, or diagnosis of the IDDQ patterns in the specified failure file.

For scan and chain test patterns, you must translate the ATE failure log into a failure file as described in the "Guidelines for Preparing the ATE Failure File" section in the *Tessent Diagnosis User's Manual*. For IDDQ patterns, refer to "IDDQ Diagnosis" for the required failure file format.

Note

The tool can automatically detect invalid fail file annotations that may cause it to treat missing fails as passes, such as a misplaced "failure_buffer_limit_reached none" annotation. See "Failure Truncation Handling" in the *Tessent Diagnosis User's Manual* for more about annotations. This feature only supports pattern-based fail files and stuck-at faults on a single scan chain for designs without SSN. The tool reports a warning that is similar to the following when it detects an annotation problem:

```
Warning: Incorrect failure truncation annotation or defect on the input of
faulty chain: '<chain_name>' is highly probable. For accurate diagnosis
please make sure failure truncation annotation in failure logfile
'<faillog_name>' is correct.
```

It is critical that the data in the test patterns, flattened netlist, and failure file be synchronized for diagnosis. For more information, refer to "Input File Requirements" in the *Tessent Diagnosis User's Manual*.

For complete information on running diagnosis and interpreting the results, refer to the *Tessent Diagnosis User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The following warning message displays when the flat model contains some timing exceptions and the static timing exception analysis data is not available in the flat model and needs to be re-analyzed:

```
// Warning: The flat model does not include static timing exception
analysis data because it was saved before the create_patterns command.
This can result in longer diagnosis run times. Suggest using the flat
model that includes static timing exceptions analysis to improve diagnosis
time.
```

Arguments

- ***failure_filename***

  A required string that specifies the name of the file that contains the failing test pattern information for diagnosis. This file can use either the pattern-based or cycle-based format as described in the "Guidelines for Preparing the ATE Failure File" section in the *Tessent Diagnosis User's Manual*.

  For IDDQ diagnosis, specify the IDDQ failure file. Refer to "IDDQ Diagnosis" for more information.

  Note

  The filename for a core-level failure log generated during reverse mapping (refer to "Reverse Mapping Top-Level Failures to the Core" in the *Tessent Diagnosis User's Manual* for details) contains escaped characters if the core instance name contains escapes. In this case, the *failure_file* string requires additional escape characters. For example, you must specify *failure_file* for an escaped log file named *test.flog__c2_1\[3\]__core2__internal* as:

  ```
  read_failures {test.flog__c2_1\\\[3\\]__core2__internal}
  ```

- **-Internal**

  An required switch that directs the tool to use the in-memory failure file. You must issue a read_failures command before you use the diagnose_failures -internal command. Refer to "Diagnosis Improvements and Retrieving Internal Scan Cell Information" in the *Tessent Diagnosis User's Manual* for complete information.

- -Output *report_filepath* | > *pipe_to_filename*

  An optional choice that directs the tool to write the diagnostic report in ASCII format to the specified file. By default, the diagnostic report displays on screen.

  -Output *report_filepath:* An optional switch and string pair that specifies the output file. This option adds the physical layout information to the report. The physical layout information is not included in the transcript.

  > *pipe_to_filename*: An optional redirection operator and string pair that specifies the output file. Choose this option when you have specified set_diagnosis_options -internal_failing_cells on. The internal failing cell data is only reported to the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

transcript unless you redirect it to a file; however, the -output switch does not include the internal failing cell data in its report. Refer to "Internal Scan Cell Profiling for Compressed Patterns" for more information.

- -Csv *filename*

  An optional switch and string pair that directs the tool to write the diagnostic report in Comma Separated Value (CSV) format to the specified file. The CSV format does not include the list of failing patterns for each symptom. By default, the diagnostic report displays on screen.

- -Replace

  An optional switch that directs the tool to overwrite existing diagnostic report files in either ASCII or CSV format. By default, the tool does not overwrite existing files.

- -Faulty_chain *faulty_chain_name faulty_chain_type*

  An optional switch and repeatable string and literal pair that enables you to run diagnosis on specified chain failures in addition to the scan failures contained in a failure file.

  faulty_chain_name — string that specifies the name of the faulty chain.

  fault_type — literal that specifies the fault type associated with the faulty chain. Valid fault types include:

  STUCK_AT_1

  STUCK_AT_0

  SLOW_TO_FALL

  SLOW_TO_RISE

  SLOW

  FAST_TO_RISE

  FAST_TO_FALL

  FAST

  INDETERMINATE — specifies the fault type cannot be determined.

Examples

Example 1

The following example: sets *pattern_file1* as the external test pattern source, injects a stuck-at-0 fault into the design, writes out a failure file named fail_patterns, and performs a diagnosis of the failing patterns in the *fail_patterns* file.

```
set_system_mode analysis
read_patterns pattern_file1
write_failures fail_patterns i_1006/i1 -stuck_at 0

//  failing_patterns=9  simulated_patterns=36  simulation_time=0.00 sec

diagnose_failures fail_patterns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
...
```

Still using the external test pattern source file *pattern_file1*, the following example: injects an additional stuck-at-0 fault into the fail_patterns failure file, performs a diagnosis of the failing patterns in the *fail_patterns* file, and sends the report to a file named *fail_diags*.

```
write_failures fail_patterns i_1005/i1 -stuck_at 0 -replace

//  failing_patterns=4  simulated_patterns=36  simulation_time=0.00 sec

diagnose_failures fail_patterns -output fail_diags
```

Example 2

The following example loads patterns from two files into the tool external pattern database and performs two separate diagnoses using the default settings. The example writes the resulting reports to files instead of displaying them onscreen. The failure file, *scan_failure_file*, contains only a scan test section ("scan_test" keyword followed by scan test fails), so the tool performs only a scan diagnosis for that file. The second file, *chain_failure_file*, contains a chain test section ("chain_test" keyword followed by chain test fails) as well as a scan test section. The tool performs only a chain diagnosis for that file.

```
read_patterns pat_file1
read_patterns pat_file2 -append
diagnose_failures scan_failure_file -output scan_diag.ascii -csv scan_diag.csv
diagnose_failures chain_failure_file -output chain_diag.ascii -replace
```

Example 3

The following example first performs chain diagnosis on a failure file that contains a chain_test section. The example next attempts a diagnosis of a failure file that contains no chain_test data; however, because the set_diagnosis_options command is set to chain mode, the tool skips this diagnosis.

```
read_patterns pat_file1
set_diagnosis_options -mode chain
diagnose_failures chain_failure_file -output chain_diag.ascii -replace
diagnose_failures scan_only_failure_file -output scan_diag.ascii -csv scan_diag.csv

//  Error: Cannot identify faulty chains. Please either use chain test
//    patterns or specify faulty chains.
```

Example 4

The following example sets the external test pattern source to *pattern_file1*, runs diagnosis on the chain_foo chain failure and the failed patterns failure file.and runs diagnosis on the chain_foo chain failure and the *failed cycles* failure file.

```
set_system_mode analysis
read_patterns pattern_file1
diagnose_failures failed_cycles -faulty_chain chain_foo FAST_TO_FALL
```

Related Topics

display_diagnosis_report

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_diagnosis_options

set_fault_type

write_diagnosis

write_flat_model

write_failures

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

display_diagnosis_report

Context: patterns -scan_diagnosis

Mode: analysis

Prerequisites: This command can only operate on a diagnosis report file written with the diagnose_failures command with the -Output switch.

Opens a diagnosis report in which you can click symptoms and suspect locations to add related gates to the Tessent Visualizer Hierarchical Schematic window.

Usage

display_diagnosis_report ***diagnose_failures_output_filename***

Tessent Visualizer Menu Path: Open > Diagnosis Report Viewer

Description

The display_diagnosis_report command opens the specified diagnosis report file in the Diagnosis Report Viewer of Tessent Visualizer. In the displayed report, each symptom and all suspect locations (pin and net pathnames) show up as active links. Clicking a suspect's pin_pathname link adds the gate associated with that pin to the Tessent Visualizer Hierarchical Schematic window. Clicking a suspect's net_pathname link adds all the gates connected to that net. Clicking a symptom link adds all the gates with pins in that symptom's suspect list.

Refer to "Diagnosis Reporting" in the *Tessent Diagnosis User's Manual* for a detailed description of the types of information contained in the diagnosis report.

Arguments

- ***diagnose_failures_output_filename***

  A required string that specifies the pathname of a diagnosis report file written out using the diagnose_failures -Output command.

Examples

The following example performs scan diagnosis using the set of ATPG patterns in the pattern file *pat_file1* and the failure information in the failure file *scan_failure_file*, writing the diagnosis report in ASCII format to the file *scan_diag.ascii*. The example then opens Tessent Visualizer if it is not already open and opens the diagnosis report in the Diagnosis Report Viewer.

```
read_patterns pat_file1
diagnose_failures scan_failure_file -output scan_diag.ascii -replace
display_diagnosis_report scan_diag.ascii
```

Related Topics

diagnose_failures

set_diagnosis_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

display_message

Context: unspecified, all contexts

Mode: all modes

Sends a message to the specified message stream.

Usage

display_message ***string*** [-error | -warning | -info | -note | -screen_only] [-abort]

Description

The string can contain \n to break the message into multiple lines. The tool prefixes messages sent in the message stream with an "// Error: ", "// Warning:" , "//" , "//" or "//Note:" string as appropriate, and automatically indents successive lines remaining.

The output of Tcl commands such as "puts" redirects to the screen, to a log file, or both as with any other tool output.

Arguments

- ***string***

  A required string that specifies the message to be sent to the message stream. The string supports the use of the newline character sequence "\n" to break the message into multiple lines. The first character of the string must be something other than a dash ("-") character.

- -error

  An optional literal that specifies to send a message into the error stream. This message does not interrupt the execution of the script. You can issue multiple error messages and then terminate the script using a {return -code error "string"} command. This return code causes the dofile to abort unless you specify "set_tcl_shell_options -abort_dofile_on_error off" or the -abort switch to the display_message command.

- -warning

  An optional literal that specifies to display a warning.

- -info

  An optional literal that specifies to display an information note. The message has a double slash "//" prefix.

- -note

  An optional literal that specifies to display a note.

- -screen_only

  An optional Boolean switch that specifies that the message string is for the transcript window only.

  When specified, the string is not sent to the log file as it normally does and it suppresses the new line character at the end of the string. When using the -screen_only switch, you use

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

explicit “\n” characters to force the new string to appear on a new line. You can also use the “\r” character at the beginning of the string to bring back the cursor to the beginning of the line and overwrite the existing string on the screen. Using the “\r” character is useful when you want to echo a progress status message on the screen. When you do that, make sure the new string is as long as the original string that was displayed on the line; otherwise, the string contains leftover characters from the previous string.

- -abort

  An optional switch that forces the tool to exit.

Examples

The following example issues two error messages and then stops the invocation of the current Tcl command using a Tcl return -code error command.

```
display_message “this value is wrong” -error
display_message “this other value is also wrong” -error
return -code error
```

Related Topics

register_tcl_command

unregister_tcl_command

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

display_specification

Context: dft

Mode: setup, analysis

Displays the DftSpecification(*design_name*,*id*) configuration data in a tab of the Config Data Browser window.

Usage

display_specification [*id*] [-design_name *design_name*] [-create]

Description

Displays the DftSpecification(*design_name*,*id*) configuration data in a tab of the Config Data Browser window.

You can create a DftSpecification from scratch in the Config Data Browser by using the -create option. You can also modify an existing DftSpecification.

When only one DftSpecification wrapper is defined in memory, you do not need to supply any arguments. The tool automatically displays the single DftSpecification wrapper.

You can define or modify a DftSpecification using the Config Data Browser window. This window provides a graphical user interface for inserting Tessent DFT IP into existing designs. For information on using the Config Data Browser, refer to "Config Data Browser" in the *Tessent Shell User's Manual*.

Arguments

- *id*

  An optional string that specifies the DftSpecification wrapper to be displayed. As described in the DftSpecification wrapper syntax, the wrapper is identified by the two strings design_name and id. If you have only one DftSpecification wrapper present in memory that matches "DftSpecification(design_name,*)", you can omit the id from the display_specification command. However, if several DftSpecification wrappers matching "DftSpecification(design_name,*)" exist in memory, you must specify the appropriate id when running the display_specification command. Refer to the description of the -design_name option to understand how the value of design_name is obtained.

- -design_name *design_name*

  An optional switch and string pair that specifies the design_name value used to identify the DftSpecification wrapper. The -design_name value defaults to the current design if it has been set with the set_current_design command. If the current design has not been set, it is extracted from the DftSpecification wrapper existing in memory.

  When the id is specified, if exactly one wrapper matches "DftSpecification (*,<id>)" in memory, the design_name is extracted from the design_name parameter of the DftSpecification wrapper. If more than one wrapper matches, the -design_name must be

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

specified unless the -create option is specified. When the id is not specified, if exactly one wrapper matches “DftSpecification (*,*)” in memory, the design_name is extracted from the design_name parameter of the DftSpecification wrapper. If more than one wrapper matches, the -design_name must be specified.

- -create

  An optional switch that disables the existence check of the DftSpecification wrapper and specifies to create and display an empty DftSpecification wrapper. If either of the id or -design_name options are not specified, their values are determined using the heuristic described in the id and -design_name options description. If the tool cannot determine the id, the default is “rtl” in the dft -rtl context and “gate” in the dft -no_rtl context. If the tool cannot determine the design_name, it sets it to “?” in the created file, which you can then modify in the Config Data Browser. If you do not change the “?” to a valid design name, the tool reports an error when you attempt to process the DftSpecification.

Examples

Example 1

The following example reads a DftSpecification from a file and displays it in the Config Data Browser.

```
set_context dft -rtl

read_config_data ./my_design.dft_spec_rtl

display_specification
```

Example 2

The following example creates an empty DftSpecification and displays it in the Config Data Browser, enabling you to add content using the graphical user interface. The created DftSpecification wrapper is named “DftSpecification(mydesign,rtl)”. The *design_name* is taken from the name of the current design. The *id* defaults to “rtl” because the command was called inside the dft -rtl context.

```
set_context dft -rtl

read_verilog mydesign.v

set_current_design mydesign

display_specification
```

Related Topics

process_dft_specification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dofile

Context: unspecified, all contexts

Mode: all modes

Runs the commands contained within the specified file.

Usage

dofile ***filename*** [-History]

Description

The dofile command sequentially runs the commands contained in a specified file. This command is especially useful when you must issue a series of commands. Rather than running each command separately, you can place them into a file in the required order and then run them by using the dofile command. You can also place comment lines in the file by starting the line with a double slash (//); the tool handles these lines as comments and ignores them.

The dofile command sends each command expression (in order) to the tool, which in turn displays each command line from the file before running it. If the tool encounters an error due to any command, the dofile command stops its execution, and displays an error message. You can enable the dofile command to continue regardless of errors using the “set_tcl_shell_options -abort_dofile_on_error off” command.

The dofile command is very similar to the Tcl source command except the dofile command has the following additional usability features:

- Loads relative paths called with dofile <RelativeFilePath> relative to the position of the dofile containing the dofile command.

  You must use the same syntax when using the source command as shown here: source [file dirname [info script]]/<RelativePathToFile>. When the source command is used, the file is processed purely by the Tcl shell.

- Accepts // as comments, as well as #.
- Automatically escapes [#] and [#:#] entries in bus names.

Arguments

- ***filename***

  A required string that specifies the name of the file that contains the commands that you want the tool to run.

- -History

  An optional switch that specifies for the tool to add the commands from a dofile to the command line history list. By default, the commands in a dofile are not inserted into the history list, but the dofile command itself is added to the list.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example runs, in order, all the commands from the file *command_file*:

dofile command_file

The command_file may contain any Tcl or application command available. An example of a command_file is as follows:

```
foreach_in_collection net [get_nets u1/*] {
   if {[sizeof_collection [get_fanins $net]] == 0} {
      delete_connections [get_fanouts $net -stop_on pin]
      delete_nets $net
   }
}
```

Related Topics

history

set_tcl_shell_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

echo

Context: unspecified, all contexts

Mode: all modes

Issues a user-defined string to the transcript.

Usage

echo ***"string"*** [{> | >>} *file_pathname*]

Description

Issues a user-defined string to the transcript.

The echo command issues a user-defined string to the transcript or to a pathname, if you use one of the file redirection operators.

Note

Commands that use either the > or >> file redirection operator are first checked for correctness. Syntax errors are reported to the display prior to the command's execution. The redirection operator does not hide these errors.

Arguments

- ***string***

  A required string. The string that you want echoed to the transcript. Double quotation marks are required if the string contains spaces or special characters.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example redirects output from several commands into a single output file, *my_scan_report*. The first command creates or replaces the *my_scan_report* file. The second and following commands append to the same file.

```
echo "----------- scan cells ------------" > my_scan_report
report_scan_cells >> my_scan_report
echo "----------- scan chains ----------" >> my_scan_report
report_scan_chains >> my_scan_report
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

get_transcript_style

no_transcript

set_logfile_handling

set_transcript_style

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

elaborate_icl_modules

Context: all contexts

Mode: all modes

Prerequisites: There must be ICL for the specified modules and for all of their descendants.

Elaborates the specified ICL modules.

Usage

elaborate_icl_modules ***obj_spec***

Description

For certain ICL tracing commands, like get_icl_fanins_in_module or get_icl_fanouts_in_module, and for the introspection of objects of certain types, like icl_pin_in_module, it is required to have elaborated ICL modules. Usually, the ICL elaboration happens when the current design is set or after ICL extraction has finished, but it can be triggered explicitly by means of the command elaborate_icl_modules. The ICL elaboration resolves parameters and expressions, creates internal representations of individual bits of ports and registers, remodels complex ICL building blocks (for example, OneHotDataGroup) and creates explicit connections for elements that are only implicitly connected. In order to resolve instance port bindings and hierarchical aliases, the command invokes the elaboration of the specified modules and of all of their descendants. Therefore, the ICL for all of those modules must have been read earlier by means of the read_icl command.

Arguments

- ***obj_spec***

  A required value that specifies a list of one or more ICL module names or a collection of one or more icl_module objects.

Examples

elaborate_icl_module {tap block1 block2}

Related Topics

get_icl_fanins_in_module

get_icl_fanouts_in_module

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

encrypt_layout

Context: patterns -scan_diagnosis

Mode: setup

Encrypts a layout database (LDB) for purposes of data transfer.

Usage

encrypt_layout ***ldb_path*** [-replace]

Description

Note

The encryption process is destructive in nature because once the names are removed, they cannot be returned to the LDB. Keep a golden original copy of the LDB for archival purposes.

This command is useful for transferring design and layout data to third parties (such as to foundries). The command removes net and instance names from the LDB and encrypts the remaining design and all of the layout data within the LDB name you specify. The only use should be for layout-aware diagnosis in the Tessent environment.

After encryption, Tessent YieldInsight flows and other flows that rely on layout names do not work. This is to be expected and is the recommended state for security. Other affected flows include (but are not limited to) those using commands that act on real design names (such as report_instance_polygon), commands that manipulate layout core information (such as add_layout_core_information), and the open_layout -verify command.

Layout-aware and chain diagnosis, and RCD/RCAD constant creation, function as normal. Within the LDB, layer and via names remain un-encrypted to enable design (for example, GDS-based) debugging and the RCD flow.

Note

Encryption based on compressed layout data is not supported. Before encrypting the LDB, uncompress the layout data by using the uncompress_layout command.

Arguments

- ldb_path

  A required string that specifies the pathname of the encrypted LDB. An error occurs if you attempt to name the encrypted LDB the same name as the source LDB.

- -replace

  An option that specifies to overwrite an existing encrypted LDB of the same name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example opens an existing LDB and encrypts it under a new name.

```
open_layout ../design.layout
encrypt_layout ../design_encrypted
```

Related Topics

open_layout

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

execute_cdp_test

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Runs the specified test using the currently connected tester.

Usage

execute_cdp_test ***test_name*** [-collect_data_type_list *data_type_list*]
[-force_diagnose_on_failure][-save_waveforms *file_name*]

Description

By default, this command returns pass/fail (go-nogo) results for the associated test procedure. When you specify -collect_data_type_list option, the tool collects the failure data during test execution and returns the data in a tabulated format. The tool provides the failing data with respect to the ReadVar variables originally specified in the PDL that you used to generate the pattern.

This command returns a “0” if the test passes, and a “1” if it fails, which enables you to create scripts leveraging the status.

When you are using SimDUT as your tester, the tool converts the test pattern file to simulation events, and then the Verilog simulator returns the failing information to SimDUT.

Arguments

- ***test_name***

  A required string that specifies the name of the test you want to run.

- -collect_data_type_list *data_type_list*

  An optional switch and literal that specifies the type of failure data to return. The data type options are:

  gonogo — pass/fail results

  raw_fails — the number of failures only; returns 0 if the test passed.

  variable — failing variables/registers

  instrument — failing instruments

  diagnosis — diagnosis datalog for failing instruments

- -force_diagnose_on_failure

  An optional switch that forces automatic failure diagnosis if the specified test fails.

- -save_waveforms *file_name*

  An optional switch that saves the test waveforms of the output pins to the specified file name. This switch is not available when running SimDUT. It is available only in desktop and adaptor simulation mode.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

When you specify this switch, the tool draws the waveforms using text that flows down the page. It may help your on-screen viewing to slightly tilt your head to the left as you read the recorded waveform levels from top to bottom while also scrolling down to view the report.

You can use the -save_waveforms switch by itself to collect a waveform for the first failure only, but we recommend using it with the -collect_data_type_list switch to save waveforms for all data collected.

The tool saves waveforms using plain text characters in a three column format similar to the following example.

```
addr        expW(P8)    actW(P8)
101         _|          _|
102         X           |_
103         |_          _|  FAILED: cycle_count = 104, pin_index = 8
```

In the example, vector address 101 is a pass because the expected and actual level for pin index 8 (P8) are both low. Vector address 102 is a pass because the expected level is a don't care. Vector address 103 is a fail because the expected level is high but the actual level is low.

Examples

Example 1

The following example runs test1 on the DUT and returns the results for the collect_failures procedure for the test pattern associated with test1.

execute_cdp_test test1 -collect_data_type_list variable

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  command: execute_cdp_test test1 -collect_data_type_list variable
//  Test 'test1' failed.
//  Variable failures and unmapped failures of pattern
// 'DLV2_1S0_ETMemory_Full.svf' :
//
//  PatternSet  Variable                        Pin  Expected
//                                                     Actual
//  ----------  ----------------------------  ---  --------
//        main             BP2_WIR.IR_STATUS2  TDO        b1
//                                                        b0
//  ----------  ----------------------------  ---  --------
//        main        BP2_WIR.IR_STATUS1.DONE  TDO        b1
//                                                        b0
//  ----------  ----------------------------  ---  --------
//        main  BP2_WDR.WBP1_INST_POINTER_REG  TDO   b000000
//                                                   b001001
//  ----------  ----------------------------  ---  --------
//        main       BP2_WDR.WBP1_A_ADD_REG_Y  TDO     b0000
//                                                     b1111
//  ----------  ----------------------------  ---  --------
//        main       BP2_WDR.WBP1_A_ADD_REG_X  TDO     b0000
//                                                     b1000
//  ----------  ----------------------------  ---  --------
//        main       BP2_WDR.WBP1_GO_ID_REG10  TDO        b0
//                                                        b1
//  ----------  ----------------------------  ---  --------
```

Example 2

The following example runs MemoryBist_P1 on the DUT and returns the results for the collect_failures procedure for the test pattern associated with MemoryBist_P1.

execute_cdp_test MemoryBist_P1 -collect_data_type_list {variable diagnosis}

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  command: execute_cdp_test test1 -collect_data_type_list {variable diagnosis}
//  Test 'MemoryBist_P1' failed.
//  Variable failures and unmapped failures of pattern 'MemoryBist_P1.stil' :
//
//  PatternSet     Variable                                          Pin  Expected
//                                                                        Actual
//  -----------    ----------------------------------------------  ---  -------
//  run_time_prog  core_inst1_top_rtl_tessent_mbist...MBISTPG_GO(1)  tdo  b1
//                                                                        b0
//  ---------------------------------------------------------------------------------
//  run_time_prog  core_inst1_blockA_clka_i2_mem6_i...US_SHADOW_REG  tdo  b00
//                                                                        b11
//  ---------------------------------------------------------------------------------
//  run_time_prog  core_inst1_blockA_clka_i2_mem6_i...nst.GO_ID_REG  tdo  b000000000000
//                                                                        b000000000010
//  ---------------------------------------------------------------------------------
//  run_time_prog  core_inst1_blockA_clka_i1_mem6_i...US_SHADOW_REG  tdo  b00
//                                                                        b11
//  ---------------------------------------------------------------------------------
//  run_time_prog  core_inst1_blockA_clka_i1_mem6_i...nst.GO_ID_REG  tdo  b00000000000
//                                                                        b00000001000
//  ---------------------------------------------------------------------------------
//
// Diagnostic Result: CDPResult/TestExecutionResult/DiagnosticResult/PatternSets(1)
// PatternSetData(0)/InstrumentResult(0)
//  PatternsSpec Path: Patterns(MemoryBist_P1)/TestStep(run_time_prog)/MemoryBist/None
//  Failing Controller Verilog Instance: core_inst1/top_rtl_tessent_mbist_c1_controller_inst
//    Tested Controller Steps: 0-2
//    Step: 2
//      Algorithm: SMARCHCHKBCIL
//      Tested Memories:
//        M12:core_inst1/blockA_clka_i1/mem6
//        M13:core_inst1/blockA_clka_i2/mem6
//      Failing Memory Verilog Instance: core_inst1/blockA_clka_i2/mem6
//        Tested Ports: 0
//        Failing Port: 0
//          Bitmap:
//          -----------  --------  --------  --------  ---  ----  -------  ---
//          Phase Or     Physical  Physical  Physical  Bit  Phys  Logical  Log
//          Instruction  Bank      Row       Column    Pos  Exp   Address  Exp
//          -----------  --------  --------  --------  ---  ----  -------  ---
//          2_R1W1       0x0       0x0       0x0       3    1     0x0      1
//          3_R1W0       0x0       0x0       0x0       3    1     0x0      1
//          3_R0R0       0x0       0x0       0x0       3    0     0x0      0
//          3_R1W0       0x0       0x0       0x0       3    1     0x0      1
//          3_R0R0       0x0       0x0       0x0       3    0     0x0      0
//          5.5_FP       0x0       0x0       0x0       3    1     0x0      1
//          5.5_FP       0x0       0x0       0x1       3    1     0x1      1
//          5.5_FP       0x0       0x0       0x2       3    1     0x2      1
//          5.5_FP       0x0       0x0       0x3       3    1     0x3      1
```

Example 3

The following command returns the pass/fail results for the test pattern associated with test2.

execute_cdp_test test2

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
command: execute_cdp_test test2
Test 'test2' failed.
```

Example 4

The following script runs the MemoryBist_P1 test ten times, captures the pass/fail results, and reports the results.

```
# Run test 10 times and capture pass/fail results
set status_array {}
for { set i 1 } { $i <= 10 } { incr i } {
    lappend status_array [ execute_cdp_test MemoryBist_P1 ]
}
# Report results after all tests complete
set i 1
foreach failflag $status_array {
    puts -nonewline "** Test $i :  "
    if { $failflag } { puts "FAIL" } else { puts "PASS" }
    incr i
}
** Test 1 :  PASS
** Test 2 :  PASS
** Test 3 :  PASS
** Test 4 :  PASS
** Test 5 :  FAIL
** Test 6 :  PASS
** Test 7 :  PASS
** Test 8 :  PASS
** Test 9 :  PASS
** Test 10 :  PASS
SETUP>
```

Example 5

The following example shows diagnosis of boundary scan failures from a multi-die interconnect test. The tool reports the diagnosis result in a tabular form using the following columns:

- **BoundaryScanRegister name** — Name of the boundary scan register and the instance number in parentheses.
- **Relative Cycle** — Any relative cycles. The report does not include this column if there are no relative cycles to report.
- **Die top ports list** — Top-level list of ports failing for the named boundary scan register and instance.
- **Expected Value** — Expected value to pass the cycle.
- **Actual Value** — Actual value failing the cycle.

**execute_cdp_test MultiDieInterconnect **
-collect_data_type_list { variable instrument diagnosis } -force_diagnose_on_failure

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Test 'MultiDieInterconnect' failed.
//  Variable failures and unmapped failures of pattern 'MultiDieInterconnect.stil' :
//
//  PatternSet                          Variable                 Pin            Expected
//                                                                                Actual
//  ---------------------------------  -----------------------  ---  --------…----…----
//  MultiDieInterconnect__interconnect  BoundaryScanRegister(3)  tdo  bXXX1010…1101…1011
//                                                                     bXXX1010…1100…1011
//  ---------------------------------  -----------------------  ---  --------…----…----
//  MultiDieInterconnect__interconnect  BoundaryScanRegister(6)  tdo  bXXX1010…0101…XX01
//                                                                     bXXX1010…0100…XX01
//  ---------------------------------  -----------------------  ---  --------…----…----
//
//  Diagnostic Result:
//     CDPResult/TestExecutionResult/DiagnosticResult/PatternSets(0)/PatternSetData(0)
//  PatternsSpec Path:
//     Patterns(MultiDieInterconnect)/TestStep(MultiDieInterconnect)/
//
//   Failing BoundaryScan top ports:
//   ==============================
//   -----------------------------------  --------  ----------     --------  ------
//   BoundaryScanRegister                 Relative  Die top        Expected  Actual
//   name                                 Cycle     ports list     Value     Value
//   -----------------------------------  --------  ----------     --------  ------
//   MultiDieInterconnect__interconnect::
//                       BScanRegister(3)  489       top2/gpio[61]  1         0
//                                         134       top1/gpio[125] 1         0
//   MultiDieInterconnect__interconnect::
//                       BScanRegister(6)  489       top2/gpio[61]  1         0
//                                         134       top1/gpio[125] 1         0
//   Patterns(MultiDieInterconnect) : fail
//       TestStep(MultiDieInterconnect) : fail
//           MultiDieInterconnect : fail
//               RunTest(inst_reg) : pass
//               RunTest(bypass_reg) : pass
//               RunTest(id_reg) : pass
//               RunTest(bscan_reg) : pass
//               RunTest(interconnect) : fail
```

Related Topics

add_cdp_test

launch_sid_tester

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

execute_gpib_command

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Enables you to control and program GPIB equipment.

Usage

execute_gpib_command ***gpib_command*** **-address** ***gpib_address*** [-read_result [-timeout *read_timeout_period*]]

Description

Note

Using this command assumes that you have configured the equipment on the GPIB as described in General Purpose Interface Bus (GPIB) standard IEEE-488. It also assumes that the GPIB instrument supports the GPIB commands you specify.

In Tessent SiliconInsight, you generally use GPIB equipment to control power supplies and clock generators for testing purposes. However, you may also control any GPIB instrument—such as a PMU, temperature control, or digital IO—as long as it supports the GPIB standard.

To control GPIB instruments with Tessent SiliconInsight, use the Prologix 6.0 adaptor. Connect this adaptor to a USB port on the Desktop computer and route the GPIB cables from the adaptor to the GPIB instruments. You do not need a setup file. Refer to "Setting Up the Desktop Environment" in the *Tessent SiliconInsight User's Manual for Tessent Shell* for details.

Combining this command with the execute_cdp_test and add_cdp_test commands enables you to generate detailed programs for debugging and characterizing DUTs.

Arguments

- ***gpib_command***

  A required string that specifies a GPIB command and arguments. The GPIB command must be a valid GPIB command that is supported by the targeted GPIB instrument. Specify the command between quotation marks.

- **-address** ***gpib_address***

  A required switch and integer pair that specifies the address of the targeted GPIB equipment, where the ***gpib_address*** is an integer from 0 to 30. This switch enables execution of the GPIB command on the GPIB equipment configured to listen at the specified address.

- -read_result

  An optional switch that directs Tessent SiliconInsight to read the results that the GPIB equipment returns after it runs the specified GPIB command. Use this switch when the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

GPIB command is a query command and you expect a query result from the GPIB equipment. By default, the tool does not read data from the GPIB instrument.

- -timeout *read_timeout_period*

  An optional switch and integer pair that specifies the maximum length of time (in milliseconds) to wait for the result of the specified command to appear on the GPIB before timing out. The default wait period is 2 seconds (2000 ms).

Examples

Refer to "Examples of Controlling Instruments With GPIB" in the *Tessent SiliconInsight User's Manual for Tessent Shell* for more examples.

Example 1

The following shmoo script sweeps the power from 1.1V to 2.5V with increments of 0.1V, and clock frequency from 500MHZ to 1000MHZ with increments of 100MHZ. It shows that the GPIB command for the clock frequency is "SetFrequency *frequency*" and for the power supply is "SetVoltage *volts*".

The assumptions are:

- The Prologix USB-to-GPIB adaptor is physically connected to the Desktop computer.
- The Prologix adaptor cables connect to a clock generator at GPIB address 5 and a power supply (PS) at GPIB address 8, and the GPIB instruments are correctly configured per their documentation.
- The MemoryBist_P1 test was loaded into the hardware using the add_cdp_test command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# As a best practice, before running the script, verify responsiveness
# of the instruments on the GPIB network. The following commands
# return strings that identify the instruments.

execute_gpib_command "*IDN?" -address 5 -read_result
execute_gpib_command "*IDN?" -address 8 -read_result

# The shmoo script
for { set p 1.1 } { $p <= 2.5 } { incr p 0.1 } {
   # Set the voltage
   execute_gpib_command { SetVoltage  $p } -address 8
   for { set frq 500 } { $frq <= 1000 } { incr frq 100 } {
      # Set the frequency for the clock generator
       execute_gpib_command { SetFrequency  $p } -address 5

      # Run the test
      set test_failed [execute_cdp_test MemoryBist_P1]
      puts -nonewline  "PS=${p}V  FRQ=${frq}MHZ  test="
      if { $test_failed } {
         puts "Failed"
      } else {
         puts "Passed"
      }
   }
}
```

Example 2

The following command directs the tool to run the ReadCurrent command on the GPIB instrument that is enabled at address 12. The ReadCurrent command returns results, which Tessent SiliconInsight Desktop then reads because you have specified the -read_result switch. The tool waits no more than 500 ms for the results to appear.

execute_gpib_command “ReadCurrent” -address 12 -read_result -timeout 500

Related Topics

add_cdp_test

execute_cdp_test

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

execute_tester_command

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Enables you to verify and debug the wiring connection between an adaptor used for Tessent SiliconInsight and the design under test.

Usage

execute_tester_command [exec_vecs *adaptor_pin_list vector_list*]
[find_ir_length [*trst tms tdi tdo tck*]]
[check_intermittent_failures *pattern_filename maximum_iterations*]

Description

Enables you to verify and debug the wiring connection between an adaptor used for Tessent SiliconInsight and the design under test.

Use the exec_vecs option to program any of the supported Tessent SiliconInsight adaptors' pins to logic values 0 and 1. Doing this can help you debug the wiring between the adaptor and the DUT.

Use the find_ir_length option to verify that the TAP controller, if present, is wired correctly. If the command returns an error instead of the length of the instruction register, then the TAP pins are not wired correctly.

Refer to "Debug the Wiring Connection Between the Adaptor and the DUT" in the *Tessent SiliconInsight User's Manual for Tessent Shell* for details.

Arguments

- exec_vecs

  An optional literal string that specifies to run the specified vectors at the specified logic values.

- *adaptor_pin_list*

  A list that specifies one or more pin names on the adaptor that you want to control.

- *vector_list*

  A list of logic values (0 or 1) for the specified pins.

- find_ir_length

  An optional literal string that specifies to verify that the five adaptor pins that connect to TAP controller pins are wired correctly. When specified without TAP pin assignments, the tool defaults to using the TAP pin mapping defined in the adaptor's setup specification Pinmap wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- *trst tms tdi tdo tck*

  An optional ordered list of pin numbers that corresponds to the five pins that connect the adaptor to the TAP controller. Specify the pin numbers in the order indicated by the command syntax.

- check_intermittent_failures

  An optional literal string that activates the check for intermittent failures for a given pattern. If an intermittent failure is detected, the tool generates a message similar to the following in *sid_tester.log*:

  ```
  Check for intermittent fails; iteration = 1
  Check for intermittent fails; iteration = 2
  WARNING: INTERMITTENT_FAILURE: Got intermittent fails for
  'MemoryBist_P1_2.svf' on iteration 2.
     Previous run failures: ' 13@5@3 17@25@3 17@34@3 17@56@3 17@63@3'
     Current  run failures: ' 13@5@3 17@25@3 17@34@3 17@56@3 17@63@3 17@63@8'
  ```

  Search the log file for “INTERMITTENT_FAILURE” to locate all instances of intermittent failures.

- *pattern_filename*

  An optional filename that specifies the diagnostic pattern to use to check for intermittent failures.

- *maximum_iterations*

  An optional integer that specifies the maximum number of iterations to run when checking for intermittent failures.

Examples

Example 1

The following example for a signalyzerSHA40 adaptor sets pins C0 and C7 to 1 and the rest of the pins (C1, C2, C3, C4, C5, and C6) to 0:

execute_tester_command exec_vecs { C0 C1 C1 C3 C4 C5 C6 C7 } 10000001

Example 2

The following example runs the diagnostic pattern in *MemoryBist_P1_2.svf* up to 250 times to check for intermittent failures when running the test MemoryBist_P1:

execute_tester_command check_intermittent_failures MemoryBist_P1_2.svf 250

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

exit

Context: unspecified, all contexts

Mode: all modes

Terminates the application tool program.

Usage

exit [-force] [*exitval*]

Description

Terminates the application tool program.

The exit command terminates the tool session and returns to the operating system. You should either save the current test patterns or modified design before exiting the tool.

If you are operating in interactive mode (not running a dofile) and you do not save the current test pattern set or modified design, the tool displays a warning message, and you are given the opportunity to continue the session and save the test patterns before exiting.

Arguments

- -force

  An optional argument that forces termination of the current tool session without saving any data.

- *exitval*

  An optional integer that specifies an exit code.

  This option can be used to return a non-zero value when exiting the tool. By default, when there's no error, “0” is returned to the caller while non-zero usually indicates an error condition. The specific value returned can be used to differentiate the type of condition that resulted in issuing the exit command. When a tool terminates on a fatal signal whose number is *N*, the tool uses the value 128+*N* as the exit status.

Examples

The following example quits the tool without saving the current test pattern set.

```
set_system_mode analysis
create_patterns
exit -force
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

expand_compressed_patterns

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: analysis

Expands compressed 1-hot patterns for better internal failing flop diagnosis.

Usage

expand_compressed_patterns [-CHAIN ***chain_name***...]
[-MAX_1hot_pattern ***pattern_number***] [-MAP_file ***map_filename***] [-REPlace]

Description

Expands compressed 1-hot patterns for better internal failing flop diagnosis.

See "1hot Compressed Pattern Expansion" in the *Tessent Diagnosis User's Manual* for complete information.

Arguments

- -CHAIN ***chain_name***...

  An optional switch and repeatable string that identifies the scan chain name(s) to process for 1-hot expansion.

- -MAX_1hot_pattern ***pattern_number***

  An optional switch and required integer that specifies the maximum number of patterns to produce during expansion. The default value is 1000. If this limit is exceeded, the tool stops command execution and prints a warning message that the limit was exceeded.

- -MAP_file ***map_filename***

  An optional switch and required string that specifies the map filename where 1-hot pattern-IDs, original pattern-IDs, and mapped channel information is saved. Use the -replace option to overwrite an existing map file.

- -REPlace

  An optional switch for overwriting an existing file.

Related Topics

read_patterns

set_pattern_filtering

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

extract_fault_sites

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Prerequisites: The design must be flattened to the simulation model, and you must have a Layout Database opened using the open_layout command.

Extracts interconnect bridges and opens from a layout database and writes out the data in UDFM file format.

Usage

extract_fault_sites **-output_file *output_filename***
[-defect_types {all | bridges | opens}]
[-min_tca *min_tca_value*]
[-max_tca *max_tca_value*]
[-max_distance *max_distance_value*]
[-max_defects {*max_defects_value* | unlimited}]
[-max_bridges {*max_bridges_value* | unlimited}]
[-max_opens {*max_opens_value* | unlimited}]
[-open_sorting {segment | net}]
[-critical_area_calculation {method1 | method2}]
[-strip_prefix *prefix*]
[-add_prefix *prefix*]
[-marker_file *filename*]
[-replace]
[-verbose]
[-details {on | off}]

Description

This command extracts interconnect bridges and opens from a Layout Database (LDB) and writes them to a file in UDFM file format.

Note

The extract_fault_sites command is only compatible with LDBs created with Tessent v2014.4 or later.

For complete information, refer to "Interconnect Bridge and Open Extraction and UDFM Creation" in the *Tessent Scan and ATPG User's Manual*.

Note

Using this command requires an Automotive-Grade ATPG license feature. See "Feature Entry in the License File" in the *Managing Tessent Software* manual.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When running the extract_fault_sites command, the tool outputs a summary of the applied fault site extraction settings and a distribution graph, displaying the number of defects in relation to the total critical area. Refer to "Example 6" on page 848 for an example.

Arguments

- -output_file ***output_filename***

  A switch and string pair that specifies the name of the UDFM file into which the tool writes the extracted fault sites. The UDFM files are automatically compressed if the extension is *.gz*.

- -defect_types all | bridges | opens

  An optional switch and literal pair that specifies the defect type(s) the tool writes to the UDFM file.

  all — Writes out both bridges and opens.

  bridges — Writes out all bridges including those to power and to ground. This is the default.

  opens — Writes out opens.

  You specify multiple defect types using the following syntax:

  ```
  -defect_types {bridges opens}
  ```

- -min_tca *min_tca_value*

  An optional switch and value pair that specifies the minimum total critical area of a defect to be extracted from the LDB. A valid *min_tca_value* is a non-negative real number. The tool extracts all defects with a total critical area greater than or equal to the specified value. The default is 0, which means no limitation is set.

- -max_tca *max_tca_value*

  An optional switch value pair that specifies the maximum total critical area of a defect to be extracted from the LDB. A valid *max_tca_value* is a non-negative real number. By default no limitation is set.

- -max_distance *max_distance_value*

  An optional switch and value pair that specifies the maximum distance between two nets to be accepted as a bridge defect. A valid *max_distance_value* is a non-negative real number, where the specified number is the factor to be multiplied by the technology length to calculate the real physical distance.

  The default is 3.0; this means that by default the tool only extracts from the LDB those bridges with a distance between the nets of three times the technology length, or less.

  This setting has no effect on interconnect open defects.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -max_defects *max_defects_value* | unlimited

  An optional switch and value or literal pair that specifies the maximum total number of defects (containing bridges and opens) the tool writes to the UDFM file. Specify one of the following:

  max_defects_value — A natural number that specifies the maximum number of defects.

  unlimited — A literal that instructs the tool to write all defects to the UDFM file. This may cause increases in both runtime and the size of the output file. This is the default.

- -max_bridges *max_bridges_value* | unlimited

  An optional switch and value or literal pair that specifies the maximum number of bridges the tool extracts from the LDB. Specify one of the following:

  max_bridges_value — A natural number that specifies the maximum number of bridges.

  unlimited — A literal that instructs the tool to extract all bridges from the LDB. This may cause increases in both runtime and the size of the output file. This is the default.

- -max_opens *max_opens_value* | unlimited

  An optional switch and value or literal pair that specifies the maximum number of opens the tool extracts from the LDB. Specify one of the following:

  max_opens_value — A natural number that specifies the maximum number of opens.

  unlimited — A literal that instructs the tool to extract all opens from the LDB. This may cause increases in both runtime and the size of the output file. This is the default.

- -open_sorting segment | net

  An optional switch and literal pair that defines the sorting of the open's section in the UDFM file. Specify one of the following:

  segment — A literal that instructs the tool to sort the open groups in descending order by segment, specifically by the total critical area of the corresponding net segment of the open groups. This is the default.

  net — A literal that instructs the tool to sort the list of fault definitions using the following two levels:

  - **First Level** — The tool sorts in descending order the fault definitions by the summarized total critical area of all segments for each open group.
  - **Second Level** — The tool sorts all fault definitions for the segments of a particular open group in descending order by the specific total critical area value.

- -critical_area_calculation method1 | method2

  An optional switch and literal pair that specifies the method the tool uses to calculate the critical area. Choose one of the following:

  method1 — An integral calculation. Depending on the specified defect type, either method 1 for bridges, or method 1 for opens and vias is applied. This is the default. Refer to "Critical Area Calculation Formula for Bridges" and "Critical Area

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Calculation Formula for Opens and Vias" in the *Tessent Scan and ATPG User's Manual*.

method2 — An analytical solution based on a wider range of bridge defects but taking critical area saturation into account. Refer to "Alternative Critical Area Calculation Formula for Bridges" in the *Tessent Scan and ATPG User's Manual*.

Note

As method 2 is only supported for bridges, the tool issues an error message when you specify method 2 with the "-defect_types opens" switch and literal.

- -strip_prefix *prefix*

  An optional switch and string pair that removes the prefix of the net path name if there is an inconsistency between the flat model and LDB net names.

- -add_prefix *prefix*

  An optional switch and string pair that adds a prefix to the net path name if there is an inconsistency between the flat model and the LDB net names.

- -marker_file *filename*

  An optional switch and string pair that instructs the tool to generate a marker file with the specified *filename* and that contains the extracted defects.

- -replace

  An optional switch that instructs the tool to overwrite the contents of a UDFM or marker file in the event a file of the same name exists.

- -verbose

  An optional switch that instructs the tool to output the progress of the various steps of the extract_fault_sites command.

- -details on | off

  An optional switch and literal pair that controls the amount of data extracted from the LDB. Specify one of the following:

  on — A literal that instructs the tool to extract verbose information from the LDB. This is the default.

  off — A literal that instructs the tool to extract only essential information from the LDB resulting in smaller UDFM output files.

Examples

Example 1

In this example, the tool extracts interconnect bridges with the command's default parameters. As a result, the tool-generated UDFM file, *interconnect_bridges.udfm*, contains all interconnect bridges with a distance between the bridging objects up to three times the technology length. The -max_distance switch's default is 3.0; consequently, the tool considers interconnect bridges up to three times the technology length.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_context patterns -scan_diagnosis
read_flat_model design.flat.gz
open_layout design.ldb
// optionally, specify the set_marker_file_options marker_file_options here.

extract_fault_sites -defect_types bridges -output_file interconnect_bridges.udfm \
   -marker_file interconnect_bridges.marker
```

Example 2

In this example, the tool extracts interconnect opens.

The tool-generated UDFM file is named *interconnect_opens.udfm*, and the tool-generated marker file is named *interconnect_opens.marker*. The maximum number of opens is set to "unlimited", which means the tool extracts all existing opens and writes the result to the UDFM file, regardless of the total critical area. For the marker file, the number of opens the tool writes to the file depends on the settings for the marker file options. See set_marker_file_options.

```
extract_fault_sites -defect_types opens -output_file interconnect_opens.udfm \
   -marker_file interconnect_opens.marker -max_opens unlimited
```

Example 3

In the following example, the default defect type "bridges" is applied. The tool extracts interconnect bridges up to a defect size of 10 times the technology length. In addition, the tool only writes out those bridges with a TCA value greater than or equal to 100. The tool-generated UDFM file is named *interconnect_bridges.udfm*, and the tool-generated marker file is named *interconnect_bridges.marker*.

```
extract_fault_sites -output_file interconnect_bridges.udfm \
   -marker_file interconnect_bridges.marker -max_distance 10.0 -min_tca 100.0
```

Example 4

In this example, the tool creates three UDFM files and their corresponding marker files.By default, the tool only considers the defect type "bridges" for each file pair.

- The first extract_fault_sites command creates a UDFM file for the bridges with the total critical area equal or greater than 100.0.
- The second extract_fault_sites command creates a UDFM file for the bridges that have a total critical area in the range of a value equal or greater than 100.0 up to 1000.0.
- The third extract_fault_sites command creates a UDFM file for the bridges with a total critical area in the range of a value equal or greater than100.0 up to 500.0. That means, all bridges with a total critical area less than 100 or greater than 500 are not written out.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

**extract_fault_sites -output_file tca_100_or_greater.udfm \
-marker_file tca_100_or_greater.marker -min_tca 100.0**

**extract_fault_sites -output_file tca_100_1000.udfm -marker_file tca_100_1000.marker \
-min_tca 100.0 -max_tca 1000.0**

**extract_fault_sites -output_file tca_100_500.udfm -marker_file tca_100_500.marker \
-max_tca 500.0 -min_tca 100.0**

Example 5

In this example, the tool extracts both interconnect bridges and opens. The tool extracts interconnect bridges up to a distance of 10 times the technology length. For opens, the tool does not consider the -max_distance switch setting. The tool-generated UDFM file is named *all.udfm*, and the tool-generated marker file is named *all.marker*.

**extract_fault_sites -output_file all.udfm -max_distance 10 -defect_types { all } \
-marker_file all.marker -verbose**

The output of the -verbose switch provides additional information about the progress of the various steps and looks as follows:

```
//  Interconnect bridge extraction started...
//  Interconnect bridge extraction completed.
//  1101486 bridges have been ignored due to distance filtering
//    (max. distance: 3.000).
//  Bridge fault location mapping started...
//   10% of the bridge fault location mapping has been performed.
//     Estimated remaining effort: 304 sec
//   20% of the bridge fault location mapping has been performed.
//     Estimated remaining effort: 249 sec
//   30% of the bridge fault location mapping has been performed.
//     Estimated remaining effort: 213 sec
//   40% of the bridge fault location mapping has been performed.
//     Estimated remaining effort: 180 sec
//   50% of the bridge fault location mapping has been performed.
//     Estimated remaining effort: 148 sec
//   60% of the bridge fault location mapping has been performed.
//     Estimated remaining effort: 118 sec
//   70% of the bridge fault location mapping has been performed.
//     Estimated remaining effort: 87 sec
//  Bridge fault location mapping completed.
```

Example 6

This example illustrates the log file output that contains a summary of command line options, and a distribution graph containing the number of defects in relation to the total critical area:

**extract_fault_sites -output_file apt1_EXT_bridges_tca0_100000.udfm \
-marker_file apt1_EXT_bridges_tca0_100000.marker -defect_type bridges \
-max_bridges unlimited -max_distance 10.0 -min_tca 0 -max_tca 100000 \
-strip_prefix /top -replace**

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//                    |                            # Bridges
//       TCA[ts]      |          10       100      1000     10000    100000   1000000
//                    |           |         |         |         |         |         |
//  ----------------  -  ------------------------------------------------------------
//  >0.0 -- 1.3       |  ******************************************************** 522049
//  >1.3 -- 1.6       |  ************************************************** 80106
//  >1.6 -- 2.0       |  ************************************************** 85533
//  >2.0 -- 2.5       |  ************************************************** 85089
//  >2.5 -- 3.2       |  ************************************************** 96360
//  >3.2 -- 4.0       |  ************************************************** 102026
//  >4.0 -- 5.0       |  ******************************************************* 184505
//  >5.0 -- 6.3       |  ***************************************************** 166067
//  >6.3 -- 7.9       |  ************************************************** 110158
//  >7.9 -- 10.0      |  ************************************************** 91957
//  >10.0 -- 12.6     |  ************************************************** 100347
//  >12.6 -- 15.8     |  ************************************************* 83890
//  >15.8 -- 20.0     |  ******************************************************* 115009
//  >20.0 -- 25.1     |  ******************************************************* 135980
//  >25.1 -- 31.6     |  ******************************************************** 172761
//  >31.6 -- 39.8     |  ******************************************************* 141147
//  >39.8 -- 50.1     |  ******************************************************* 121717
//  >50.1 -- 63.1     |  ***************************************************** 108647
//  >63.1 -- 79.4     |  ************************************************** 75381
//  >79.4 -- 100.0    |  ********************************************** 53962
//  >100.0 -- 125.9   |  ********************************************* 37898
//  >125.9 -- 158.5   |  ******************************************** 26094
//  >158.5 -- 199.5   |  ******************************************* 18901
//  >199.5 -- 251.2   |  ****************************************** 14155
//  >251.2 -- 316.2   |  ***************************************** 11160
//  >316.2 -- 398.1   |  ************************************** 7498
//  >398.1 -- 501.2   |  ************************************* 5132
//  >501.2 -- 631.0   |  ********************************** 3495
//  >631.0 -- 794.3   |  ********************************* 2296
//  >794.3 -- 1000.0  |  ****************************** 1386
//  >1000.0 -- 1258.9 |  ***************************** 875
//  >1258.9 -- 1584.9 |  ************************** 417
//  >1584.9 -- 1995.3 |  ********************** 174
//  >1995.3 -- 2511.9 |  ******************** 92
//  >2511.9 -- 3162.3 |  ***************** 49
//  >3162.3 -- 3981.1 |  ************* 21
//  >3981.1 -- 5011.9 |  ********* 8
//  >5011.9 -- 6309.6 |  ***** 3
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

extract_icl

Context: all contexts

Mode: all modes

Checks the ICL connectivity rules between IJTAG instances, extracts the top-level ICL module, creates an SDC file that can be used for synthesis and static timing analysis (STA), and extracts an IJTAG graybox model.

Usage

extract_icl [-write_in_tsdb {on | off | auto}] [-skip_sdc_extraction]
[-create_ijtag_graybox on | off] [-variant_icl_id *variant_id*]

Description

Refer to the extract_sdc command for details on SDC file creation and usage.

Refer to "ScanInterfaces and Associations Between Ports and ScanInterfaces" in the *Tessent IJTAG User's Manual* for an explanation of the creation, inference, checking, and introspection of ScanInterfaces.

Using the extract_icl command is almost identical to explicitly using "set_context patterns -ijtag" followed by "set_system_mode analysis" with the following notable differences:

1. If ICL is already elaborated, using "set_system_mode analysis" in context "patterns -ijtag" does not extract a new version of the top ICL module. The tool simply transitions to analysis mode so that you can use the currently elaborated ICL to retarget PDL. When you call extract_icl, the top level ICL module is deleted and a new one is forced to be re-extracted.
2. If you change the module matching options using the set_module_matching_options command, it only has effect the next time ICL matching is done. You have to reissue the "set_context patterns -ijtag" command or the set_current_design command to redo the ICL module matching. Using the extract_icl command, the module matching is redone systematically.
3. If the ICL is already elaborated and there are input constraints added on the top level IJTAG ports, they are automatically deleted prior to running ICL extraction. Those constraints are automatically added in a previous step if the ICL was elaborated and a setup to analysis transition was done such that the value loaded in the ICL network did not get disturbed during the state stability check.
4. If the ICL is already elaborated when the extract_icl command is called and the top ICL module contains ScanInterfaces, the name of those scan interfaces and their port list is stored before the current top level ICL module is deleted so that the original names and port list can be restored in the re-extracted top level ICL module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

5. Finally, when you use extract_icl, the context from which it was called is restored once the ICL is extracted. The "set_system_mode analysis" command leaves you in the analysis mode of the "patterns -ijtag" context.

As documented in the "dft_inserted_designs" section of the "Tessent Shell Data Base (TSDB)" on page 4807, if the -write_in_tsdb switch is not specified to off, the PDL files associated to the ICL files that were checked are collected into a single file and stored beside the extract ICL file. The extracted PDL file is also automatically sourced into memory after extraction. Each time the extracted ICL file is read in and the top ICL module elaborated using the set_current_design command, the PDL file is automatically sourced on memory.

As described in Table 3-12 on page 210 of the add_dft_signals command reference page, the all_test DFT signal is treated in a special manner during ICL extraction. Because the tool has automation to make sure the all_test DFT signal is always high during ICL usage, Tessent Shell forces this DFT signal to high to enable sensitizing clock gaters so that the clock path can be extracted and modeled in ICL. The ICL representation does not describe the clock gater and would become invalid if the all_test signal was not automatically set to 1. If you are using Tessent MemoryBIST and use the pre-DFT DRC DFT_C1, any clock gaters seen fanning out to memories are disabled by a special DFT signal called nonscan_test. Again, the clock gaters are not modeled in the ICL representation but there are no issues because the memoryBIST PDL automatically sets this DFT signal to 1 during all memory BIST patterns. If you want to use a different DFT signal to enable your clock gater, you must model the clock gater in ICL, as shown in "Example 3" on page 853 below.

Flow to Avoid Re-Running extract_icl

You can avoid having to re-run the extract_icl command on a parent block after redoing DFT insertion on a child block. This is possible only when the interface of the child block did not change as a consequence of redoing DFT insertion. You can still choose to re-run the extract_icl command to update the ICL/PDL.

For example, suppose that DFT insertion step named "gate" on a child block "core" is redone, which does not change the footprint of the block (no port changes to the child) but changes the netlist/ICL/PDL. After this finishes, it is possible to update the parent design "top" DFT insertion pass "gate" without having to run ICL extraction by running the following commands:

```
set_context dft -design_id gate
# The following skips reading ICL/PDL for child blocks.
read_design top -design_id gate
# Read all child blocks designs make sure all ICL/PDL in memory is
# current.
open_tsdb core.tsdb
read_design core -design_id gate -view interface
set_current_design top
write_design -tsdb -replace
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Logic Synthesis and Port Name Mapping

Logic synthesis can flatten or bit-blast ports, causing post-synthesis names to differ from those in the initial RTL. This process is automatic and typically does not require user intervention. For more information, refer to "Automatic Design Mapping" in the *Tessent Shell User's Manual*.

Licensing

If you use at least one Tessent signed instrument (MemoryBIST, LogicBIST, TestKompress, MissionMode) at your current design or at any lower hierarchical level, then any one of the respective licenses is sufficient for this command to complete. In all other cases, you need either an IJTAG or a MissionMode license.

Arguments

- -write_in_tsdb on | off | auto

  An optional switch and literal pair that controls whether the extracted ICL is automatically written out in the specified TSDB output directory. The TSDB output directory is specified with the set_tsdb_output_directory command and defaults to "tsdb_outdir".

  The auto value means "on" when the directory returned by get_tsdb_output_directory already exists, otherwise auto means "off."

  With the default value of "auto", the ICL file is only written if the directory *tsdb_outdir*/dft_inserted_designs/*current_design_design_id.dft_inserted_design* already exists. This directory is typically created by the process_dft_specification command prior to invoking the extract_icl command as shown in Example 1.

- -skip_sdc_extraction

  An optional switch that specifies skipping SDC generation during ICL extraction.

- -create_ijtag_graybox on | off

  An optional switch and literal pair that creates an IJTAG graybox view within the TSDB output directory. When you run extract_icl in RTL mode, you can generate the IJTAG graybox during ICL extraction because the tool automatically calls quick synthesis if there is RTL logic between the IJTAG instances. Quick synthesis internally calls the "analyze_graybox -ijtag" command followed by the "write_design -graybox" command. Refer to the analyze_graybox reference page for more information about IJTAG graybox analysis.

  If you need extra instances to be preserved in the IJTAG graybox view, set the preserve_in_graybox attribute on those instances before you call the extract_icl command.

- -variant_icl_id *variant_id*

  An optional switch and string pair that specifies an ICL variant identifier. It causes extraction of an additional ICL file to the TSDB output directory named "*design_name*.icl_*variant_id*." A variant ICL file may be necessary if a design has constructs that you cannot represent in ICL. For example, a bidirectional port does not exist in ICL. It is also possible for some functional logic to be ignored by ICL extraction. In both

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

cases, input constraints are used to configure the circuit such that ICL extraction succeeds. An ICL attribute named "tessent_variant_id" takes the *variant_id* value for the current design ICL module.

Examples

Example 1

The following example shows the use of the extract_icl command directly following the process_dft_specification command.

```
SETUP> set_context dft -rtl
#commands to load design appear here
SETUP> set_current_design CoreA
SETUP> set_design_level physical_block
SETUP> set_system_mode analysis
ANALYSIS> create_dft_specification
ANALYSIS> process_dft_specification
ANALYSIS> extract_icl
```

Example 2

The following example shows the use of the extract_icl command as part of the "patterns -scan" context to extract the ICL network such that it can be used for test setup. For efficiency, it is recommended to extract the ICL only once after the DFT is completed, and to reuse the extracted ICL in all pattern -scan usage. ICL extraction may also not be possible after layout if the ICL modules were ungrouped during layout. The standard flow is to use the pre-layout ICL file extracted either on the RTL or netlist view for all subsequent pattern generation phases. Formal verification is typically used to verify that the functional mode, including the IJTAG network, was not affected by the layout process.

```
SETUP> set_context patterns -scan
#commands to load design and cell library appear here
SETUP> set_current_design
SETUP> extract_icl
SETUP> set_test_setup_icall myproc1
```

Example 3

In the normal flow, clock gaters are opened using either the all_test or the nonscan_test DFT signals as explained in the last section of the Description section above. If you want to enable the clock gaters using a different DFT signal, you must model the clock gater in ICL using a ClockMux element as shown below. The clock gater is modeled in the ICL view and if you forget to enable the clock gater and an iClock command is issued on a ClockPort in the fanout of the clock gater, the tool issues an error telling you that the clock is blocked at the clock gater as shown below.

The ICL module is defined for the clock gater cell module. It contains the ICL module attribute use_in_icl_extraction set to "when_reached_by_other_modules". This attribute tells ICL extraction to ignore this ICL module for all clock gater instances not reached while tracing the ClockPort of another module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Disabling the clock gater with a different DFT signal:

register_static_dft_signal_names my_clock_en -default_val_in_all_test 1
add_dft_signals my_clock_en

add_dft_control_points cgand1/FE -dft_sig my_clock_en
check_design_rules

To extract ICL when the clock gater has an ICL Clock port in its fanout, model the clock gater as follow:

```
Module cgand {
  Attribute use_in_icl_extraction = "when_reached_by_other_modules";
  ClockPort CK;
  ToClockPort GCK {
    Source mux;
  }
  DataInPort FE;

  ClockMux mux SelectedBy FE {
    1'b0 : 1'b0;
    1'b1 : CK;
  }
}
```

If an iProc calls iClock on a ClockPort that is in the fanout of the clock gater and you did not set the DFT signal my_clock_en to 1, the tool issues this error:

```
//  Error: Unable to trace iClock 'sub1.clk' to top module input port.
//  Encountered a constant ICL source connection 'cgand1.mux' while
//  tracing clock source.
```

If the clock gater was not modeled in ICL, the tool would not report this issue, and the problem would only show up during simulation.

Related Topics

open_visualizer

process_dft_specification

set_system_mode

set_tsdb_output_directory

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

extract_inter_cell_data

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Prerequisites: The design must be flattened to the simulation model, and you must have a Layout Database opened using the open_layout command.

Extracts neighborhood cell combinations and writes the combinations out in UDFM file format.

Usage

extract_inter_cell_data **-output_file *output_filename***
[-cell_pairs {*cellname1 cellname2 ...*}]
[-max_cell_pairs {*max_cell_pairs_value* | unlimited}]
[-min_locations *min_locations_value*]
[-strip_prefix *prefix*]
[-add_prefix *prefix*]
[-min_tbl *min_tbl_value*]
[-placement_filter {off | horizontal | vertical}]
[-marker_file *filename*]
[-replace]
[-list_cells]

Description

This command extracts neighborhood cell combinations from a Layout Database (LDB) and writes the cell combinations out to a file in UDFM file format.

Note

Using this command requires an Automotive-Grade ATPG license feature.

Tessent Log File

When running the extract_inter_cell_data command, the tool generates a Tessent log file. Refer to "Example 6" on page 858 for an explanation of the file's contents.

Arguments

- **-output_file *output_filename***

  A required switch that specifies the name of the file to write the extracted inter-cell data.

- -cell_pairs {*cellname1 cellname2 ...*}

  An optional switch and string or strings enclosed in braces ({ }) that specifies one or more neighboring cell pairs to be extracted. The syntax for extracting multiple cell pairs is as follows:

```
-cell_pairs { AND2_X1 NAND3_X4 AO222_X1 AOI222_X2 }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In this example, the first cell pair is “AND2_X1 NAND3_X4”, and the second cell pair is “AO222_X1 AOI222_X2”.

Valid cell names are all cells that are present in the design. Note that the writing of the cell names is case-sensitive. You may also use wildcards in cases when the exact cell name is not known, or you want to extract all cells of a certain type. The syntax for using wildcards is as follows:

```
-cell_pairs { AND* OR* }
```

In this example, all cell pairs with *cellname1* beginning with “AND” and *cellname2* beginning with “OR” are extracted.

You must specify an even number of cells between the braces. When no cell pair is specified, the tool extracts all existing neighbor cell combinations. When the tool does not find the specified cells, or when a wildcard is set and the tool finds no corresponding cells, then the tool issues an error message containing the list of all existing cells.

- -max_cell_pairs *max_cell_pairs_value* | unlimited

  An optional switch and integer or literal pair that specifies the maximum number of cell pairs the tool extracts from the LDB. Note that the tool sorts the cell pairs from most important to least important. When you use this command, the tool writes out the specified number of the most important cell pairs. The criteria for what is most important is based on the total bridging length (TBL) per cell pair and its number of locations (instantiations) in the design.

  The default is 100,000.

- -min_locations *min_locations_value*

  An optional switch to filter out cell pairs to be written to the data file.

  For example, if you specify the “-min_locations 3” command and switch, the tool writes out all cell pairs instantiated at least three times. In contrast, the tool omits all other cell pairs instantiated less than three times.

  The default is 1.

- -strip_prefix *prefix*

  An optional switch and string pair that removes the prefix of the instance path name in the event of inconsistency in the instance names between the flat model and the LDB.

- -add_prefix *prefix*

  An optional switch and string pair that adds a prefix to the instance path name in the event of inconsistency in the instance names between the flat model and the LDB.

- -min_tbl *min_tbl_value*

  An optional switch and value pair that specifies the minimum total bridging length (TBL) per cell pair the tool uses during extraction from the LDB. Using this option enables filtering out cell pairs that do not meet the criteria.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The default is 0.0.

- -placement_filter off | horizontal | vertical

  An optional switch and value pair that specifies the selection of horizontally or vertically neighboring cell pairs to be extracted. By default, no filter is set, that means both horizontally and vertically neighboring cell pairs are extracted.

- -marker_file *filename*

  An optional switch and string pair that generates a marker file with the specified filename and contains the specified extracted cell pairs.

- -replace

  An optional switch that overwrites an existing file. This switch applies to both the data file and the marker file.

- -list_cells

  An optional switch that lists all cells that are present in the netlist library as well as the LDB library of the design.

Examples

Example 1

In this example, all existing neighbor cell combinations are written to the output file named *neighborhood.data* by replacing an existing file with the same name.

```
set_context patterns -scan
read_flat_model design.flat.gz
open_layout design.ldb
// optionally, specify the set_marker_file_options marker_file_options here.

extract_inter_cell_data -output_file neighborhood.data -marker_file neighborhood.marker \
    -replace
```

Example 2

In this example, all existing horizontal neighbor cell combinations are written to the output file named *horizontal_neighbors.data* by replacing an existing file with the same name.

```
extract_inter_cell_data -output_file horizontal_neighbors.data \
    -placement_filter horizontal -replace
```

Example 3

In this example, all existing horizontal cell pairs of the cells named AND2_X1 and NAND3_X4 are extracted and written to the output file named *two_cells_only.data* by replacing an existing file with the same name.

```
extract_inter_cell_data -output_file two_cells_only.data -cell_pairs { AND2_X1 NAND3_X4 } \
    -placement_filter horizontal -replace
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

In this example, the 10,000 most important cell pairs are written to the output file named *top_10000_cells.data*, by replacing an existing file with the same name. The criteria of most important is based on the total bridging length (TBL) per cell pair and its number of locations (instantiations) in the design.

```
extract_inter_cell_data -output_file top_10000_cells.data -max_cell_pairs 10000 -replace
```

Example 5

In this example, all neighbor cell combinations with cell1 beginning with “AND” and cell2 beginning with “OR” are written to the output file named *AND_OR.data* by replacing an existing file with the same name. Also a list of cells containing all cells beginning with “AND” and “OR” is written to the log file.

```
extract_inter_cell_data -output_file AND_OR.data -cell_pairs {AND* OR*} -list_cells -replace
```

Example 6

The following shows an example Tessent log file that the tool generates when you issue the extract_inter_cell_data command:

There are three major parts to look at. The log file starts with a summary of the applied command line options followed by a distribution graph, displaying the number of required CellModelGen-runs in relation to the total bridging length (TBL) of the cell pairs.

```
extract_inter_cell_data -output_file apt1_EXT_cells_tbl0_100000.udfm \
   -marker_file apt1_EXT_cells_tbl0_100000.marker -min_tbl 0 -max_cell_pairs unlimited \
   -strip_prefix /top -list_cells -replace
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This is followed by the list of cells (see the following example) showing an extract:

```
//  Total number of library cells            : 599
//  Total number of filler cells             : 32
//  LADB cells: Number of cells              : 604

//  -------------- -  -------- - ------ - ------- - ------- - ---------- - ---- -
//    Cellname     |  Type     | Active | DX (um) | DY (um) | Placements | Pins |
//  -------------- -  -------- - ------ - ------- - ------- - ---------- - ---- -
//  xor2_08        |  BOTH     | yes    | 8.16    | 4.32    | 2          | 3    |
//  xor2_04        |  BOTH     | yes    | 6.24    | 4.32    | 26         | 3    |
//  xor2_02        |  BOTH     | yes    | 5.28    | 4.32    | 167        | 3    |
//  xor2_01        |  BOTH     | yes    | 4.80    | 4.32    | 37         | 3    |
//  xnor2_08       |  BOTH     | yes    | 8.16    | 4.32    | 1          | 3    |
//  fill_01        |  LEF      | no     | 0.96    | 4.32    | 203        | 1    |
//  sffqsn_01      |  BOTH     | yes    | 12.00   | 4.32    | 4          | 6    |
//  sffqrn_01      |  BOTH     | yes    | 12.48   | 4.32    | 310        | 6    |
//  sffq_01        |  BOTH     | yes    | 10.56   | 4.32    | 894        | 5    |
//  or4_08         |  BOTH     | yes    | 6.24    | 4.32    | 1          | 5    |
//  or4_01         |  BOTH     | yes    | 2.88    | 4.32    | 1          | 5    |
//  or3b_04        |  BOTH     | yes    | 4.80    | 4.32    | 1          | 4    |
    ...
//  or2_02         |  BOTH     | yes    | 2.40    | 4.32    | 880        | 3    |
//  or2_01         |  BOTH     | yes    | 1.92    | 4.32    | 230        | 3    |
//  net_01         |  NETLIST  | no     | -       | -       | -          | -    |
//  oai422b_01     |  BOTH     | yes    | 3.36    | 4.32    | 159        | 5    |
    ...
//  oai4112_12     |  BOTH     | yes    | 10.08   | 4.32    | 8          | 5    |
//  nor2b_12       |  BOTH     | yes    | 10.08   | 4.32    | 2          | 3    |
//  and3b_04       |  BOTH     | yes    | 4.80    | 4.32    | 1          | 4    |
//  -------------- -  -------- - ------ - ------- - ------- - ---------- - ---- -

//  Total number of library cell instances   : 845018
//  Number of considered library instances   : 599
//  Number of created cell pair combinations : 123942
//  Number of cell pair instances            : 252133
```

The table header gives a summary of the table as follows:

```
// Total number of library cells: 599
```

This is the number of different used cells in the design that are both in Verilog and in LEF.

```
// Total number of filler cells: 32
```

This is the number of cells that are added to the layout as filler cells but are ignored, because they are not part of the Verilog file and have no function in the design.

```
// LADB cells: Number of cells: 604
```

This is the number of valid LEF cells. The difference between the number of library cells and the number of LADB cells results from cells that are only in LEF and not in Verilog, or vice versa.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The content of the table by column is as follows:

- **Cellname** — These are all Verilog cell names and all valid LEF cell names
- **Type** — This is the type of each cell
  - **NETLIST** — This cell is only defined in Verilog
  - **LEF** — This cell is only defined in the LEF files
  - **BOTH** — This cell is defined both in Verilog and in LEF

    If the Type is “BOTH” then the cell names are taken into account for the cell neighbor extraction.
- **Active** — Displays if the cell name is taken into account (yes) or not (no). You can reduce the number of active cells with the command line option “-cell_pairs {cell1 cell2}”, this sets all other cells not Active.
- **DX**, **DY** — These are the cell dimensions in microns from the specified LEF files
- **Placements** — The number of placements of this cell from DEF
- **Pins** — The number of signal input and output pins *excluding* power/ground pins (LEF)

The table footer gives a summary of the design as follows:

```
//  Total number of library cell instances   : 845018
```

This is the number of occurrences of the different library cells in the design.

```
//  Number of considered library instances   :    599
```

This is the number of cells in the design that are considered for the cell pair creation.

```
//  Number of created cell pair combinations : 123942
```

This is the number of different cell pair combinations in the design.

```
//  Number of cell pair instances            : 252133
```

This is the number of occurrences of the different cell pair combinations in the design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This is followed by a cell pair distribution graph, see below example:

```
//  ---------------- - ------------------------------------------------------------------
//                   |                          # Tbl[%]
//     # CMG runs     |   25                50                      75                100
//                   |    |                 |                        |                 |
//  ---------------- - ------------------------------------------------------------------
//  0 -- 2477        | **** 34.35%
//  2478 -- 4955     | ************* 42.77%
//  4956 -- 7433     | ******************* 48.26%
//  7434 -- 9911     | ********************** 52.50%
//  9912 -- 12389    | *************************** 55.88%
//  12390 -- 14867   | ****************************** 58.97%
//  14868 -- 17345   | ********************************** 61.73%
//  17346 -- 19823   | ************************************ 64.18%
//  19824 -- 22301   | ************************************** 66.26%
//  22302 -- 24779   | **************************************** 68.19%
//  24780 -- 27257   | ****************************************** 69.90%
//  27258 -- 29735   | ******************************************* 71.51%
//  29736 -- 32213   | ********************************************* 73.06%
//  32214 -- 34691   | ********************************************** 74.60%
//  34692 -- 37169   | ********************************************** 76.15%
//  37170 -- 39647   | ********************************************** 77.69%
//  39648 -- 42125   | ********************************************** 79.07%
//  42126 -- 44603   | ********************************************** 80.43%
//  44604 -- 47081   | ********************************************** 81.63%
//  47082 -- 49559   | ********************************************** 82.73%
//  49560 -- 52037   | ********************************************** 83.76%
//  52038 -- 54515   | ********************************************** 84.79%
//  54516 -- 56993   | ********************************************** 85.80%
//  56994 -- 59471   | ********************************************** 86.66%
//  59472 -- 61949   | ********************************************** 87.52%
//  61950 -- 64427   | ********************************************** 88.38%
//  64428 -- 66905   | ********************************************** 89.23%
//  66906 -- 69383   | ********************************************** 90.01%
//  69384 -- 71861   | ********************************************** 90.70%
//  71862 -- 74339   | ********************************************** 91.38%
//  74340 -- 76817   | ********************************************** 92.07%
//  76818 -- 79295   | ********************************************** 92.75%
//  79296 -- 81773   | ********************************************** 93.44%
//  81774 -- 84251   | ********************************************** 94.08%
//  84252 -- 86729   | ********************************************** 94.59%
//  86730 -- 89207   | ********************************************** 95.11%
//  89208 -- 91685   | ********************************************** 95.62%
//  91686 -- 94163   | ********************************************** 96.14%
//  94164 -- 96641   | ********************************************** 96.65%
//  96642 -- 99119   | ********************************************** 97.17%
//  99120 -- 101597  | ********************************************** 97.57%
//  101598 -- 104075 | ********************************************** 97.91%
//  104076 -- 106553 | ********************************************** 98.26%
//  106554 -- 109031 | ********************************************** 98.60%
//  109032 -- 111509 | ********************************************** 98.94%
//  111510 -- 113987 | ********************************************** 99.29%
//  113988 -- 116465 | ********************************************** 99.48%
//  116466 -- 118943 | ********************************************** 99.65%
//  118944 -- 121421 | ********************************************** 99.83%
//  121422 -- 123942 | ********************************************** 100.00%
//  ---------------- - ------------------------------------------------------------------
```

The number of CellModelGen runs (# CMG runs) displays the number of extracted cell pairs, in this case 123,942. The total bridging length in percent (# Tbl [%]) is a guidance to decide for how many extracted cell pairs a CellModelGen run shall be performed. For example, with 44,603 CellModelGen runs already 80.43 percent of the total bridging length is covered.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

extract_sdc

Context: all contexts

Mode: all modes

Creates an SDC file that can be used for synthesis and static timing analysis (STA).

Usage

extract_sdc [-output_file *output_file*]

Description

This command creates SDC constraints for the current design and its sub-blocks for synthesis and static timing analysis (STA). Refer to "Generating and Using SDC for Tessent Shell Embedded Test IP" in the *Tessent Shell User's Manual* for complete information on using SDC with Tessent Shell.

The extract_sdc command is run automatically as part of extract_icl, therefore you would not typically run this command. If you do run extract_sdc, it requires the design's ICL description, DFT signals and clocks. These are all automatically loaded from the TSDB using the read_design command with the -no_hdl option.

Arguments

- -output_file *output_file*

  An optional switch and file path that enables you to explicitly specify where the SDC file is written. The default location for the SDC is in the Tessent Shell Data Base (TSDB) at the following location:

  `dft_inserted_designs/`*`design_name_design_id.dft_inserted_design`*

Examples

The example runs the extract_sdc command:

INSERTION> extract_sdc

```
// Note: Changing the context to 'patterns -ijtag'.
// ---------------------------------------------------------------------
// Begin ICL elaboration and checking.
// -----------------------------------
// ICL elaboration completed, CPU time=0.05 sec.
// ---------------------------------------------------------------------
// Warning: Input constraint 'C0' on ICL port 'bisr_clk' is removed.
// Warning: Input constraint 'C0' on ICL port 'bisr_shift_en' is removed.
// Warning: Input constraint 'C0' on ICL port 'ijtag_sel' is removed.
// Warning: Input constraint 'C0' on ICL port 'ijtag_ue' is removed.
// Warning: Input constraint 'C1' on ICL port 'ijtag_reset' is removed.
// Writing SDC file:
./tsdb_outdir/dft_inserted_designs/blockB_rtl.dft_inserted_design/
blockB.sdc
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

Delaying Clock Pulses in Shift and Capture to Handle Slow Scan Enable Transitions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

filter_collection

Context: unspecified, all contexts

Mode: all modes

Filters a previously created collection, resulting in a new collection.

Usage

filter_collection ***base_collection expression*** [-regexp] [-nocase]

Description

Filters a previously created collection, resulting in a new collection.

In many cases, the commands that create collections support a -filter option, which performs filtering as part of the collection process. Usually this is a more efficient method of filtering. However, you may want to use the filter_collection command if you are filtering a large collection multiple times with different criteria.

The filter_collection command results either in a new collection that contains a subset of the objects in the input base_collection, or in an empty collection, which indicates that the expression filtered out all elements of the input base_collection.

Arguments

- ***base_collection***

  A required string that specifies the base collection to be filtered. This collection is copied to the resulting collection, and objects are removed if they are evaluated as false by the conditional expression.

- ***expression***

  A required string that specifies an expression with which to filter the collection. The attributes used within the equation must exist for the object types contained in the collection. Otherwise, an error is generated. See section "Attribute Filtering Equation Syntax" on page 3832 for more details on filtering attribute equation format. The =~ and !~ matching operator use simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option.

- -regexp

  An optional switch that directs the matching patterns as a regular expression instead of as a simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a description of the regular expression syntax.

- -nocase

  An optional switch that specifies that the pattern match is case insensitive.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example creates a collection of instances of type design.

```
set var [filter_collection [get_instances *] {type == "design”}]
```

```
{Adder1 Adder2}
```

Example 2

The following example starts with a collection of all instances and filters the collection based on an attribute_equation of user-defined attributes.

```
set var [filter_collection [get_instances] {myType1 || myType2 == “ABC"}]
```

The above example is equivalent to this:

```
set var [get_instances -filter {myType1 || myType2 == “ABC"}]
```

Related Topics

add_to_collection

append_to_collection

compare_collections

copy_collection

foreach_in_collection

index_collection

is_collection

range_collection

remove_from_collection

sizeof_collection

sort_collection

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

find_design_names

Context: all contexts

Mode: setup, analysis

Displays design object hierarchical names matched by an input string, which may include asterisk (*) or question mark (?) wildcard characters.

Usage

find_design_names ***name_pattern*** [-LOcal | -Hier]
[-Design | -NETList | -LIbrary | -All]
[-INStance | -Net | -Pin [INPut | OUtput | INOut | ALLIn | ALLOut] |
[[-Cell | -Module] -LISt_references]]
[{> | >>} *file_pathname*]

Description

Displays design object hierarchical names matched by an input string, which may include asterisk (*) or question mark (?) wildcard characters.

Note

The get_* Tessent Shell commands include additional features, including the capability to search with regular expressions, without regard to case, and so on. Consider using these commands unless you have a specific need for find_design_names.

Arguments

- ***name_pattern***

  A required string, which may include asterisk (*) or question mark (?) wildcard characters.

- -LOcal

  An optional switch that matches wildcard characters within the current hierarchy level.

- -Hier

  An optional switch that matches the regular expression across hierarchy boundaries (for example, a* matches a1/b/c). This is the default.

- -Design

  An optional switch that matches only pathnames to objects at the topmost library cell level.

- -NETList

  An optional switch that matches objects from the top of the design down to the topmost library cell.

- -LIbrary

  An optional switch that matches objects within any level of library cells.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -All

  An optional switch that specifies matches objects at all levels of the design. This is the default.

- -INStance

  An optional switch that matches only instance pathnames. This is the default.

- -Net

  An optional switch that matches only net pathnames.

- -Pin

  An optional switch that matches only pin pathnames (any pin direction). The following optional pin filters restrict which pins are matched:

  INPut — Match only input pin pathnames.

  OUtput — Match only output pin pathnames.

  INOut — Match only bidirectional pin pathnames.

  ALLIn — Match both input and bidirectional pin pathnames.

  ALLOut — Match both output and bidirectional pin pathnames.

- -Cell

  An optional switch that finds all library cell (model) names matching the specified regular expression.

- -Module

  An optional switch that finds all netlist module names matching the specified regular expression.

- -LISt_references

  An optional switch for use with -Cell or -Module that lists the pathnames of all instances in the current design that reference each cell or module matching the regular_expression.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following examples display object pathnames for various input wildcard strings, given a netlist with the following instance hierarchy:

```
/
   tiny_i
      U5
   ret_i
      intreg1_reg_0 ... intreg1_reg_31
      add_20
         U1_0 ... U1_3
      add_30
         U5 ... U12
      mul_18
         U5 ... U868
         FS
            U5 ... U33
      mul_19
         FS
            U5 ... U278
         U5 ... U181
      mul_22
         U5 ... U735
         FS
            U15 ...
```

and assuming the U5 instances all reference the following library cell:

```
model LSR2BUFA(Q, QN, S, R, G, SD, RD) (
   input(S, R, G, SD, RD) ()
   output(Q) (primitive = _buf UP1 (QT, Q);)
   output(QN) (primitive = _buf UP2 (QNT, QN);)
   intern(QT_int) (instance = LSI_LSR2 UD1 (QT_int, S, R, G, SD, RD);)
   intern(QNT_int) (instance = LSI_LSR2N UD2 (QNT_int, S, R, G, SD, RD);)
   intern(QT) (instance = LSI_NOTI UD3 (QT, QT_int);)
   intern(QNT) (instance = LSI_NOTI UD4 (QNT, QNT_int);)
)
```

Example 1

find_design_names /ret_i/add_2* -instance -design -hier

```
// Note: Matched 4 names
/ret_i/add_20/U1_0
/ret_i/add_20/U1_1
/ret_i/add_20/U1_2
/ret_i/add_20/U1_3
```

Example 2

find_design_names /ret_i/add_2* -instance -netlist -hier

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Note: Matched 5 names
/ret_i/add_20
/ret_i/add_20/U1_0
/ret_i/add_20/U1_1
/ret_i/add_20/U1_2
/ret_i/add_20/U1_3
```

Finds instance add_20 under /ret_i/, and also descends the hierarchy to find all netlist instances under /ret_i/add_20/.

Example 3

find_design_names /ret_i/add_2* -inst -netlist -local

```
// Note: Matched 1 names
/ret_i/add_20
```

This example shows that -Local does not descend the hierarchy to find more matches as the previous example does.

Example 4

find_design_names /ret_i/add_2* -ins -design -local

```
// Note: Matched 0 names
```

There are no instances of a library cell under /ret_i/ with instance name starting with add_2.

Example 5

find_design_names /ret_i/*_2? -ins -netlist -local

```
// Note: Matched 2 names
/ret_i/add_20
/ret_i/mul_22
```

Found 2 instances under /ret_i/.

Note

/ret_i/gt_68_2 did not match because the “?” in the wildcard expression requires another character after the “_2”.

Example 6

find_design_names */U5 -inst -design -hier

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Note: Matched 7 names
/tiny_i/U5
/ret_i/add_20/U5
/ret_i/mul_18/U5
/ret_i/mul_18/FS/U5
/ret_i/mul_19/U5
/ret_i/mul_19/FS/U5
/ret_i/mul_22/U5
```

Example 7

find_design_names ret_i/mul*/U5 -ins -des -local

```
// Note: Matched 3 names
/ret_i/mul_18/U5
/ret_i/mul_19/U5
/ret_i/mul_22/U5
```

Example 8

find_design_names ret_i/mul*/U5 -ins -design -hier

```
// Note: Matched 5 names
/ret_i/mul_18/U5
/ret_i/mul_18/FS/U5
/ret_i/mul_19/U5
/ret_i/mul_19/FS/U5
/ret_i/mul_22/U5
```

Example 9

find_design_names ret_i/mul_18/U5* -ins -library -hier

```
// Note: Matched 5 names
/ret_i/mul_18/U5
/ret_i/mul_18/U5/UD1
/ret_i/mul_18/U5/UD2
/ret_i/mul_18/U5/UD3
/ret_i/mul_18/U5/UD4
```

Example 10

find_design_names ret_i/mul*/U5/* -pin output -design -local

```
// Note: Matched 6 names
/ret_i/mul_18/U5/Q
/ret_i/mul_18/U5/QN
/ret_i/mul_19/U5/Q
/ret_i/mul_19/U5/QN
/ret_i/mul_22/U5/Q
/ret_i/mul_22/U5/QN
```

Example 11

find_design_names * -mod -list_references

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Note: Found 3 netlist modules
// Note: Instances of netlist module or_gate
/o1
/o2
// Note: Instances of netlist module pme_caam_RC_CG_MOD_335839
/RC_CG_HIER_INST3921
```

Example 12

find_design_names * -cell -list

```
// Note: Found 6 library cells
// Note: Instances of library cell HD45_SZ_MUX21X2
/mux
// Note: Instances of library cell HD45_SZ_LDLRX2
// Note: Instances of library cell HD45_SH_OR2X2
/o1/foo
/o2/foo
// Note: Instances of library cell HD45_SV_CNHLSX4
/RC_CG_HIER_INST3921/RC_CGIC_INST
// NOte: Instances of library cell HD45_SH_SDFPRQX4
/data_out_reg_0
/data_out_reg_1
/data_out_reg_2
/data_out_reg_3
/data_out_reg_4
/data_out_reg_5
/shadow
/non_scan
```

Related Topics

get_instances

get_modules

get_nets

get_pins

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

foreach_in_collection

Context: unspecified, all contexts

Mode: all modes

Iterates over each element in a collection.

Usage

foreach_in_collection ***itr_var collection*** {***body***}

Description

Iterates over each element in a collection.

You cannot use the Tcl-supplied foreach command to iterate over collections because foreach operates on a Tcl list, and a collection is not a Tcl list but a pointer to an array of data on the C++ side. You can use the get_name_list command to convert the collection of objects into a Tcl list of object names, but you should avoid doing this if the collection is very large or if the object names are not unique, such as port names. For example, a port object is a specific port name on a specific module. Specifying just the name of the port does not indicate which module it belongs to.

You can nest the foreach_in_collection command within other control structures, including foreach_in_collection.

A collection remains in memory for as long as it is referenced by a Tcl variable. A collection is deleted once it is no longer referenced by any Tcl variable. If you make the mistake of using the Tcl command foreach to iterate on a collection, the command has the negative side effect of de-referencing the collection and may destroy it. Always use foreach_in_collection to iterate on the elements of a collection.

Arguments

- ***itr_var***

  A required string that specifies the name of a Tcl variable. During each iteration, itr_var holds a string pointer to a collection of exactly one object.

- ***collection***

  A required value that specifies a collection over which to iterate. It may be the content of a Tcl variable that points to a collection or the result of an introspection command that generates a collection. Note, just like the Tcl foreach command, a snapshot of the collection is taken before the first iteration. This means that even if the append_to_collection command is used to increase the size of the collection as part of the iterations, the foreach_in_collection command does not see the newly added elements. Example 3 shows how to collect new elements into a separate collection and use a Tcl while command to repeat the loop until the collection is truly empty.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ***body***

  A required string that specifies a script to run per iteration. The entire body must be enclosed within a set of braces.

Examples

Example 1

The following example connects a set of pins that has a special attribute called myAtt set to True to a net that has the same name as the pin. If not already present, the net is created. Note that the example assumes that all pins are scalar objects. Extra code would be needed to deal with bused pins and create the entire bused nets with one command. The is_bus and other built-in bus attributes are available to do this efficiently. Note how the lindex command converts the return value of the get_attribute_value_list command into a string so that it can be used to construct the net name using ${parent}/${leaf}.

```
foreach_in_collection pin [get_pins -filter “myAtt”] {
    set leaf [lindex [get_attribute_value_list $pin -name leaf_name] 0]
    set parent [lindex [get_attribute_value_list $pin -name parent_instance] 0]
    if {[sizeof_collection [get_nets ${parent}/${leaf} -silent]] = 0} {
        create_net ${parent}/${leaf}
    }
    create_connections $pin ${parent}/${leaf}
}
```

Note how filtering based on the value of an attribute is performed to reduce the size of the collection on which to iterate. The user could have chosen to loop on all pins and check inside the loop if the myAtt attribute is set to true, but this is very inefficient because filtering is done in C++ as opposed to looping in Tcl.

```
#inefficient use of looping. Best to filter collection before looping on
#it as shown above.


foreach_in_collection pin [get_pins] {
    if {[get_attribute_value_list $pin -name myAtt]} {
        set leaf [lindex [get_attribute_value_list $pin -name leaf_name] 0]
        set parent [lindex [get_attribute_value_list $pin -name parent_instance] 0]
        if {[sizeof_collection [get_nets ${parent}/${leaf} -silent]] = 0} {
            create_net ${parent}/${leaf}
        }
        create_connections $pin ${parent}/${leaf}
    }
}
```

Example 2

The following example loops on a collection of pins and deletes them. The same pin object is added twice into the collection being looped on to illustrate the fact that obsolete objects are still looped on and that a -silent switch must be used so that they are ignored by the delete_pin command. Without the -silent option, an error would be generated the second time the pin object u1/P2 was looped on.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set my_pins [get_pins {u1/P2 u1/P*}]
puts "my_pins are [get_name_list $my_pins]"
foreach_in_collection pin $my_pins {
   delete_pin $pin -silent
}
```

```
my pins are u1/P2 u1/P1 u1/P2 u1/P3
```

Example 3

The following example shows how to iterate on a collection of objects until no objects are left to process. The procedure called "proc_that_returns_new_objects" is called on each object and may return new objects to process. The "append_to_collection col" command cannot be used to process the new objects because the foreach_in_collection command took a snapshot of "col" before the first iteration. Instead, the new objects are collected into a collection variable called "newCol" and it is assigned back to "col" after the foreach_in_collection command is done. The loop is repeated for as long as "newCol" is not empty after the foreach_in_collection command has completed.

```
set col [<start collection>]
while {[sizeof_collection $col] > 0} {
  set newCol {}
  foreach_in_collection obj $col {
    append_to_collection newCol [proc_that_returns_new_objects $obj]
  }
  set col $newCol
}
```

Related Topics

add_to_collection

append_to_collection

compare_collections

copy_collection

filter_collection

get_common_parent_instance

get_modules

get_nets

index_collection

is_collection

remove_from_collection

sizeof_collection

sort_collection

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

format_dictionary

Context: all including the unspecified one

Mode: all

Generates a Tcl dictionary in a human-readable format.

Usage

format_dictionary ***dictionary*** [-left_indent *left_indent*]
[-keys_that_are_strings *keys_that_are_strings*]
[-keys_that_are_lists *keys_that_are_lists*]
[-list_elements_per_line {1 | all}]
[-variable_name *variable_name*]
[-banner *string*]
[-preserve_collection_pointers]

Description

Generates a Tcl dictionary in a human-readable format. The dictionary consists of entries on separate lines with child elements indented by two white spaces relative to their parents entries.

By default, elements are considered sub-dictionaries if they contain an even number of elements. Otherwise, they are considered lists.

You can send the content of the dictionary to a file by using the -variable_name option.

Arguments

- ***dictionary***

  A string that defines a Tcl dictionary similar to the one created using the Tcl dict command or the one returned by the get_design_sources -file_dictionary command.

- -left_indent *left_indent*

  An optional switch and value pair that controls the indentation of the dictionary, where the value is a positive integer. When specified, the tool adjusts the entire dictionary left by the specified *left_indent* value, maintaining child element indentation.

- -keys_that_are_strings *keys_that_are_strings*

  An optional switch and value pair that specifies to treat the specified leaf names of keys as strings. By default, if a string contains an even number of words, the tool treats it as a sub-dictionary. If a string contains an odd number of words, the tool considers it a list that you can control with the -list_elements_per_line option. When you mark a key as holding the value of a string, the string is shown inside quotation marks.

- -keys_that_are_lists *keys_that_are_lists*

  An optional switch and value pair that specifies to treat the specified leaf names of keys as lists. By default, if a list contains an even number of elements, the tool treats it as a sub-dictionary. When you mark a key as holding the value of a list, the list is shown inside

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

braces ({}). You can control the number of elements to display per line with the -list_elements_per_line option.

- -list_elements_per_line 1 | all

  An optional switch that specifies the number of elements from a list to display on each line. By default, the entire list displays on a single line. Specify 1 to display one element per line. The tool treats the value of a key as a list when it contains an odd number of elements or it is specified by the -keys_that_are_lists option.

- -variable_name *variable_name*

  An optional switch that specifies the name of a Tcl variable. When specified, the formatted dictionary is included in a "set *variable_name*{}" wrapper, which the tool displays in the transcript. When you specify *variable_name* with a redirection symbol (>) and a filename, the tool sends the dictionary to the specified file.

- -banner *string*

  An optional switch and string pair that specifies a banner text that can be passed as a string, which can contain newlines.

- -preserve_collection_pointers

  An optional switch that preserves the collection indexes (@<integer>). By default, a value that is a collection is considered a list.

Examples

The following example uses the Tcl dict command to create a Tcl dictionary then shows how it looks raw versus how it looks when formated with the format_dictionary command.

The last invocation shows how to use the command to export the dictionary to file in a human-readable manner.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
> dict set mydict square1 color red
> dict set mydict square1 dimensions width 3
> dict set mydict square1 dimensions height 5
> dict set mydict square1 usage "for learning"
> dict set mydict square1 partners {round triangle line point}
> puts $mydict
square1 {color red dimensions {width 3 height 5} usage {for learning}
partners {round triangle line point}}
> puts [format_dictionary $mydict]
square1 {
  color red
  dimensions {
    width 3
    height 5
  }
  usage {
    for learning
  }
  partners {
    round triangle
    line point
  }
}

> puts [format_dictionary $mydict -keys_that_are_list partners]
square1 {
  color red
  dimensions {
    width 3
    height 5
  }
  usage {
    for learning
  }
  partners {round triangle line point}
}
> puts [format_dictionary $mydict -keys_that_are_list partners
-list_elements_per_line 1]
square1 {
  color red
  dimensions {
    width 3
    height 5
  }
  usage {
    for learning
  }
  partners {
    round
    triangle
    line
    point
  }
}
> format_dictionary $mydict -keys_that_are_list partners
-list_elements_per_line 1 -variable_name mydict > mydict.txt
> cat mydict.txt
set mydict {
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
    square1 {
      color red
      dimensions {
        width 3
        height 5
      }
      usage {
        for learning
      }
      partners {
        round
        triangle
        line
        point
      }
    }
  }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 5
Command Dictionary (G-Q)

The Command Dictionary describes the application commands for Tessent Shell. For quick reference, the commands appear alphabetically with each beginning on a separate page.

The command usage line syntax conventions are common to all products documented in this manual. See "Conventions for Command Line Syntax" on page 97.

Command Descriptions **884**
get_attribute_list 898
get_attribute_option 901
get_attribute_value_list 903
get_attributed_objects 907
get_auxiliary_pins 909
get_boundary_scan_port_option 912
get_bsdl_extraction_option 914
get_bundle_objects 916
get_capture_timing_option 919
get_burnin_option 923
get_cdp_test_data 924
get_cdp_test_list 926
get_clock_option 927
get_clocks 932
get_common_parent_instance 935
get_config_elements 936
get_config_messages 940
get_config_value 943
get_context 958
get_current_design 962
get_current_mode 964
get_current_physical_block 965
get_current_silicon_insight_setup 966
get_current_simulation_context 967
get_default_ssn_datapath_configuration 968
get_defaults_value 969
get_design_level 971
get_design_objects 972
get_design_sources 975
get_dfm_rules 977
get_dft_cell 979
get_dft_cell_selection_mapping 989

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_dft_info_dictionary 990
get_dft_signal 997
get_dft_signals_option 1004
get_dft_specification_requirement 1005
get_drc_handling 1007
get_equivalent_editable_node 1008
get_fanins 1010
get_fanouts 1014
get_fault_type 1018
get_flat_model_option 1019
get_gate_pins 1020
get_icl_extraction_options 1026
get_icl_fanins 1027
get_icl_fanins_in_module 1029
get_icl_fanouts 1031
get_icl_fanouts_in_module 1033
get_icl_instances 1035
get_icl_modules 1040
get_icl_module_parameter_list 1043
get_icl_module_parameter_value 1044
get_icl_network 1046
get_icl_objects 1048
get_icl_pins 1055
get_icl_ports 1059
get_icl_scan_interface_chain_list 1062
get_icl_scan_interface_list 1063
get_icl_scan_interface_port_list 1065
get_icl_scan_mux 1068
get_icl_scope 1070
get_icl_ssn_datapath_list 1072
get_icl_ssn_datapath_ports 1073
get_iclock_list 1075
get_iclock_option 1077
get_ijtag_instances 1081
get_ijtag_instance_option 1083
get_ijtag_retargeting_options 1085
get_input_constraints 1088
get_insertion_option 1092
get_insert_test_logic_option 1095
get_instance_polygons 1098
get_instances 1099
get_instrument_dictionary 1106
get_instrument_parent_icl_module_list 1112
get_iproc_argument_default 1113
get_iproc_argument_list 1115

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_iproc_body 1116
get_iproc_list 1117
get_iproc_option 1118
get_layout_core_instance 1120
get_license_queue_timeout 1121
get_load_unload_timing_option 1122
get_loadboard_loopback_option 1127
get_logfile 1128
get_logical_library_list 1129
get_lvx_option 1130
get_master_module_name 1131
get_memory_instances 1132
get_memory_instance_option 1135
get_module_matching_options 1138
get_module_parameter_list 1142
get_module_parameter_value 1144
get_modules 1146
get_multiprocessing_option 1153
get_name_list 1155
get_nets 1159
get_net_name_list 1164
get_open_pattern_set 1165
get_pattern_cycle_count 1166
get_pattern_set_data 1167
get_pattern_set_list 1177
get_pattern_set_option 1178
get_pin_name_list 1180
get_pins 1181
get_port_name_list 1192
get_ports 1193
get_procfile_name 1203
get_read_verilog_option 1204
get_resource 1205
get_run_synthesis_options 1207
get_scan_chain_families 1209
get_scan_chain_option 1210
get_scan_elements 1211
get_scan_modes 1218
get_scratch_directory 1219
get_silicon_insight_job_status 1220
get_silicon_insight_option 1221
get_simdut_data 1222
get_simulation_context_list 1224
get_simulation_library_sources 1226
get_simulation_option 1228

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_simulation_value_list 1230
get_single_attribute_value 1232
get_single_name 1234
get_ssn_datapath_option. 1237
get_ssn_option 1239
get_static_dft_signal_icall 1240
get_synchronous_clock_groups 1242
get_system_mode 1243
get_tcl_shell_option 1244
get_test_end_icall_list 1245
get_test_point_analysis_options 1246
get_test_point_insertion_options 1250
get_test_point_sharing_restrictions 1252
get_test_point_type. 1254
get_test_points 1255
get_test_setup_icall_list 1260
get_testbench_simulation_options 1261
get_tester_option. 1263
get_timeplate_list 1264
get_tool_info. 1265
get_tool_option 1267
get_trace_flat_model_option 1268
get_transcript_style. 1271
get_tsdb_info 1273
get_tsdb_list 1279
get_tsdb_option. 1280
get_tsdb_output_directory 1281
get_validated_objects 1282
get_write_patterns_options. 1285
get_xclock_handling. 1288
get_xy_net. 1289
get_xy_instance 1290
help 1291
history 1293
iApply 1295
iCall. 1298
iClock 1302
iClockOverride 1304
iComparePort 1307
identify_redundant_faults 1310
iForcePort 1314
iMerge. 1318
import_clocks 1320
import_dfm 1321
import_patterns_from_svf 1323

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

import_scan_mode 1326
index_collection 1329
iNote 1331
insert_test_logic 1337
intercept_connection 1340
iOverrideScanInterface 1348
iPDLLevel 1351
iPrefix 1352
iProc 1354
iProcsForModule 1356
iPulseClock 1358
iRead 1360
iRelease 1362
iReset 1363
iResumePdlRetargeting 1365
iRunLoop 1367
is_collection 1370
is_valid_object_name 1372
iScan 1373
isolate_icl_node 1377
iState 1378
iSuspendPdlRetargeting 1381
iTake 1385
iTopProc 1387
iUseProcNameSpace 1389
iWrite 1391
launch_sid_tester 1393
lock_current_registration 1395
macrotest 1396
map_tsdb_container_paths 1403
merge_cdp 1405
migrate_layout 1406
move_config_element 1407
move_connections 1412
no_transcript 1420
open_layout 1423
open_pattern_set 1427
open_tsdb 1438
open_visualizer 1440
order_patterns 1444
printenv 1448
process_atpg_failures 1449
process_dft_specification 1451
process_patterns_specification 1456
process_top_module_connections 1461

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Command Descriptions

This section describes the Tessent Shell commands alphabetically, G through Q. The beginning of each command description shows the contexts and modes that support the command and provides the following information:

Context: Lists each context and sub-context that supports the command. “All contexts” means that the command is supported in all available contexts. “Unspecified” means that you do not have to set a context to use the command.

Mode: Lists each mode that supports the command. “All modes” means that the command is supported in all available modes. In some cases, a particular mode supports the command only within certain contexts, and this is noted in parentheses after the mode type.

Table 5-1. Commands G-Q

Command	Description
get_attribute_list	Returns an alphabetically sorted Tcl list of attribute names.
get_attribute_option	Retrieves the current setting of an attribute's option.
get_attribute_value_list	Retrieves the value of an attribute on the specified design objects.
get_attributed_objects	Retrieves the complete collection of objects matching one of the specified object type lists and having one of the attributes found in the *attribute_name_list* specified on the object.
get_auxiliary_pins	Returns the input or output data or enable auxiliary pins associated to ports
get_boundary_scan_port_option	Returns the cell_options defined on the specified port, or the specified pin_order_file name, or the pad_io_ports collection.
get_bsdl_extraction_option	Returns a value for a BSDL attribute previously set with the set_bsdl_extraction_options command.
get_bundle_objects	Returns a collection of bundle/leaf objects in the current design specified by the name_patterns list in conjunction with the specified options. Those bundle (leaf in case of descendants) objects are ancestors or descendants of the objects that match the specified *name_patterns* depending upon the specified options.
get_capture_timing_option	Returns specific capture timing options specified with the set_capture_timing_options command.
get_burnin_option	Returns the current setting of an option specified with the set_burnin_options command.
get_cdp_test_data	Returns requested information for a specific CDP test.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-1. Commands G-Q (cont.)

Command	Description
get_cdp_test_list	Returns a Tcl list of tests currently available in the CDP.
get_clock_option	Returns the setting of an option specified with the add_clocks command.
get_clocks	Returns a collection of user clocks based on filters you specify.
get_common_parent_instance	Returns the common ancestor of all instances, pins, or nets found in ***object_spec***.
get_config_elements	Returns a collection of configuration elements or a count of configuration elements when the -count option is used.
get_config_messages	Returns a list of message strings attached to a configuration element.
get_config_value	Returns a value associated with a configuration element based on the specified option.
get_context	Introspects the current context.
get_current_design	Returns a collection of one element that specifies the top-level design when previously set.
get_current_mode	Returns the current test mode, as specified by the set_current_mode command.
get_current_physical_block	Returns the current physical block specified with the set_current_physical_block command.
get_current_silicon_insight_setup	Returns the currently selected SiliconInsight setup as specified in the TSI setup spec.
get_current_simulation_context	Returns the name of the current simulation context.
get_default_ssn_datapath_configuration	Retrieves an override setting of an ICL primary target on a SSN datapath.
get_defaults_value	Obtains the default value of a property as specified in one of the DefaultsSpecification(company | user | group)/ DftSpecification or DefaultsSpecification(company | user | group)/PatternsSpecification wrappers.
get_design_level	Returns the design level previously defined with the set_design_level command.
get_design_objects	Returns a collection of design objects associated to the object_ids you supply.
get_design_sources	Returns the pathnames or file extensions previously specified with the set_design_sources command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-1. Commands G-Q (cont.)

Command	Description
get_dfm_rules	Returns a TCL list of the DFM rules contained in the layout database.
get_dft_cell	Retrieves the named sets of cells to use when test hardware must be inserted (as a Tcl list of the currently available cell selection names), or a particular cell given the name and required cell function of a specified set of cells.
get_dft_cell_selection_mapping	Returns the library cell selection associated to the specified power domain.
get_dft_info_dictionary	Creates a Tcl dictionary containing the information about the DFT inserted in the RTL that must be considered during scan insertion.
get_dft_signal	Obtains the source of a DFT signal or various aspects about the DFT signal.
get_dft_signals_option	Returns the current setting specified by the set_dft_signals_options command.
get_dft_specification_requirement	Returns the requirements set using the set_dft_specification_requirements command.
get_drc_handling	Retrieves the DRC handling status for the DRC with the specified name.
get_equivalent_editable_node	Checks if a node is editable. If the node is not editable, then returns the node's editable equivalent node if it exists.
get_fanins	Returns a collection of all requested objects found in the fan-in of the specified pin, net, or port object.
get_fanouts	Returns a collection of all requested objects found in the fanout of the specified pin, net, or port object.
get_fault_type	Returns the current fault model type, as previously specified by the set_fault_type command.
get_flat_model_option	Retrieves the current setting of the specified flat model option.
get_gate_pins	Returns a collection of gate_pin objects found below the current design in the flat model.
get_icl_extraction_options	Provides access to the settings specified by the set_icl_extraction_options command.
get_icl_fanins	Returns a collection of ICL objects found in the fan-in of the specified ICL objects.
get_icl_fanins_in_module	Returns a collection of objects found in the fan-in of the specified icl_pin_in_module or icl_port objects.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-1. Commands G-Q (cont.)

Command	Description
get_icl_fanouts	Returns a collection of ICL objects found in the fanout of the specified ICL objects.
get_icl_fanouts_in_module	Returns a collection of objects found in the fanout of the specified icl_pin_in_module or icl_port objects.
get_icl_instances	Returns a collection of all ICL instances instantiated relative to the current design that match the specified <*name_patterns> list or -filter expression.
get_icl_modules	Returns a collection of all ICL modules that match the specified name_patterns list or -filter expression.
get_icl_module_parameter_list	Returns the list of parameters in an ICL module.
get_icl_module_parameter_value	Returns the value of the parameter on the specified module and indicates whether it is the default value or is overridden.
get_icl_network	Context: patterns -silicon_insight -ijtag
get_icl_objects	Returns a collection of ICL object matching the requested criteria.
get_icl_pins	Returns a collection of all hierarchical ICL pins instantiated relative to the current design that match the specified name_patterns list or -filter expression.
get_icl_ports	Returns a collection of all ICL ports that match the specified name_patterns list or -filter expression.
get_icl_scan_interface_chain_list	Returns the names of the explicitly defined scan chains for the specified client scan interface in an existing ICL top module, or the names of the explicitly defined scan chains for the specified client scan interface that are created in the new ICL top module during ICL extraction.
get_icl_scan_interface_list	Returns the names of the scan interfaces in an existing ICL top module, if the ICL top module exists.
get_icl_scan_interface_port_list	Returns the names of the ports for the specified scan interface in an existing ICL top module, or the names of the ports for the specified scan interface that are created in the new ICL top module during ICL extraction.
get_icl_scan_mux	Context: patterns -silicon_insight -ijtag
get_icl_scope	Introspects the current ICL scope.
get_icl_ssn_datapath_list	Returns the SSN datapath specifications added with the add_icl_ssn_datapaths, or returns the list of SSN datapaths in the ICL module of the current design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-1. Commands G-Q (cont.)

Command	Description
get_icl_ssn_datapath_ports	Returns the SSN datapath specifications specified with the set_icl_ssn_datapath_ports command, or returns the collection of the SSN datapaths ports of a particular type in the ICL module of the current design.
get_iclock_list	Returns a list of ICL port and pin names on which you have issued an iClock command.
get_iclock_option	Returns the effective or specified source, source type, frequency multiplier, frequency divider, clock period and other information related to the iClock or iClockOverride commands for the specified icl_port or pin_name.
get_ijtag_instances	Returns a collection of instances for which there is a matched ICL module.
get_ijtag_instance_option	Returns the value of the selected option for the specified Ijtag instance.
get_ijtag_retargeting_options	Returns the value of the specified option of the previous set_ijtag_retargeting_options command or its default value if the set_ijtag_retargeting_options command was not run.
get_input_constraints	Returns the present constraint status of primary input ports or pseudo ports.
get_insertion_option	Returns the value of any insertion option that is specifiable with the set_insertion_options command.
get_insert_test_logic_option	Retrieves a prefix/infix value that is used when logic is created by the insert_test_logic command.
get_instance_polygons	Context: patterns -scan_diagnosis
get_instances	Returns a collection of all instances instantiated relative to the current design that match the specified *name_patterns* list, subject to filtering imposed by the specified options.
get_instrument_dictionary	Returns the Tcl dictionary created by the instrument processed by process_dft_specification or process_patterns_specification.
get_instrument_parent_icl_module_list	Returns the parent or grandparent module names for ICL instance module names that match the specified *instrument_module_name*.
get_iproc_argument_default	Returns the default value for the specified arg_name of the specified proc_name attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by module_name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-1. Commands G-Q (cont.)

Command	Description
get_iproc_argument_list	Returns the list of argument names for the specified iProc attached to the ICL module that was specified by the last iProcsForModule command, or to the ICL module specified by module_name.
get_iproc_body	Returns the body for the specified iProc attached to the ICL module that was specified by the last iProcsForModule command, or to the ICL module specified by module_name.
get_iproc_list	Returns a Tcl list of iProcs attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by module_name.
get_iproc_option	Returns properties of an iProc or iTopProc specification.
get_layout_core_instance	Returns the current layout core instance that was set with the set_layout_core_instance command.
get_license_queue_timeout	Returns the value specified by the previous set_license_queue_timeout command.
get_load_unload_timing_option	Returns the value for load/unload timing options set with the set_load_unload_timing_options command for an icl_module, a module, the current_design, or the current physical block.
get_loadboard_loopback_option	Returns the settings of options specified with the add_loadboard_loopback pairs command.
get_logfile	Returns the name of and pathway to the current log file.
get_logical_library_list	Returns a Tcl list with all logical design library names defined with the set_logical_design_libraries command.
get_lvx_option	Returns the current setting of an option specified with the set_lvx_options command.
get_master_module_name	Returns a string with the master_name attribute value for a design or icl module.
get_memory_instances	Returns a collection of the instances for which there is a matched TCD Memory description.
get_memory_instance_option	Returns the value of an option on a specific memory instance, or returns the value of an option that applies to all memory instances.
get_module_matching_options	Returns the settings specified with the set_module_matching_options and the “set_insertion_options -edited_module_prefix” commands for module matching.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-1. Commands G-Q (cont.)

Command	Description
get_module_parameter_list	Returns the list of parameters in a module.
get_module_parameter_value	Returns the value of the parameter on the specified module object.
get_modules	Returns a collection of all modules that match the specified *name_patterns* list.
get_multiprocessing_option	Returns the value of a single specified variable previously set with the set_multiprocessing_options command.
get_name_list	Returns the names of the specified objects, or the names of the name list with simplified escaped identifiers, if possible.
get_nets	Returns a collection of all hierarchical nets instantiated relative to the current design that match the specified name_patterns list, subject to filtering imposed by the specified options.
get_net_name_list	Returns the design names of the associated port nets for a collection of ICL pins or ports.
get_open_pattern_set	Returns the name of the currently open pattern set.
get_pattern_cycle_count	Reports the total number of cycles in the currently loaded WGL or STIL test pattern set.
get_pattern_set_data	Returns requested details of the internal representation of the specified pattern set.
get_pattern_set_list	Returns a Tcl list of all existing and closed pattern sets in the order of their creation.
get_pattern_set_option	Returns the value of the requested option for the specified pattern_set_name.
get_pin_name_list	Returns the names of the associated design pins for a given collection of ICL pins.
get_pins	Returns a collection of all hierarchical pins instantiated relative to the current design that match the specified name_patterns list, subject to filtering by other options.
get_port_name_list	Returns the names of the associated design ports for a collection of ICL ports.
get_ports	Returns a collection of all hierarchical ports on the current design or a given module that match the specified filters.
get_procfile_name	Returns the name of the procfile that was defined using the set_profile_name command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-1. Commands G-Q (cont.)

Command	Description
get_read_verilog_option	Returns the options specified with the set_read_verilog_options command.
get_resource	Returns true when all resources in the specified resource_list are available.
get_run_synthesis_options	Returns the values for options that can be set for the run_synthesis command.
get_scan_chain_families	Gets scan chain families with certain characteristics.
get_scan_chain_option	Returns the setting of an option specified in the set_scan_chain_options command.
get_scan_elements	Returns a collection of scan elements that the tool has identified in the design.
get_scan_modes	Returns the scan modes with certain characteristics.
get_scratch_directory	Returns the absolute path to the *scratch* directory.
get_silicon_insight_job_status	Returns the status of diagnosis jobs that are running in the background.
get_silicon_insight_option	Returns current settings of options specified in the set_silicon_insight_option command.
get_simdut_data	Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight
get_simulation_context_list	Returns the available simulation contexts in a Tcl list.
get_simulation_library_sources	Returns the pathnames or file extensions previously specified with the set_simulation_library_sources command.
get_simulation_option	Returns the values for options that can be set for the set_simulation_options command.
get_simulation_value_list	Returns a Tcl list of the simulation values on the specified gate_pin objects.
get_single_attribute_value	Retrieves the value of an attribute on the specified design object.
get_single_name	Returns a string with the name of the element specified by the *object_spec* or the *name* arguments with simplified escaped identifiers, if possible.
get_ssn_datapath_option	Returns settings specified using the set_ssn_datapath_options command.
get_ssn_option	Returns the status of SSN in the current physical block or design specified with the set_ssn_options command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-1. Commands G-Q (cont.)

Command	Description
get_static_dft_signal_icall	Returns the iCall corresponding to the setting specified with the set_static_dft_signal_values command.
get_synchronous_clock_groups	Returns a Tcl list of collection handles. Each collection is a synchronous group of clock objects.
get_system_mode	Returns the system mode.
get_tcl_shell_option	Introspects the Tcl shell options as specified with the set_tcl_shell_options command.
get_test_end_icall_list	Returns the iCalls added to the test_end procedure by the set_test_end_icall command. Each iCall is a list containing the iProc and its arguments.
get_test_point_analysis_options	Returns the current setting of an option specified with the set_test_point_analysis_options command. If you have not specified the option, the tool returns the default value of that option.
get_test_point_insertion_options	Returns the current setting of an option specified with the set_test_point_insertion_options command. If you have not specified the option, the tool returns the default value of that option.
get_test_point_sharing_restrictions	Report current test point sharing restrictions.
get_test_point_type	Reports the current test point type as specified by the set_test_point_types command.
get_test_points	Returns a collection of all test points, or a collection of test points matching an optional Tcl list of specified test point attributes.
get_test_setup_icall_list	Returns the iCalls added to the test_setup procedure by the set_test_setup_icall command. Each iCall is a list containing the iProc and its arguments.
get_testbench_simulation_options	Returns the value of a given option that may have been configured using the set_testbench_simulation_options command.
get_tester_option	Enables you to confirm the value of an individual tester-specific option that controls how the tool writes tester patterns when you use SSN.
get_timeplate_list	Returns a list of the currently defined timeplates.
get_tool_info	Returns information about the current tool and the current run as a Tcl result.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-1. Commands G-Q (cont.)

Command	Description
get_tool_option	Returns certain Tessent Shell options from the set_tool_options command.
get_trace_flat_model_option	Introspects the default behavior of the trace_flat_model command.
get_transcript_style	Returns the keyword that specifies the style in which the tool transcribes commands.
get_tsdb_info	An introspection command used to extract information stored in the *.tsdb_info* files found in the *dft_inserted_design* directories of the TSDB directories.
get_tsdb_list	Returns a Tcl list of directory paths, relative to the current working directory, corresponding to all opened TSDB directories.
get_tsdb_option	Retrieves the current setting of an option for referencing local and external Tessent Shell Database (TSDB) paths in design source dictionary files.
get_tsdb_output_directory	Returns the directory used by the process_dft_specification command—the directory previously set by the set_tsdb_output_directory command.
get_validated_objects	Validates the values supplied to arguments or switches of a command created with the register_tcl_command command.
get_write_patterns_options	Returns introspection results from either port lists or vector callbacks.
get_xclock_handling	Returns the options specified with the set_xclock_handling command.
get_xy_net	Context: patterns -scan_diagnosis, patterns -silicon_insight
get_xy_instance	Context: patterns -scan_diagnosis, patterns -silicon_insight
help	Lists commands and associated usage syntax.
history	Displays a list of previously run commands.
iApply	Triggers the retargeting of all queued iRead and iWrite commands.
iCall	Calls an iProc registered against the ICL module associated with the specified *effective_icl_instance_path*.
iClock	Checks whether a controlling clock path from a valid clock source to the specified clock port exists.
iClockOverride	Models how the functional clocking has been programmed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-1. Commands G-Q (cont.)

Command	Description
iComparePort	Enables you to compare a value on an arbitrary primary output port or bidirectional port inside an ijtag pattern set.
identify_redundant_faults	Identifies redundant faults among current faults whose redundancy has not yet been determined.
iForcePort	Enables you to force a constant value to an arbitrary primary input port or bidirectional port inside an ijtag pattern set.
iMerge	Specifies the beginning and the end of a set of iCall commands that are meant to be processed in parallel, as much as possible, in the final representation of the test patterns.
import_clocks	Imports functional clocks defined during the pre-DFT design rule checking.
import_dfm	Imports a Calibre-compatible results database (RDB) file into a violations database (VDB).
import_patterns_from_svf	Enables you to integrate the actions described in the SVF file into the currently open pattern set.
import_scan_mode	Imports configuration settings from tcd_scan.
index_collection	Extracts a single object at a specified index in a collection and creates a new collection containing only that object.
iNote	Inserts a note in the opened pattern set.
insert_test_logic	Inserts the test structures you define into the netlist and stitches up the scan chains to increase the design's testability.
intercept_connection	A command that uses the get_dft_cell command to obtain a cell with the specified function name and uses it to intercept a connection to a pin, port, or net.
iOverrideScanInterface	Imposes user-specified behavior on the operation of a ScanInterface.
iPDLLevel	Identifies the level and version of PDL commands.
iPrefix	Sets the iPrefix path that is used to compute the effective_icl_instance_path for the iCall command and other PDL commands.
iProc	Specifies a PDL procedure that can run later when referenced by the iCall command.
iProcsForModule	Specifies the ICL module for which subsequent iProc commands refer to.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-1. Commands G-Q (cont.)

Command	Description
iPulseClock	Starts or restarts a synchronous clock that previously has been forced to its off-state by means of the iForcePort command.
iRead	Adds a read operation to the command queue to be solved by the next iApply command.
iRelease	Releases a resource previously taken by means of iTake.
iReset	Adds a sequence of actions to the current pattern set that is required to set the ICL network into the reset state.
iResumePdlRetargeting	Returns the IJTAG retargeter to the normal mode of operation after a previous run of the iSuspendPdlRetargeting command.
iRunLoop	Creates a vector loop of a given duration.
is_collection	Indicates whether string is a collection handle.
is_valid_object_name	Indicates whether a string is a valid object name.
iScan	Adds a scan operation to the command queue for the next iApply command to solve.
isolate_icl_node	Context: patterns -silicon_insight -ijtag
iState	Manipulates the state of the internal model for the IJTAG retargeter directly, without creating any additional vectors in the currently opened pattern set.
iSuspendPdlRetargeting	Enables the ability to force ScanInterface ports to a particular value using iForcePort and iPulseClock commands. Enables the creation of delays using an iRunLoop command. Facilitates applying stimulus to the ScanInterface and other ports of the current design.
iTake	Takes ownership of ICL resources to prevent the retargeting software from altering their states during the processing of subsequent iApply commands or during the processing of concurrent iProcs in iMerge parallelization.
iTopProc	Specifies a PDL procedure that can run later when referenced by the iCall command, when its associated ICL module is the top level module.
iUseProcNameSpace	Specifies a name space that is used for all subsequent iCalls made within the iProc in which the iUseProcNameSpace command appears.
iWrite	Adds a write operation to the command queue to be solved by the next iApply command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-1. Commands G-Q (cont.)

Command	Description
launch_sid_tester	Directs Tessent Shell to launch the SID tester process on a local host using the selected_tester and the port number as specified in the configuration data file.
lock_current_registration	Locks all previously registered commands and attributes preventing them from being unregistered and modified.
macrotest	Generates manufacturing test patterns for embedded logic and memories referred to as macros.
map_tsdb_container_paths	Maps Tessent Shell Database (TSDB) container relative or absolute file paths to logical names and logical names to absolute paths.
merge_cdp	Merges legacy LV flow and Tessent-Shell-based ESOE CDPs into the current Tessent Shell-based CDP.
migrate_layout	Generates an LDB directory hierarchy that contains the specified preexisting LDB.
move_config_element	Moves a configuration element from one location to another.
move_connections	Moves connections from pin or net objects to pin objects.
no_transcript	Turns off transcripting for the specified commands.
open_layout	Opens an existing Tessent Diagnosis tool-compatible Layout Database directory.
open_pattern_set	Opens an empty named pattern set and makes it ready to be populated with the specified PDL commands.
open_tsdb	Makes the contents of the TSDB directory visible to the tool. The specified TSDB output directory is automatically opened.
open_visualizer	Opens the Tessent Visualizer window.
order_patterns	Reorders the internal test pattern set as specified.
printenv	Prints out the values of the Linux environment variables.
process_atpg_failures	Interprets ATPG failure log files generated on ATE. Produces flop failure files for a given characterization and debug package (CDP) and test.
process_dft_specification	Validates and processes the content contained in a DftSpecification wrapper.
process_patterns_specification	Validates and processes the content of the PatternsSpecification(*design_name*, *design_id*, *pattern_id*) wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-1. Commands G-Q (cont.)

Command	Description
process_top_module_connections	Validates and processes the content contained in a TopModuleConnections wrapper to automatically create the DFT ports on tiled modules and connect them at the chip level.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_attribute_list

Context: unspecified, all contexts

Mode: all modes

Returns an alphabetically sorted Tcl list of attribute names.

Usage

Usage 1: List attributes specified on a group of objects

get_attribute_list ***obj_spec*** [-predefined | -user_defined] [-silent]

Usage 2: List attributes registered for a groups of object types

get_attribute_list **-object_types** ***type_list*** [-predefined | -user_defined] [-silent]

Usage 3: List all registered attributes

get_attribute_list [-predefined | -user_defined] [-silent]

Description

Returns an alphabetically sorted Tcl list of attribute names.

- **Usage 1** — Returns the attributes set on any object found in ***obj_spec***.

  The attributes on an object inherited from an associated object are reported. For example, an attribute registered for the module object type and set on a given module object are reported in the list of attributes for the instances of that module. By default, attributes that have not been set on the objects are not listed unless the attribute was registered with the -show_default option.

  If an object in ***obj_spec*** does not exist, an error message displays unless the -silent switch is specified.

- **Usage 2** — Returns the attributes registered for all specified object types.

  These do not include the inherited attributes from one object type to another such as the module attribute inherited by the associated instances.

- **Usage 3** —Returns all registered attributes.

In all three usages, only predefined attributes are returned when the -predefined option is specified. Only user-defined attributes are returned when the -user_defined option is specified.

Arguments

- ***obj_spec***

  A required value used in Usage 1 that specifies an object name, a Tcl list of object names, or a collection of objects.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -object_types *type_list*

  A required switch and value pair used in Usage 2 that enumerates the object types for which the attribute list is requested. The currently supported object types are module, port, instance, pin, net, and icl_module. The list of supported object types is augmented in successive releases, and, eventually, all object types are available for introspection.

- -predefined | -user_defined

  An optional switch that constrains the returned list to only pre-defined attribute names or only user-defined attribute names. If this argument is not specified, by default no constraints are applied and both pre-defined and user-defined attributes are returned.

- -silent

  An optional switch that specifies to suppress error messages if there are objects in the ***obj_spec*** that do not exist.

Examples

Example 1

The following example lists the attributes set on three pins:

get_attribute_list {tiny1_3/d1 tiny1_3/d2 tiny1_3/d3}

```
base_name direction has_functional_source is_hard_module is_valid
leaf_name library_name master_module_name module_name name object_type
parent_instance parent_is_hard_module tie_value
```

Example 2

The following example shows the impact of using the -show_default switch, when registering an attribute, on the get_attribute_list command:

register_attribute -name a1 -value_type boolean -object_types pin
get_attribute_list {u1/pin1 u1/pin2} -user_defined

```
{}
```

register_attribute -name a2 -value_type boolean -object_types pin -show_default
get_attribute_list {u1/pin1 u1/pin2} -user_defined

```
a2
```

set_attribute_value u1/pin1 -name a1

```
{u1/pin1}
```

get_attribute_list {u1/pin1 u1/pin2} -user_defined

```
a1 a2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

get_attribute_value_list

register_attribute

report_attributes

reset_attribute_value

set_attribute_options

set_attribute_value

unregister_attribute

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_attribute_option

Context: unspecified, all contexts

Mode: all modes

Retrieves the current setting of an attribute's option.

Usage

get_attribute_option **-name** ***attribute_name*** **-object_type** ***type_name***
{-export_during_write
| -preserve_boundary_in_flat_model
| -applies_to_child_instances
| -display_in_gui
| -gui_marking_index}

Description

Retrieves the current setting of an attribute's option.

The get_attribute_option command returns a string with the option settings of the specified attribute. These options can be modified after the attribute has been registered using the set_attribute_options command.

Arguments

- **-name** ***attribute_name***

  A required switch and value pair that specifies the name of the attribute whose option values are to be returned.

- **-object_type** ***obj_type***

  A required switch and value pair that specifies the object type whose attribute option settings are to be returned.

- -export_during_write

  An optional switch that retrieves the current value of the export_during_write option for the specified attribute name on the specified object type. For more information on the -export_during_write switch, see the set_attribute_options command.

- -preserve_boundary_in_flat_model

  An optional switch that retrieves the current value of the -preserve_boundary_in_flat_model option for the specified attribute name on the specified object type. For more information on the -preserve_boundary_in_flat_model switch, see the set_attribute_options command.

- -applies_to_child_instances

  An optional switch that retrieves the current value of -applies_to_child_instances for the specified attribute name on the specified object type. For more information on the -applies_to_child_instances switch, see the set_attribute_options command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -display_in_gui

  An optional switch that retrieves the current value of the -display_in_gui option for the specified attribute name on the specified object type. For more information on the -display_in_gui switch, see the set_attribute_options command.

- -gui_marking_index

  An optional switch that retrieves the current value of the -gui_marking_index for the specified attribute name on the specified object type. For more information on the -gui_marking_index switch, see the set_attribute_options command.

Examples

The following example retrieves the -applies_to_child_instances setting for the attribute named my_power_domain that is registered for the instance object type:

```
get_attribute_option -name my_power_domain -object_type instance \
        -applies_to_child_instances
```

```
on
```

Related Topics

get_attribute_value_list

register_attribute

report_attributes

reset_attribute_value

set_attribute_options

set_attribute_value

unregister_attribute

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_attribute_value_list

Context: unspecified, all contexts

Mode: all modes

Retrieves the value of an attribute on the specified design objects.

Usage

get_attribute_value_list ***object_spec*** **-name** ***attribute_name*** [-is_specified] [-silent]

Description

Retrieves the value of an attribute on the specified design objects.

Like all other Tessent commands ending with an _list suffix, this command returns a list with all the values set for the attribute on each of the objects in ***object_spec***. This is also the case when you pass one object. You can use lindex or a foreach loop to return the single values.

If an object in ***object_spec*** does not exist, or if the attribute specified by ***attribute_name*** does not exist on the design object, an error message displays unless the -silent switch is specified.

This command is used to introspect attributes on all object types. When ***object_spec*** is a Tcl list of names, the object types are queried in a specific order until one of the data models has the object by the requested name. Use “get_attribute_value_list [get_XXX YYY]” when you want to see the attributes of object named YYY of type XXX.

For example, the design may have a pin named u1/p1 and a net by the same name. To query the attributes of the net object, use the following:

```
get_attribute_value_list [get_nets u1/p1]
```

If you use the following:

```
get_attribute_value_list u1/p1
```

then the tool returns the attributes of the pin object because the pin object model is queried before the net object model.

The object data model types are queried in the following order:

1. module
2. instance
3. port
4. pin
5. net
6. pseudo_port

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

7. gate_pin
8. icl_module
9. icl_instance
10. icl_instance_in_module
11. icl_port
12. icl_pin
13. icl_pin_in_module
14. icl_scan_interface_of_module
15. icl_scan_interface
16. icl_scan_register_in_module
17. icl_scan_register
18. icl_scan_mux_in_module
19. icl_scan_mux
20. icl_one_hot_scan_group_in_module
21. icl_one_hot_scan_group
22. icl_alias_in_module
23. icl_alias
24. config_wrapper
25. config_property
26. config_repeatable_property
27. config_data_wrapper
28. config_csv_wrapper

Arguments

- ***object_spec***

  A required value that specifies a collection of one or more objects.

  It is also legal to supply a Tcl list of one or more object names, although we do not recommend this for the following reasons:

  - Objects of different types can have the same name, which can lead to confusion.
  - Objects found may not support the requested attribute, which can generate an error.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- For types that accept escaped identifiers, using such a name may not find the object at all. For example, using an escaped port name like {\nport } with "get_single_attribute_value [list {\nport }] ...]" finds nothing. This is because escaping is not necessary for a port name that is valid without escaping. For that specific example, you should use "get_ports [list {\nport} -use_path_matching_options", and then pass the resulting collection object to the get_single_attribute_value command.

Tip

If your ***object_spec*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- **-name *attribute_name***

  A required switch and string pair that specify the name of the attribute whose value is to be returned.

- -is_specified

  An optional Boolean switch that returns 1 when the attribute was explicitly set on the object otherwise a 0 is returned.When an attribute is unspecified, its value is derived from different criteria. Instance level attributes where get_attribute_option -apply_to_child_instances returns true inherit their value from their parent instance. If the instance does not have a parent instance, then its default value is used. Attributes that do not support inheritance always have their default value when unspecified.

- -silent

  An optional switch that specifies to suppress error messages if an object in ***object_spec*** does not exist, or if the attribute specified by attribute_name does not exist for a given design object.

Examples

Example 1

The following example returns the value of parent_instance value for pins. The get_attribute_value_list command returns a Tcl list of values even if there is only one object in obj_spec. Use lindex as shown below to convert any one element list into a string that you can then concatenate with other strings to create object names. The 0 argument to the lindex command below takes the first (and only) element of the list.

get_attribute_value_list u1/u5/u8/ABC -name parent_instance

```
u1/u5
```

get_attribute_value_list u1/DEF -name parent_instance

```
{}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

lindex [get_attribute_value_list u1/DEF -name parent_instance] 0

Example 2

The following example retrieves values for the attribute named “is_hard_module” on the listed modules.

get_attribute_value_list {alu mux2 rom3} -name is_hard_module

```
false true false
```

set myList {mux2 alu}
get_attribute_value_list $myList -name is_hard_module

```
true false
```

Related Topics

get_attribute_option

register_attribute

report_attributes

reset_attribute_value

set_attribute_options

set_attribute_value

unregister_attribute

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_attributed_objects

Context: unspecified, all contexts

Mode: all modes

Retrieves the complete collection of objects matching one of the specified object type lists and having one of the attributes found in the *attribute_name_list* specified on the object.

Usage

get_attributed_objects **-attribute_names *attribute_name_list* -object_types *type_list***

Description

Use this command to return the complete collection of objects matching one of the specified object type list and having one of the attribute found in the *attribute_name_list* specified on the object.

This command works for any user-defined attributes and is much more efficient than getting the full collection of objects of a given type and filtering full collection based on attributes. For example, a design may have 5 millions pins but only 200 with the given attribute specified on it. Instead of filtering a 5 million object collection to find the 200 elements you care about, the commands constructs the collection directly from the attribute table.

There are two types of attributes: access attributes and storage attributes. Access attributes are attributes that reflect the value of a property from an internal data model. For example, the direction attribute of a pin is an access attribute. It reflects the direction of the pin as stored in the design database. A storage attribute on the other hand is not attached to an internal data structure. A dedicated hash table is created to store information about an object. All attributes you register with the register_attribute command are storage attribute. Most attributes not registered by you are access attribute.

The get_attributed_objects command only works with storage attributes. If you specify the name of an access or other unsupported attribute with the -attribute_names switch, the tool issues an error message. Use the "report_attributes -name attribute_name" command if you are not sure about an attribute. The command returns "Class : pre-defined vs user-defined". When it returns "user-defined", you can use this attribute with the get_attributed_objects command.

Arguments

- **-attribute_names *attribute_name_list***

  A switch-value pair used to specify a list of attribute names. The attribute names must be user-defined registered attributes for all the specified object types.

- **-object_types *type_list***

  A switch-value pair used to specify a list of object types for which you want its attributes objects.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example looks for all pin or port objects with the xyz_function attribute specified:

```
set attributed_objects [get_attributed_objects -attribute_names xyz_function -object_types {port pin}]
```

This example is part of "Example 4" on page 1623 in the register_drc command section. Refer to this example for the context of its usage.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_auxiliary_pins

Context: dft, patterns

Mode: setup, analysis, insertion

Returns the input or output data or enable auxiliary pins associated to ports

Usage

get_auxiliary_pins ***port_spec*** [-direction {input |output}] [-enable] [-exists] [-silent]

Description

An introspection command used to get the auxiliary pins associated to a port. The auxiliary pin information is automatically available when the auxiliary logic is created using the AuxiliaryInputOutputPorts wrapper of the BoundaryScan or EmbeddedBoundaryScan wrapper of the DftSpecification. As can be seen in Figure 13-46 on page 4138 of the AuxiliaryInputOutputPorts wrapper section, the multiplexer used to inject the auxiliary output data on an output or inout pad is done on the test side of the Output JTAG multiplexer to avoid cascading two multiplexers along the functional path. This is why when you implement boundary scan for your design, you must also implement the auxiliary data logic at the same time. If you are not using Boundary Scan in your design, you can still implement the auxiliary data logic using a DftSpecification wrapper that looks similar to the following:

```
read_config_data -from_string {
 DftSpecification(corea,rtl2) {
  BoundaryScan {
   BoundaryScanCellOptions {
    * : no_bscan_cell;
   }
   AuxiliaryInputOutputPorts {
    auxiliary_input_ports  : gpio[11:4];
    auxiliary_output_ports : gpio[7:0];
   }
  }
 }
}
process_dft_specification
```

If you are implementing the auxiliary logic manually or with a third-party tool, you can describe the mapping between the top-level ports and the auxiliary data pins using a Tcl dictionary called "::auxiliary_data_dict" that uses the following format:

```
<port_name> {
  auxiliary_output_pin          <pin_name>
  auxiliary_output_enable_pin   <pin_name>
  auxiliary_input_pin           <pin_name>
  auxiliary_input_enable_pin    <pin_name>
}
... #repeat for all ports
```

You can use the Tcl "dict" command to populate this dictionary once and access the information using the get_auxiliary_pins command afterward. The following is an example of how to use

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

the Tcl "dict" command to set the auxiliary_data_input entry for the port "gpio[1]" in the dictionary:

```
dict set ::auxiliary_data_dict gpio[1] auxiliary_input_pin u1/aux_in1
```

Arguments

- ***port_spec***

  A required string-value pair specifying a Tcl list of port names or a collection containing port objects.

- -direction {input |output}

  An optional switch-value pair specifying if the input or output data logic of the port is requested. Only inout ports can have both input and output auxiliary logic. When unspecified, the switch defaults to input.

- -enable

  An optional Boolean switch used to request the enable pin, instead of the data pin, for the specified auxiliary logic direction.

- -exists

  An optional Boolean switch used to request the existence of the auxiliary logic pin instead of the pin object itself. A 1 is returned when the requested input/output data/enable auxiliary pin exists, otherwise a 0 is returned

- -silent

  An optional Boolean switch that suppresses the error message that is normally generated when the command is called without the -exists switch and the requested input/output data/enable auxiliary pin does not exist. Instead, an empty collection is silently returned.

Examples

The following example uses the get_auxiliary_pins command to get the auxiliary input and output data pins of specific ports and use their names to populate the port_pin_name properties found within the EdtChannelsIn/EdtChannelsOut wrappers of the EDT DftSpecification wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set spec [create_dft_spec -sri_sib_list {occ}]
read_config_data -in_wrapper $spec -from_string {
  Edt {
    ijtag_host_interface : Sib(edt);
    Controller(c1) {
      longest_chain_range                  : 10, 50;
      scan_chain_count                     : 40;
      input_channel_count                  : 2;
      output_channel_count                 : 1;
      connect_bscan_segments_to_lsb_chains : on;
      Connections {
        EdtChannelsIn(1) {
        }
        EdtChannelsIn(2) {
        }
        EdtChannelsOut(1) {
          Pipelinestage {
          }
        }
      }
    }
  }
}
set_config_value port_pin_name \
  -in $spec/Edt/Controller(c1)/Connections/EdtChannelsIn(0) \
  [get_single_name [get_auxiliary_pins gpio[0] -direction input]]
set_config_value port_pin_name \
  -in $spec/Edt/Controller(c1)/Connections/EdtChannelsIn(1) \
  [get_single_name [get_auxiliary_pins gpio[1] -direction input]]
set_config_value port_pin_name \
  -in $spec/Edt/Controller(c1)/Connections/EdtChannelsOut(0) \
  [get_single_name [get_auxiliary_pins gpio[2] -direction output]]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_boundary_scan_port_option

Context: dft, patterns

Mode: all modes

Returns the cell_options defined on the specified port, or the specified pin_order_file name, or the pad_io_ports collection.

Usage

get_boundary_scan_port_option {***port_spec*** **-cell_options** | **-packed_pin_name** | **-pin_order_file** | **-pad_io_ports**}

Description

Returns the cell_options defined on the specified port, or the specified pin_order_file name, or the pad_io_ports collection.

Arguments

- ***port_spec***

  A required string that specifies the name of a port or a collection containing a single port.

- **-cell_options**

  A Boolean that queries the cell options that were previously specified on the given port using the "set_boundary_scan_port_options -cell_options" command.

- **-packed_pin_name**

  A Boolean option that queries the device package pin names previously specified with the "set_boundary_scan_port_options -packed_pin_name <port_name_list>" command.

- **-pin_order_file**

  A Boolean option that queries the previously specified pin_order_file name using the "set_boundary_scan_port_options -pin_order_file" command.

- **-pad_io_ports**

  A Boolean option that queries the previously specified collection of pad_io ports using the "set_boundary_scan_port_options -pad_io_ports" command.

Examples

The following example requests the cell_options that were previously specified on the port dac_out.

set_boundary_scan_port_option dac_out -cell_options analog

get_boundary_scan_port_option dac_out -cell_options

```
analog
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

report_boundary_scan_port_options

set_boundary_scan_port_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_bsdl_extraction_option

Context: dft (with no subcontext)

Mode: setup, analysis

Returns a value for a BSDL attribute previously set with the set_bsdl_extraction_options command.

Usage

get_bsdl_extraction_option
{**-pulse_min_duration** | **-train_minimum_count** | **-train_max_duration** | **-manufacturer_id_code** | **-part_number_code** | **-version_code** | **-bonding_configuration_enables**}

Description

Returns a specified value for a BSDL attribute previously set by the set_bsdl_extraction_options command for the BSDL extraction.

Tip

You can report all the values for the various BSDL attributes using the report_bsdl_extraction_options command.

Arguments

- -pulse_min_duration

  A required switch that returns the value for minimum pulse duration. This is the minimum time to use between the rising and the next falling edge during the AC tests.

- -train_minimum_count

  A required switch that returns the value for the minimum number of pulses the train test must have.

- -train_max_duration

  A required switch that returns the value for the maximum time the train test can take.

- -manufacturer_id_code

  A required switch that returns the binary value of the manufacturer ID code section of the 32-bit device ID.

- -part_number_code

  A required switch that returns the binary value of the part number code section of the 32-bit device ID.

- -version_code

  A required switch that returns the binary value of the version code section of the 32-bit device ID.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -bonding_configuration_enables

  A required switch that returns the value of the bonding configuration enable pins/ports.

Related Topics

report_bsdl_extraction_options

delete_bsdl_extraction_option

set_bsdl_extraction_options

Prerequisites for BSDL Extraction

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_bundle_objects

Context: all contexts

Mode: all modes

Returns a collection of bundle/leaf objects in the current design specified by the name_patterns list in conjunction with the specified options. Those bundle (leaf in case of descendants) objects are ancestors or descendants of the objects that match the specified *name_patterns* depending upon the specified options.

Usage

get_bundle_objects [*name_patterns*]
-of_objects *of_objects*
[-parent | -children | -root | -leaf]
[-filter *attribute_equation*]
[-regexp]
[-nocase]
[-silent]

Description

When the *name_patterns* list is omitted, the tool returns all parent and child complex signal bundle objects subject to filtering imposed by the options you specify.

For complete usage information, see "Bundle Object Introspection" in the *Tessent Shell User's Manual*.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of bundle objects. The string is a Tcl list of patterns separated by spaces and enclosed in braces ({}). If no *name_patterns* are specified, the tool searches all bundle objects. The *name_patterns* are not permitted to include slashes, which are hierarchy separator symbols. See "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a complete description of the pattern syntax when the -regexp option is used and not used.

- **-of_objects *of_objects***

  A switch and Tcl list of connective objects or a collection of one or more connective objects. A connective objects are one or more port, port_bundle, pin, pin_bundle, net, or net_bundle objects.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -parent | -children | -root | -leaf

  An optional switch that specifies which bundle objects to return. Choose one from the following:

  -parent — Returns the parent bundle of the object. With this option, you can introspect the ancestors of the selected objects.

  -children — Returns the child bundles of the objects, which permits you to introspect the descendants of the selected objects.

  -root — Returns the top-level bundle.

  -leaf — Returns the leaf objects of the bundle (port_bundle -> port, pin_bundle -> pin, net_bundle -> net).

- -filter *attribute_equation*

  An optional switch value pair that filters the introspection results using the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the port, pin, net, port_bundle, pin_bundle, or net_bundle object types. See section "Attribute Filtering Equation Syntax" for more details on filtering attribute equation format. The =~ and !~ matching operators used simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option or not.

- -regexp

  An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a regular expression instead of as a simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for nets matching the *name_patterns*.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

  Note

  You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following examples demonstrate the usage of the get_bundle_objects command:

Example 1

set b1 [get_bundle_objects $b -children]

```
{Bus[0]} {Bus[1]}
```

get_bundle_objects -of_objects {Bus[0].Addr} -leaf

```
Bus[0].Addr.state_encoded[2]} {Bus[0].Addr.state_encoded[1]}
{Bus[0].Addr.state_encoded[0]} {Bus[0].Addr.dt_lcp_pulse.eventbus[2]}
{Bus[0].Addr.dt_lcp_pulse.eventbus[1]}
{Bus[0].Addr.dt_lcp_pulse.eventbus[0]} {Bus[0].Addr.eventbus_encoded[3]}
{Bus[0].Addr.eventbus_encoded[2]} {Bus[0].Addr.eventbus_encoded[1]}
{Bus[0].Addr.eventbus_encoded[0]}
```

Example 2

```
typedef struct packed {
    logic f;
} struct_t;

typedef struct_t [1:0][1:1] array_t;
input array_t data_in [2:0]
```

get_bundle_objects -of_objects [get_ports data_in[2] -bundle]

```
{data_in[2][1] data_in[2][0]}
```

get_bundle_objects -of_objects [get_ports data_in[2] -bundle] -leaf

```
{data_in[2][1][1].f data_in[2][0][1].f}
```

get_bundle_objects -of_objects [get_ports data_in[2][*] -bundle] -leaf

```
{data_in[2][1][1].f data_in[2][0][1].f}
```

get_bundle_objects -of_objects [get_ports data_in[2][*] -bundle] -children

```
{data_in[2][1][1] data_in[2][0][1]}
```

get_bundle_objects -of_objects [get_ports data_in[2][*] -bundle] -parent

```
{data_in[2] data_in[2]}
```

get_bundle_objects -of_objects [get_ports data_in[2][*] -bundle] -root

```
{data_in data_in}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_capture_timing_option

Context: all contexts

Mode: all modes

Returns specific capture timing options specified with the set_capture_timing_options command.

Usage

```
get_capture_timing_option
    [-icl_module icl_module_name
    | -module module_name
    | -current_design
    | -current_physical_block]
    -mode_type {internal | external}
    {-capture_clock_period [-time_unit {s | ms | us | ns | ps | fs | as_is}]
     | -capture_clock_edge1_percentage
     | -capture_clock_edge2_percentage
     | -capture_clock_high_pulse_width
     | -min_capture_window_in_ssn_bus_cycles
     | -generated_timeplate_name
     | -pi_setup_extra_cycles
     | -pi_hold_extra_cycles
     | -pi_force_percentage
     | -po_measure_percentage}
    [-is_specified]
```

Description

The get_capture_timing_option command returns information about a specified capture timing setting previously configured with the set_capture_timing_options command.

If you specified a module with the set_current_physical_block command and the capture timing options for the module, the tool returns those values. If you did not specify a module, the tool traverses the design hierarchy until it finds a parent that was specified as a physical block. Otherwise, the tool uses the module inherited from the current design, and returns the default values if no options were set.

If the module has multiple instantiations and all its instances inherit from the same parent, then the tool returns the values of that parent. If the instances of module inherit from different parents (see Figure 5-1), then the tool issues the following error:

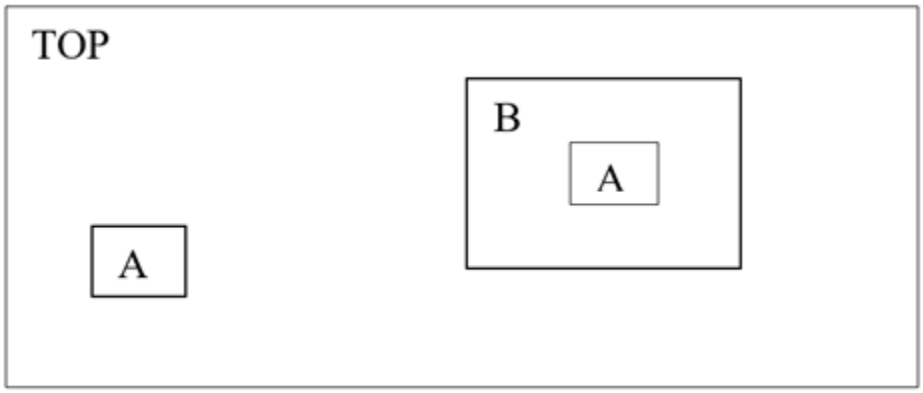
```
//  Error: The specified Module 'foo'  was not set as a physical block by
//  'set_current_physical_block ' command. It has multiple instances that
//  do not belong to the same physical block.
```

For example, Figure 5-1 shows that module A was never specified as a physical block, and module B was specified as physical block. Because module A has two instances, one of them

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

inherits from block B and the other inherits from current design TOP. This scenario will result in an error.

Figure 5-1. Design Inheritance and Module Matching



Arguments

- -icl_module *icl_module_name*

  A switch and string pair that specifies the name of a valid ICL module from which to return the timing option.

- -module *module_name*

  A switch and string pair that specifies the name of a module from which to return the timing option.

- -current_design

  A switch that specifies retrieving the timing option from the current design.

- -current_physical_block

  A switch that specifies retrieving the timing option from the current physical block. This is the default.

- -mode_type {internal | external}

  A switch and literal pair that specifies the capture mode.

- -capture_clock_period [-time_unit {s | ms | us | ns | ps | fs | as_is}]

  A switch and optional switch and literal that returns the capture clock period set with the "set_capture_timing_options -capture_clock_period" command.

  The -time_unit switch-value pair is an optional Boolean that can be used when the command is returning a Value time property value; it is ignored otherwise. When specified or defaulted to as_is, the time value is returned unmodified with the unit string specified if it existed in the value of the configuration element. When specified to a value other than as_is, the time value is converted into a real number formatted to the specified units. The unit string is also dropped. The default unit is nanoseconds (ns).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -capture_clock_edge1_percentage

  A switch that returns the edge 1 capture clock percentage value set with the "set_capture_timing_options -capture_clock_edge1_percentage" command.

- -capture_clock_edge2_percentage

  A switch that returns the edge 2 capture clock percentage value set with the "set_capture_timing_options -capture_clock_edge2_percentage" command.

- -capture_clock_high_pulse_width

  A switch that returns the capture clock high pulse width value set with the "set_capture_timing_options -capture_clock_high_pulse_width" command.

- -min_capture_window_in_ssn_bus_cycles

  A switch that returns the minimum width of the capture window in bus clock cycles set with the "set_capture_timing_options -min_capture_window_in_ssn_bus_cycles" command.

- -generated_timeplate_name

  A switch that returns the timeplate name as defined by the "set_capture_timing_options -generated_timeplate_name" command.

- -pi_setup_extra_cycles

  A switch that returns the PI setup extra cycles value set with the "set_capture_timing_options -pi_setup_extra_cycles" command.

- -pi_hold_extra_cycles

  A switch that returns the PI hold extra cycles value set with the "set_capture_timing_options -pi_hold_extra_cycles" command.

- -pi_force_percentage

  A switch that returns the PI for percentage value set with the "set_capture_timing_options -pi_force_percentage" command.

- -po_measure_percentage

  A switch that returns the PO measure percentage value set with the "set_capture_timing_options -po_measure_percentage" command.

- -is_specified

  A switch that returns true if the specified capture timing option switch was set by a previous set_capture_timing_option command.

Examples

The following example returns the internal capture timing options for the capture clock period for the module moduletest.

```
get_capture_timing_option -mode_type internal -capture_clock_period \
        -icl_module moduletest
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

report_capture_timing_options

reset_capture_timing_options

set_capture_timing_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_burnin_option

Context: patterns -scan, patterns -scan_retargeting

Mode: setup, analysis

Returns the current setting of an option specified with the set_burnin_options command.

Usage

get_burnin_option **-sequence** | **-repetitions** | **-payload_loop_count** | **-ssh_icl_instances**

Description

The get_burnin_option command returns information about the settings for creating burn-in patterns.

Arguments

The following arguments are mutually exclusive; choosing one is required:

- **-sequence**

  A required switch that returns the binary sequence to use for the burn-in payload.

- **-repetitions**

  A required switch that returns the number of times to repeat the pattern in the burn-in payload.

- **-payload_loop_count**

  A required switch that returns the number of times to loop through the payload. If no count has been specified, this returns “off”.

- **-ssh_icl_instances**

  A required switch that returns a list of the SSH ICL instances targeted with the burn-in pattern.

Examples

get_burnin_options -sequence

```
1100
```

Related Topics

set_burnin_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_cdp_test_data

Context: patterns -silicon_insight

Mode: all

Returns requested information for a specific CDP test.

Usage

get_cdp_test_data ***test_name***
{ **-user_pattern_file** | **-diagnostic_pattern_files** |
-dynamic_parameter_value ***dynamic_parameter_name*** |
{ **-user_pattern_variable_list** |
-user_pattern_variable_location_data ***user_pattern_variable_location_data*** }
[-variable_type *variable_type*] }

Arguments

- ***test_name***

  A required string that specifies the CDP test to return information for.

- **-user_pattern_file**

  A required switch that returns the path, relative to the CDP, for the main user (that is, go-nogo) pattern file for the specified CDP test.

- **-diagnostic_pattern_files**

  A required switch that returns the paths, relative to the CDP, for all diagnostic pattern files for the specified CDP test.

- **-dynamic_parameter_value** ***dynamic_parameter_name***

  A required switch and string pair that returns the value of the dynamic parameter with the specified name.

- **-user_pattern_variable_list** [-variable_type *variable_type*]

  A required switch and optional switch and literal pair that returns a list of register names in the main user pattern for the specified CDP test. By default, this returns all such registers. Restrict the list with the -variable_type switch; <*variable_type*> can be one of read, write, or any.

- **-user_pattern_variable_location_data** ***user_pattern_variable_location_data***
  [-variable_type *variable_type*]

  A required switch and string pair with an optional switch and literal pair that returns location data for specific variables or registers. By default, this returns all matching variables or registers. Restrict the list with the -variable_type switch; <*variable_type*> can be one of read, write, or any.

Examples

```
get_cdp_test_data MemoryBist_P1 -user_pattern_variable_location_data \
        run_time_prog::blockA_inst.mem1_interface_inst.GO_ID_REG
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{VectorID 26 Type read Value b00000000 VarBits 0:7 RelativeCycles 20:27 Pin tdo}
```

The values in this example correspond to the following:

- **VectorID —** Location in pattern, given as <*vector_address*>:<*cycle_count*> for STIL and <*command ID*> for SVF.
- **Type —** "read" or "write".
- **Value —** 'b' followed by the expected value in binary.
- **VarBits —** Bit range with LSB/MSB orientation. Use this with the RelativeCycles data to determine the overall order in which the pin returns the bits.
- **RelativeCycles —** Time slot offset and orientation of data on the pin.
- **Pin —** Name of the pin on which the register value appears.

For more information about these values, refer to the section "Retargeted Symbolic Variables" in the *Tessent IJTAG User's Manual*.

Related Topics

set_cdp_test_data

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_cdp_test_list

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Returns a Tcl list of tests currently available in the CDP.

Usage

get_cdp_test_list

Arguments

None

Examples

The following example shows how you can use this command to return a Tcl list of the current tests within the CDP.

```
foreach test [get_cdp_test_list] {
        execute_cdp_test $test
}
```

```
taptest membist1 membist2 logicbist1 logicbist2
```

Related Topics

execute_cdp_test

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_clock_option

Context: all contexts

Mode: setup, analysis

Prerequisites: The current design must be set.

Returns the setting of an option specified with the add_clocks command.

Usage

get_clock_option ***object_spec***
{**-type** | **-off_state** | **-pulse_always** | **-period** [-timeplate *timeplate*]
[-time_unit {ps | ns | us | ms | s}]
| -pulse_in_capture
| -label
| -reference
| -reference_inv
| -freq_multiplier
| -freq_divider}
| -dft_inject_node
| -no_dft_connections
[-silent] [-map_pseudo_ports]

Description

Returns the setting of an option specified with the add_clocks command.

The get_clock_option command returns the setting of a single option type for a single defined clock. You use the get_clocks command to gather the list of defined clocks.

Note

For more information about any of the following switches, refer to corresponding switch in the add_clocks command description.

Arguments

- ***object_spec***

  A required string that specifies an existing clock using the name of a port, pin, pseudo_port, label, or a collection of one port, or one pin, one pseudo_port, or one label object.

- -type

  An optional switch that returns one of the following literals: sync_source, async_source, branch, or generated_source. For information about the clock types, refer to the add_clocks command description.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -off_state

  An optional switch that returns 0 or 1 for synchronous clocks, and null for asynchronous clocks.

- -pulse_always

  An optional switch that returns a value of 1 if the clock is a pulse-always clock or 0 if it is not a pulse-always clock. A pulse-always clock is pulsed during every cycle of the pattern set.

- -period [-timeplate *timeplate*] [-time_unit {ps | ns | us | ms | s}]

  An optional switch that returns the clock period if specified. These two options can be used with the -period switch:

  - -timeplate *timeplate* — An optional switch and string pair used to identify the timeplate for determining the period from the timeplate specified in the procedure file. This option applies to, and must be specified for, synchronous clocks. When a synchronous clock is not pulsed in the specified timeplate, the time returned is three times the period of the timeplate. The "set time scale" statement in the procedure file defines the unit the tool uses for the timeplate period if you do not specify the -time_unit option.

  - -time_unit { ps | ns | us | ms | s } — An optional switch that specifies the time unit for the period. The time unit is not part of the returned string when using the -time_unit switch. There is no default unit for the period. The tool returns the period of synchronous clocks in the unit of the specified timeplate and the period of asynchronous clocks in the unit specified for the period in the add_clocks command.

- -pulse_in_capture

  An optional switch that returns a 0 or 1 to indicate whether the clock is a pulse-in-capture clock and the source is synchronous. This switch returns null when the source is asynchronous.

- -label

  An optional switch that returns the string value of the label name. If you did not previously specify a label, the tool supplies a unique label.

- -reference

  An optional switch that returns a collection with the object that the -reference option referred to when the add_clocks or set_clock_options commands was run. If the -reference switch was not specified, it returns an empty collection.

- -reference_inv

  An optional switch that returns a collection with the object that the -reference_inv option referred to when the add_clocks or set_clock_options commands was run. If the -reference_inv switch was not specified, it returns an empty collection.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -freq_multiplier

  An optional switch that returns the integer value of freq_multiplier, if specified; otherwise, it returns a 1.

- -freq_divider

  An optional switch that returns the integer value of freq_divider, if specified; otherwise, it returns a 1.

- -dft_inject_node

  An optional switch that returns the node object you specified with the -dft_inject_node option to the add_clocks or set_clock_options commands. If you did not specify an injection node with the -dft_inject_node switch for those commands, the tool returns a node object that is the clock node. See also the -map_pseudo_ports switch.

- -no_dft_connections

  An optional switch that returns a 0 or 1 to indicate if the clock has DFT connections as specified with the -no_dft_connections option to the add_clocks or set_clock_options commands.

- -silent

  An optional switch that suppresses the error message normally generated if the specified *object_spec* is not a clock, and returns a null. The -silent option has no effect on any other error reporting. You use this switch to determine if the *object_spec* is a clock. If the command "get_clock_option *object_spec* -type -silent" returns a null, you know *object_spec* is not a clock. When it returns a string, you know that *object_spec* is a clock and also its type.

- -map_pseudo_ports

  An optional switch that specifies mapping pseudo ports to their associated pin or net objects. When using the -reference switch or the -reference_inv switch, and the returned reference or reference_inv object is a pseudo_port, the command returns the port or pin object on which the pseudo_port was created, instead of the pseudo_port object.

  When -map_pseudo_ports is not specified, the tool returns the pseudo_port object. When the clock is not a pseudo_port, the switch has no effect.

  If you use this switch in conjunction with the -dft_inject_node switch, the tool returns the related hierarchical pin for the pseudo_port.

Examples

Example 1

The following example adds two clocks to the clock list, reports on the clocks, and then shows several ways of using the get_clock_option command:

```
SETUP> add_clocks 0 I3 -pulse_always -label clock3
SETUP> add_clocks 1 I4 -period 50 ns -freq_multiplier 2 -label clock4
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SETUP> report_clocks

```
User-defined Clocks (2):
========================
  Sync and Async Source Clocks
  ============================
   ----  ------  ---------  ------------  --------  ------------  --------------------
   Name   Label  Off State   Constraints  Internal     Period     Freq. Multiplication
   ----  ------  ---------  ------------  --------  ------------  --------------------
   'I3'   clock3      0      Pulse always     No
   'I4'   clock4      -      Asynchronous     No     50.00ns                2
```

SETUP> get_clock_option I3 -label clock

```
clock3
```

SETUP> get_clock_option I4 -freq_multiplier

```
2
```

SETUP> get_clock_option I4 -period -time_unit ps

```
50000.0
```

SETUP> get_clock_option I3 -pulse_in_capture

```
0
```

SETUP> get_clock_option I4 -freq_divider

```
1
```

SETUP> get_clock_option I3 -reference

```
{}
```

Example 2

This example demonstrates the use of the -map_pseudo_port switch. It also shows the effect when you do not use the switch.

add_clocks {/pll1_I1/tdr1_IPLL1/U15/Y} -pseudo_port my_cut_pi -reference tck

```
//  Note: Primary input 'my_cut_pi' is added at pin '/pll1_I1/tdr1_IPLL1/U15/Y'.
```

add_clocks {/pll1_I1/tdr1_IPLL1/U16/Y} -reference my_cut_pi

get_clock_option {/pll1_I1/tdr1_IPLL1/U16/Y} -reference -map_pseudo_ports

```
{pll1_I1/tdr1_IPLL1/U15/Y}
```

get_clock_option {/pll1_I1/tdr1_IPLL1/U16/Y} -reference

```
{my_cut_pi}
```

Related Topics

add_clocks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_clocks

report_clocks

set_clock_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_clocks

Context: all contexts

Mode: setup, analysis

Prerequisites: The current design must be set.

Returns a collection of user clocks based on filters you specify.

Usage

get_clocks [*name_patterns* [-match {all | pin_or_port_name | label_name}] [-regexp] [-nocase]] [-type {sync_source | async_source | generated | branch}] [-silent] [-map_pseudo_ports]

Description

Returns a collection of user clocks based on filters you specify.

The get_clocks command returns a collection of all clock objects in your design created with add_clocks or a subset based on filters you specify.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to filter the list of returned list of clock objects.

- -match {all | pin_or_port_name | label_name}

  An optional switch and literal pair that specifies to match the *name_patterns* with pin or port names, label names, or both (All). By default, the command matches all.

- -regexp

  An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a real regular expression instead of as a simple wildcard pattern.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for clocks matching the *name_patterns*.

- -type {sync_source | async_source | generated | branch}

  An optional switch and literal pair that specifies the type of clock to return. By default, the command returns all types of clocks. For information about the clock types, refer to the "Usage" section of the add_clocks command description.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

- -map_pseudo_ports

  An optional switch that specifies mapping pseudo ports to their associated pins or net objects. When the clock is a pseudo_port and this switch is used, the command returns the port or pin object on which the pseudo_port was created, instead of the pseudo_port object.

  When -map_pseudo_ports is not specified, the tool returns the pseudo_port object.

  This switch has no effect when the clock is not a pseudo_port.

Examples

Example 1

The following example creates two sync_source clocks and then shows different ways to use the get_clocks command:

SETUP> add_clocks I1 -pulse_always -label clock1

SETUP> add_clocks I2 -pulse_in_capture -label clock2

SETUP> get_clocks

```
{I1 I2}
```

SETUP> get_clocks I* -match label_name

```
//  Error: 'I*' is not a clock.
{}
```

SETUP> get_clocks * -match label_name

```
{I1 I2}
```

SETUP> get_clocks clock* -match label_name

```
{I1 I2}
```

SETUP> get_clocks I* -type sync_source

```
{I2 I1}
```

SETUP> get_clocks I* -type async_source

```
{}
```

SETUP> get_clocks c*2 -match label_name

```
{I2}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

This example creates a pseudo_port and then demonstrates using the -map_pseudo_port switch to return the pin information for the pseudo_port. It also shows the result when the switch is not used.

add_clocks {/pll1_I1/tdr1_IPLL1/U15/Y} -pseudo_port my_cut_pi -reference tck

```
//  Note: Primary input 'my_cut_pi' is added at pin '/pll1_I1/tdr1_IPLL1/
U15/Y'
```

get_clocks my_cut_pi -map_pseudo_port

```
{pll1_I1/tdr1_IPLL1/U15/Y}
```

get_clocks my_cut_pi

```
{my_cut_pi}
```

Related Topics

add_clocks

delete_clocks

get_clock_option

report_clocks

set_clock_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_common_parent_instance

Context: all contexts

Mode: all modes

Returns the common ancestor of all instances, pins, or nets found in ***object_spec***.

Usage

get_common_parent_instance ***object_spec***

Description

Returns the common ancestor of all instances, pins, or nets found in ***object_spec***.

You can use this command during dft insertion when you want to place a driving gate at the common ancestor of all its fanouts.

Arguments

- ***object_spec***

  A required value that specifies a Tcl list of one or more object names or a collection of one or more objects to find their common ancestor.

Examples

The following example finds the common ancestor of all pins having the attribute myAtt set to true. The name of the returned instance object is then used to instantiate a driver cell. Quotation marks enclose the concatenation of ${parent_name} and /myGate to make sure that the command preserves any white spaces that may exist in ${parent_name} (when it contains escaped identifiers).

```
set parent [get_common_parent_instance [get_pins -filter myAtt]]
set parent_name [get_single_name $parent]
set driver [create_instance “${parent_name}/myGate” -of_module MyMod]
```

Related Topics

create_instance

delete_instances

rename_instance

replace_instances

uniquify_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_config_elements

Context: unspecified, dft, patterns

Mode: all modes

Returns a collection of configuration elements or a count of configuration elements when the -count option is used.

Usage

get_config_elements [*name_patterns*] [-hierarchical] [-in_wrappers *wrapper_object_spec*]
[-partition *partition*] [-count] [-type *type*] [-filter *filter*]
[-regexp] [-nocase] [-silent]

Description

Returns a collection of configuration elements or a count of configuration elements when the -count option is used.

When no *name_patterns* is specified, then all elements are returned unless the -in_wrappers option is specified, in which case only the elements below the specified wrappers are returned. When using one or many *name_patterns*, only the elements whose name matches at least one of the *name_patterns* are returned.

Arguments

- *name_patterns*

  An optional string that specifies one name pattern or a Tcl list of name patterns to search. The name patterns are automatically decomposed into a series of anchored leaf name patterns using the / as a separator. The name identifiers are always matched ignoring case. The ids found inside the parentheses are matched taking into account case unless the -nocase option is used. The comma has special meaning inside the parentheses and is used to split the patterns and match each id independently. A leaf pattern that does not include a set of parentheses matches all elements matching the leaf name irrespective of whether it has ids or not. See the examples below for a complete illustration of the matching method.

  When the -in_wrappers option is specified, the *name_patterns* are described relative to the specified wrappers, When the -hierarchical option is specified, the *name_patterns* are matched recursively against all elements below the specified wrappers.

- -hierarchical

  An optional string that specifies that the *name_patterns* are matched recursively against all elements below the specified wrappers.

  You should use this option with care and avoid using it inside scripts that are intended to be run continually throughout a development cycle. For example, you would not normally use the “get_pins clka -hierarchical” command to find “u1/u2/clka” in a script that is meant to run on a design as it evolves. For the same reason, you should not use the -hierarchical

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

option in a configuration editing script. Although a *leaf_name* may be unique now, there is no guarantee that it continues to remain unique in the future.

- -in_wrappers *wrapper_object_spec*

  An option and value pair that specifies that the configuration elements are to be searched inside specific wrappers. When the -in_wrappers option is specified, the *name_patterns* are searched relative to the specified wrappers. The *wrapper_object_spec* can be the complete names of wrappers relative to the root of the partition or a collection containing one or many wrapper elements as returned by get_config_elements or "get_config_value -object". When specifying the names of the wrapper, you can use the -partition option to specify in which partition the wrappers exist.

- -partition *partition_name*

  An optional switch and value pair that specifies the partition from which the configuration element is to be obtained. When using the -in_wrappers option, the -partition option can only be specified if the *wrapper_object_spec* is the name of a wrapper. When the wrapper_object_spec is a collection containing wrapper objects, the -partition option cannot be specified because a specific partition is already associated to the configuration objects. All configuration data you normally interact with is in the default partition, so you typically never need to specify the -partition option.

  The tool defines two partitions:

  - **default partition** — a partition that stores all user specifications; contains the definition of DftSpecification, and DefaultsSpecification/DftSpecification wrappers
  - **tcd partition** — a read-only partition in which the read_core_descriptions command stores the data contained inside the Core(*module_name*) wrapper

- -count

  An optional Boolean option that specifies to return the number of elements matched as opposed to the elements themselves.

- -type wrapper | data_wrapper | csv_wrapper | property

  An optional switch and value pair that specifies to limit the search to a specific configuration element type. When the -type option is omitted, all types are considered in the matching.

- -filter *attribute_equation*

  An optional switch and string pair that specifies to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the configuration element types being filtered. See section "Attribute Filtering Equation Syntax" for more details on filtering attribute equation format. The =~ and !~ matching operators use simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -regexp

  An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a regular expression instead of as a simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when matching the Ids of wrapper and repeatable properties. The wrapper and property names are always matched ignoring casing.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

  Note

  You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the "set_tcl_shell_options -" switch description for detailed usage.

Examples

The following example operates on the following configuration data:

```
tmp(1) {
  ABC (Def,2) {
    prop1 : 1;
    Prop2 : 2;
  }
  abc(def,1) {
  }
  abcdef {
  }

}
```

Case 1

This matches the wrappers with leaf name "abc" inside wrapper tmp(1) independent of whether it has an id. The matching is case-insensitive because only the id matching is case-sensitive.

get_config_elements abc -in tmp(1)

```
{/tmp(1)/ABC(Def,2) /tmp(1)/abc(def,1)}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Case 2

This matches the wrappers with the first id equal to “Def”. Notice that it does not match abc(def,1) as ids are matched considering casing unless the -nocase option is used.

get_config_elements *(Def,*) -in tmp(1)

```
{/tmp(1)/ABC(Def,2)}
```

get_config_elements *(Def,*) -in tmp(1) -nocase

```
{/tmp(1)/ABC(Def,2) /tmp(1)/abc(def,1)}
```

Case 3

This matches elements below the top-level wrapper starting with a*.

get_config_element */a*

```
{/tmp(1)/ABC(Def,2) /tmp(1)/abc(def,1) /tmp(1)/abcdef}}
```

Case 4

This matches elements below the top-level wrapper starting with a* and having two ids.

get_config_element */a*(*,*)

```
{/tmp(1)/ABC(Def,2) /tmp(1)/abc(def,1)}}
```

Case 5

This matches elements starting with ‘p’ anywhere inside tmp(1).

get_config_element p* -in tmp(1) -hierarchical

```
{/tmp(1)/ABC(DeF,2)/prop1 /tmp(1)/ABC(DeF,2)/Prop2}
```

Case 6

This matches elements starting with ‘p’ anywhere inside tmp(1) and returns the amount of elements that matched instead of the elements themselves.

get_config_element p* -in tmp(1) -hierarchical -count 2

Related Topics

add_config_element

get_config_value

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_config_messages

Context: all contexts

Mode: all modes

Returns a list of message strings attached to a configuration element.

Usage

get_config_messages ***config_object_spec***
[-string | {-objects [-hierarchical]}]
[-all | {[-error] [-warning] [-info]]

Description

Returns a list of message strings attached to a configuration element.

By default, all message types are returned unless you specify at least one of the -error, -warning, or -info options. If you specify the -objects option, the command returns the configuration element if it has messages of the specified type.

Arguments

- ***config_object_spec***

  A required value that specifies the name of a configuration element or a collection containing one configuration element.

- -string

  An optional Boolean option that specifies that the command is to return the messages formatted into strings exactly as they are shown with the report_config_messages command. The entries start with E#), W#) and I#) based on the type of the message and # is incremented from 0 independently for each type.

  The use of -string is mutually exclusive with the -objects option, When either is specified, -string is assumed.

- -objects

  An optional Boolean option that specifies to return a collection of configuration elements having at least one message of the specified type.

- -hierarchical

  An optional Boolean option that specifies that the config_object_spec is a wrapper and that the command should return all elements below the specified wrapper having at least one configuration message for the specified types. This option is only permitted in combination with the -objects option

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -all

  An optional Boolean option that specifies that all messages types are to be returned. The use of the -all option is mutually exclusive with the specification of the -error, -warning and -info options. When none of the four options is specified, -all is assumed.

- -error

  An optional Boolean option that specifies that the messages returned are of type error. The use of the -error option is mutually exclusive to the specification of the -all, -warning, and -info options. When none of the -error, -warning and -info options is specified, -all is assumed.

- -warning

  An optional Boolean option that specifies that the messages returned are of type warning. The use of the -error option is mutually exclusive with the specification of the -all, -warning, and -info options. When none of the -error, -warning, and -info options is specified, -all is assumed.

- -info

  An optional Boolean option that specifies that the messages returned are of type info. The use of the -error option is mutually exclusive with the specification of the -all, -warning, and -info options. When none of the -error, -warning and -info options is specified, -all is assumed.

Examples

The following example adds an error message to the property rtl_extension inside DftSpecification(mydesign,rtl). The get_config_messages is then used to get the message string list on the property. The command "get_config_messages -objects -hierarchical" is also used to get a collection of all elements below DftSpecification(mydesign,rtl) with an error message.

```
set_context dft -rtl

set spec [add_config_el DftSpecification(mydesign,rtl)]

add_config_message "this is an error string \non two lines" \
    -config_element [get_config_el rtl_extension -in $spec] \
    -error -display

//  Error: /DftSpecification(mydesign,rtl)/rtl_extension
//         this is an error string
//         on two lines

get_config_messages [get_config_el rtl_extension -in $spec]

E0) this is an error string
    on two lines

get_config_messages $spec -objects -hier {/DftSpecification(mydesign,rtl)/rtl_extension}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

add_config_message

delete_config_messages

report_config_messages

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_config_value

Context: unspecified, all contexts

Mode: all modes

Returns a value associated with a configuration element based on the specified option.

Usage

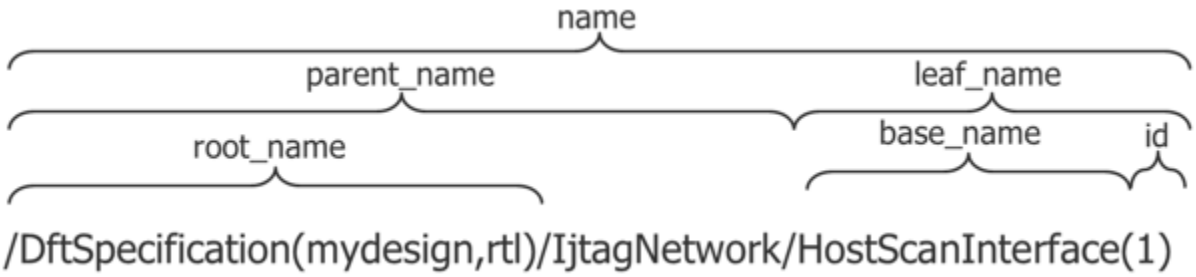
get_config_value ***config_object*** [-in_wrapper *wrapper_object_spec*]
[-time_unit {s | ms | us | ns | ps | fs | as | as_is]
[-number_format {int | bin | hex | as_is}]
[-reference_name | -reference_object | -parent_name | -parent_object | -type | -name |
-id *id* | -value_id *value_id* | -value_as_list | -is_specified | -has_inherited_default |
-id_count | -value_count | -exists | -root_name | -root_object | -leaf_name | -base_name |
-meta_name | -meta_object | -meta_id | -partition_name]
[-default_value | -comment]
[-tolower | -toupper]
[-file_name]
[-object]
[-partition *partition*]
[-silent]

Description

Returns a well-formatted Tcl list when the property permits a comma-separated list of values. See the Value ID description in the metadata section that shows that terminating the ID with an asterisk "*" indicates that the value accepts a comma-separated list of values.

Figure 5-2 below illustrates some of those values.

Figure 5-2. Various Components of a Configuration Element



Arguments

- ***config_object***

  A required value that specifies the name of a configuration element or a collection containing one configuration element. When ***config_object*** is a collection, the -in_wrapper and -partition options cannot be used because the object contains all of the information about the configuration element. In order to use the -in_wrapper option, the ***config_object*** value must be a name and the name must be relative to the specified wrapper element. You

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

can use a positional id for any level of hierarchy. You can also replace the id of a leaf name by the position. For example, the path “tmp(1)/<0>/wrp<1>” means “tmp(1)/wrp(abc)/wrp(def)” if “wrp(abc)” is the first element found inside “tmp(1)”, and “wrp(def)” is the second instance of a wrapper with a base name of “wrp” inside “tmp(1)/wrp(abc)”.

- -in_wrapper *wrapper_object_spec*

  An optional switch and value pair that is used to specify the parent wrapper in which the configuration element exists. The *wrapper_object_spec* value is the name of a wrapper or a collection containing a wrapper element. The name in ***config_object*** is relative to the specified parent wrapper.

- -time_unit s | ms | us | ns | ps | fs | as | as_is

  An optional Boolean option that can be used when the command is returning a Value time property value; it is ignored otherwise. When specified or defaulted to as_is, the time value is returned unmodified with the unit string specified if it existed in the value of the configuration element. When specified to a value other than as_is, the time value is converted into a real number formatted to the specified units. The unit string is also dropped.

- -number_format int | bin | hex | as_is

  An option and value pair that can be used when the command is returning a Value binary property or hexadecimal value. It is ignored otherwise. When specified or defaulted to as_is, the value is returned unmodified with the format if it existed in the value of the configuration element. When specified to a value other than as_is, the binary or hexadecimal value is converted into the specified format.

- -reference_name

  An optional Boolean option that specifies that the data requested is the name of the referenced metadata element. If the metadata is not a Wrapper reference property, this returns a null string.

- -reference_object

  An optional Boolean option that specifies that the data requested is a collection containing the referenced metadata element. If the metadata is not a Wrapper reference property, this returns an empty collection.

- -parent_name

  An optional Boolean option that specifies that the data requested is the name of the wrapper in which the configuration element exists. If the element is a root wrapper, the string / is returned.

- -parent_object

  An optional Boolean option that specifies that the data requested is a collection containing the wrapper in which the configuration element exists. If the element is a root wrapper inside a given partition, a collection containing the partition in which the root wrapper exists is returned.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -type

  An optional Boolean option that specifies that the data requested is the type of the configuration element. The type is either wrapper, data_wrapper, csv_wrapper, property, repeatable_property, data_element, or csv_element. Table 5-2 explains each type in more details.

Table 5-2. Types of Configuration Elements

Type	Description
wrapper	Type returned for a Wrapper used to groups other wrappers and properties.
data_wrapper	Type returned for a DataWrapper used to group data_row elements.
csv_wrapper	Type returned for a CsvWrapper used to group csv_row elements.
property	Type returned for a Property.
repeatable_property	Type returned for a RepeatableProperty property.
data_row	Type returned for elements found inside a DataWrapper.
csv_row	Type returned for elements found inside a CsvWrapper.

- -name

  An optional Boolean option that specifies that the data requested is the name of the configuration element. As shown in Figure 5-2 on page 943, the name is the complete name including all hierarchical levels. If, at any level in the hierarchy, the *leaf_name* is not unique, the *leaf_name* is replaced by a positional index so as to have a unique name for each element. For example, if there are two instances of the same wrapper called “port(abc)” inside the same parent wrapper called “ports”, the name would be reported as “/Ports/Port(<2>)” assuming it is the third element inside “/ports”.

- -id position | id_name | all

  An optional switch and value pair that specifies that the data requested is the id of a wrapper or of a repeatable property. When using -id all, the command returns a well-formatted Tcl list unless the configuration element has metadata and there is only one non repeatable Id declared for it. In this case the value is returned as a Tcl string, which is the case for most wrappers and repeatable properties you introspect.

  When you have configuration elements with more than one Id, you can query one id at a time using either its position or when it has metadata, the id of the ID wrapper that declares the id. A position is represented with an integer surrounded by <>. For example, <0> is the first value, <1> is the second, and so on. In such a case, the id is returned as a string except when using the id of a repeatable id in which case a well-formatted Tcl list is returned.

- -value_as_list

  An optional switch that specifies that the command always return a list, even for properties that can hold a single value. Using this switch, it is possible to process a collection of mixed properties: where some can hold a single value (when the *value_id* of the metadata of the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

property does not end with a wildcard character, such as Value(port_pin_name)), or many values (when the value_id in metadata ends with a wildcard, such as Value(port_pin_name*)).

For example, consider the following properties from the EdtChannelsIn wrapper of the DftSpecification. The pipeline_clock property is specified with Value(port_pin_name) in the metadata, and the property port_pin_name is specified with Value(port_pin_name*):

```
EdtChannelsIn {
  pipeline_clock : "/modInst/\subMod[0] /clk";
  port_pin_name : "\port[1] [0]";
}
```

The command

```
> get_config_value pipeline_clock -in_wrapper $edtChannelInObj
```

returns the value of a Tcl string /modInst/\subMod[0] /clk, which Tcl interprets as a list of two elements: /modInst/\subMod[0] and clk, if iterated as a list.

By comparison, the command:

```
> get_config_value port_pin_name -in_wrapper $edtChannelInObj
```

returns a Tcl list of one element, {\port[1] [0]}, because the port_in_name property can be multi-valued.

Using the -value_as_list switch ensures that get_config_value always returns a list, even for the pipeline_clock property:

```
> set portPinPathList [get_config_value pipeline_clock \
  -value_as_list -in_wrapper $edtChannelInObj]
{/modInst/\subMod[0] /clk}
> set portPinPathList [get_config_value port_pin_name \
  -value_as_list -in_wrapper $edtChannelInObj]
{\port[1] [0]}
```

This way it is possible to iterate over the two properties in a single loop and process the right number of values for each, because they are now valid Tcl lists.

- -value_id position | value_name | all

  An optional switch and value pair that specifies that the data requested is the value of a property, repeatable property, or of a row inside a Data or CSV wrapper. If none of the options listed between -value_count and -reference_object inclusively are specified, the tool uses “-value_id all”.

  When using “-value_id all”, the command returns a well-formatted Tcl list unless the configuration element has metadata and there is only one non-repeatable value declared for it. In this case, the value is returned as a Tcl string, which is the case for most properties you introspect.

  When you have configuration elements with more than one elements, you can query one value at a time using either its position or, if it has metadata, the *id* of the Value wrapper that declares the value. A position is represented with an integer surrounded by <>. For example,

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

<0> is the first value, <1> is the second, and so on. In such a case, the value is returned as a string except when using the id of a repeatable Value in which case a well-formatted Tcl list is returned.

- -is_specified

  An optional Boolean option that specifies that the data requested is a Boolean value that is 1 when the configuration element is specified and 0 when it is not. Remember that singular wrappers and properties exist as soon as their parent wrapper exists and regardless of whether the singular wrapper or property is specified. The parent wrapper must exist before you can query if one of its element is specified. Unspecified elements are not included in the output file when using the write_config_data or the report_config_data commands unless the -show_unspecified option is used.

- -has_inherited_default

  An optional Boolean option that specifies that the default is inherited. It returns true when the property is declared with the ReferenceWithInheritedDefault property. For more information on the ReferenceWithInheritedDefault property, see "Wrapper" on page 4759.

- -id_count

  An optional Boolean option that specifies that the data requested is the number of ids specified for the wrapper of repeatable property. A value of 0 is returned for configuration objects that do not have ids like a wrapper declared without any Id wrapper or a property.

- -value_count

  An optional Boolean option that specifies that the data requested is the number of values specified for the property, the Data or CSV wrapper row. A value of 0 is returned for configuration objects that do not have values like a wrapper. Implicit Boolean properties still report one value because the true or false value is implicit but still present.

- -exists

  An optional Boolean option that specifies that the data requested is a Boolean value that is 1 when the configuration element exists and 0 when it does not. Remember that singular wrappers and properties exist as soon as their parent wrapper exists and whether the singular wrapper or property is specified or not. You use the -is_specified option to know if such elements are specified or not.

- -root_name

  An optional Boolean option that specifies that the data requested is the name of the root wrapper in which the configuration element exists. If the element is a root wrapper, its name is returned.

- -root_object

  An optional Boolean option that specifies that the data requested is a collection containing a root wrapper in which the configuration element exists. If the element is a root wrapper, it is returned in the collection.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -leaf_name

  An optional Boolean option that specifies that the data requested is the leaf name of the configuration element. As shown in Figure 5-2 on page 943, the leaf name is the last level of hierarchy including the base_name and the ids when they exist.

- -base_name

  An optional Boolean option that specifies that the data requested is the base name of the configuration element. As shown in Figure 5-2 on page 943, the base name is the last level of hierarchy excluding the ids when they exist.

- -meta_name

  An optional Boolean option that specifies that the data requested is the name of the Wrapper wrapper used to define the configuration element. The name is constructed assuming the metadata data was declared with no Wrapper reference property to shared declaration. The -reference_name option is used to get the name of the element referenced.

- -meta_object

  An optional Boolean option that specifies that the data requested is a collection containing the Wrapper wrapper used to define the configuration element. The object is the metadata data element as if it was declared with no Wrapper reference property to shared object. The reference_object option is used to get the referenced object.

- -meta_id

  An optional Boolean option that specifies that the data requested is a unique integer corresponding to the Wrapper wrapper used to define the configuration element. The meta_id of an element declared with the Wrapper reference property is the meta_id of the referenced element.

- -partition_name

  An optional Boolean option that specifies that the data requested is the name of the partition in which the configuration element exists. Table 5-3 shows the returned value based on which partition the configuration element exists in.

- -comment

  An optional Boolean option that specifies that the comment associated with the configuration element is to be returned as a string.

- -default_value

  An optional switch that specifies that the data requested is the default value of a property. The command returns a well-formatted Tcl list unless the configuration element has metadata and there is only one non-repeatable value declared for it. In this case, the command returns the default_value as a Tcl string, which is the case for most properties you introspect.

  You can use the -value_id option in combination with the -default_value to obtain the default value of a specific value. When requesting the default value of a single value, the return value is always a well-formatted Tcl string.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -tolower

  An optional Boolean option that can be used when the command is returning a string. It is ignored otherwise. All upper case letters are converted to lower case.

- -toupper

  An optional Boolean option that can be used when the command is returning a string. It is ignored otherwise. All lower case letters are converted to upper case.

- -file_name

  An optional Boolean option that specifies that the data requested is the filename from which the configuration element was read. It returns null if the configuration element was not read from a file using read_config_data but instead it was created in memory using add_config_element. The *file_name* value is updated to point to the output filename as soon as it is written out using the write_config_data command.

- -object

  An optional Boolean option that specifies that the data requested is a collection containing the configuration element just like add_config_element returns when adding an element.

- -partition *partition_name*

  An optional switch and value pair that is used to specify the partition in which the configuration element exists. When the option is unspecified, the default partition is used, which is where all specifications (DftSpecification and DefaultsSpecification/DftSpecification) are located. You can access the Core wrappers comprising the Tessent Core Description (TCD) file syntax in the tcd partition. Using the partition names meta: and meta:tcd enables you to access the metadata configuration elements that define the default and tcd partitions. The table below summarizes the available partition names.

Table 5-3. Available Partitions for Introspection

Partition name	Configuration data contained
	When the -partition option is unspecified, the tool accesses the default partition, which is where the DftSpecification and DefaultsSpecification/DftSpecification configuration data is located.
tcd	When the -partition option is specified with the tcd value, the Core wrappers comprising the Tessent Core Description file syntax is accessed.
meta	When the -partition option is specified with the meta value, the metadata configuration data defining the permitted syntax of the DftSpecification and DefaultsSpecification/DftSpecification is accessed.
meta:tcd	When the -partition option is specified with the meta:tcd value, the metadata configuration data defining the permitted syntax of the Core wrappers is accessed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -silent

  An optional argument that specifies to ignore objects that do not exist and not to generate error messages.

Examples

This example shows usage of the get_config_value command with a wide variety of its options exercised. You can see the syntax of the configuration data at the top and the specific configuration data being introspected just after. At the bottom of the example, you can see the metadata defining the syntax.

set_context dft -rtl

report_config_syntax SingularWrapper

```
SingularWrapper {
  WrapperWithRepeatableId(<name*>) {
    WrapperWithRepeatableId(<name*>) {
      // Same as /SingularWrapper/WrapperWithRepeatableId
    }
    time_property    : <time>        ; // default: 3.5ns
    integer_property : <integer>     ; // default: 3
    binary_property  : <binary>      ; // default: 3'b0
    string_property  : <string>, ... ;
    CsvWrapper(<module_name>,<design_id>) {
      <value>,...;
    }
    DataWrapper_with_no_value {
      <port> ; // repeatable
    }
    DataWrapper_value {
      <port> : <period> ; // repeatable
    }
  }
}
```

read_config_data ./data/cfg

report_config_data SingularWrapper

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
SingularWrapper {
  WrapperWithRepeatableId(a,b c) {
    WrapperWithRepeatableId(abc) {
      time_property : 4.5ns;
      integer_property : 6;
      binary_property : 4'b110;
      string_property : this is a string;
      CsvWrapper(moda,id1) {
        a;
        b, c;
        d, e f, g;
      }
      DataWrapper_with_no_value {
        a string ;
        1;
      }
      DataWrapper_value {
        clka : 4ns;
        clkb : 6.7ms;
        clkc : 4;
      }
    }
  }
}
```

set wrp [get_config_el WrapperWithRepeatableId(abc) -hier]

Case 1:

Get the value of a property where the type is time:

```
> puts [get_config_value time_property -in $wrp]
4.5ns
```

Case 2:

Get the time value in milliseconds with units removed:

```
> puts [get_config_value time_property -in $wrp -time_unit ms]
4.5e-06
```

Case 3:

Get the time value in nanoseconds with units removed:

```
> puts [get_config_value time_property -in $wrp -time_unit ns]
4.5
```

Case 4:

Get the value of a property where the type is integer:

```
> puts [get_config_value integer_property -in $wrp]
6
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Case 5:

Get the value of a property where the type is binary:

```
> puts [get_config_value binary_property -in $wrp]
4'b110
```

Case 6:

Get the value of a property with a binary value converted in hexadecimal:

```
> puts [get_config_value binary_property -in $wrp -number_format hex]
4'h6
```

Case 7:

Get the value of a property with a binary value converted to integer:

```
> puts [get_config_value binary_property -in $wrp -number_format int]
6
```

Case 8:

Get the value of a property where the type is string:

```
> puts [get_config_value string_property -in $wrp]
this is a string
```

Case 9:

Get the leaf_name of a wrapper:

```
> puts [get_config_value $wrp -leaf_name]
WrapperWithRepeatableId(abc)
```

Case 10:

Get the base_name of a wrapper:

```
> puts [get_config_value $wrp -base_name]
WrapperWithRepeatableId
```

Case 11:

Get a cvs wrapper object and return the number of rows in csv wrapper:

```
> set csv [get_config_value CsvWrapper(moda,id1) -in $wrp -object]
> puts [get_config_element -in $csv -count]
3
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Case 12:

Get the values of CSV rows:

```
> puts [get_config_value CsvWrapper(moda,id1)/<0> -in $wrp]
a
> puts [get_config_value CsvWrapper(moda,id1)/<1> -in $wrp]
b c
> puts [get_config_value CsvWrapper(moda,id1)/<2> -in $wrp]
d {e f} g
```

Case 13:

Get the number of values in a row:

```
> puts [get_config_value CsvWrapper(moda,id1)/<2> -in $wrp -value_count]
3
```

Case 14:

Get specific value of a CSV row:

```
> puts [get_config_value CsvWrapper(moda,id1)/<2> -in $wrp -value_id <1>]
e f
```

Above is equivalent to

```
> puts [lindex [get_config_value CsvWrapper(moda,id1)/<2> -in $wrp] 1]
e f
```

Case 15:

Get the number of the id for a wrapper element:

```
> puts [get_config_value -id_count \
          SingularWrapper/WrapperWithRepeatableId<0>]
2
```

Case 16:

Get the leaf_name of a wrapper element:

```
> puts [get_config_value -leaf_name $wrp]
WrapperWithRepeatableId(abc)
```

Case 17:

Get the name of a wrapper element:

```
> puts [get_config_value $wrp]
SingularWrapper/WrapperWithRepeatableId(abc)
```

Case 18:

Get the base_name of a wrapper element:

```
> puts [get_config_value -base_name $wrp]
WrapperWithRepeatableId
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Case 19:

Get the parent_name of a wrapper element:

```
> puts [get_config_value -parent_name $wrp]
/SingularWrapper/WrapperWithRepeatableId(a,b c)
```

Case 20:

Get the parent_object of a wrapper element:

```
> puts [get_single_name [get_config_value -parent_object $wrp]]
/SingularWrapper/WrapperWithRepeatableId(a,b c)
```

Case 21:

Get the root_name of a wrapper element:

```
> puts [get_config_value -root_name $wrp]
/SingularWrapper
```

Case 22:

Get the root_object of a wrapper element:

```
> puts [get_single_name [get_config_value -root_object $wrp]]
/SingularWrapper
```

Case 23:

Get the file_name of a wrapper element:

```
> puts [get_config_value -file_name $wrp]
/home/abc/data/ex.cfg
```

Case 24:

Gets the meta_name of a wrapper element:

```
> puts [get_config_value -meta_name $wrp]
/Wrapper(SingularWrapper)/Wrapper(WrapperWithRepeatableId)
```

Case 25:

Gets the ref_name of a metadata element. Returns null when the element is not referencing another one:

```
> puts [get_conf_value \
       SingularWrapper/WrapperWithRepeatableId/WrapperWithRepeatableId \
       -part meta -ref_name]
/Wrapper(SingularWrapper)/Wrapper(WrapperWithRepeatableId)
> puts [get_conf_value SingularWrapper/WrapperWithRepeatableId  \
       -part meta -ref_name]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Case 26:

Uses -tolower and -toupper options to convert the returned string in lower case or upper case:

```
> puts [get_config_value -base_name $wrp]
WrapperWithRepeatableId
> puts [get_config_value -base_name $wrp -toupper]
WRAPPERWITHREPEATABLEID
> puts [get_config_value -base_name $wrp -tolower]
wrapperwithrepeatableid
```

Case 27:

Reports the metadata that defines the syntax being introspected:

report_config_data SingularWrapper -partition meta

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Wrapper(SingularWrapper) {
  Repeatable : single;
  Wrapper(WrapperWithRepeatableId) {
    id(name*) {
      type : string;
    }
    Reference = /SingularWrapper/WrapperWithRepeatableId;
    Property(time_property) {
      Value(time) {
        type : time;
        DefaultValue : 3.5ns;
      }
    }
    Property(integer_property) {
      Value(integer) {
        type : int;
        DefaultValue : 3;
      }
    }
    Property(binary_property) {
      Value(binary) {
        type : binary;
        DefaultValue : 3'b0;
      }
    }
    Property(string_property) {
      Value(string*) {
        type : string;
        DefaultValue : "";
      }
    }
    CSVWrapper(CsvWrapper) {
      Repeatable : on;
      id(module_name) {
        Type : String;
      }
      id(design_id) {
        Type : String;
        IgnoreCase : On;
      }
    }
    DataWrapper(DataWrapper_with_no_value) {
      RowIdElement(port) {
        Type : String;
        repeatable : on;
      }
    }
    DataWrapper(DataWrapper_value) {
      RowIdElement(port) {
        Type : String;
        repeatable : on;
      }
      Value(period) {
        Type : time;
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

set_config_value

add_config_element

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_context

Context: unspecified, all contexts

Mode: all modes

Introspects the current context.

Usage

```
get_context [-atpg | -design_identifier | -edt |
    -extraction | -failure_mapping | -full |
    -ijtag | -logic_bist | -no_rtl | -no_sub_context |
     -rtl | -scan | -scan_diagnosis | -scan_retargeting |
     -silicon_insight | -ssn | -test_points]
```

Description

When specified without an option, this command returns dft or patterns. When specified with the -full option, this command returns the entire context, including any sub-context. When specified with one of the other available options, this command returns 1 if the specified sub-context is active.

Arguments

- -atpg

  An option that returns 1 when "set_context -scan" is specified. When set_system_mode analysis is run and no Logic BIST configuration is detected, the -atpg sub-context is set to 1.

- -design_identifier

  An option that returns the specified or inferred design_identifier value.

- -edt

  An option that returns 1 when "set_context -scan" is specified. When set_system_mode analysis is run and the EDT configuration is detected, the -edt sub-context is set to 1.

- -extraction

  An option that returns 1 when the following conditions are met:

  - The "set_context -ijtag" command was specified.
  - The design was elaborated with the set_current_design command but no top-level ICL module existed at the time of elaboration.
  - The current system mode is setup.

  When the preceding conditions are present, the top-level ICL module is extracted when you run the "set_system_mode analysis" command. If there are no child ICL modules matched to design modules, the tool generates an error.

- -failure_mapping

  An option that returns 1 when "set_context -failure_mapping" is specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -full

  An option that returns a string that fully describes the current context, sub-context(s), and additional switches. The -full option also returns the license option when the license was specified with the set_context command. This option can be used to save the current context to be restored later with the set_context command.

  To save the full context, you can store it in a Tcl variable using this command:

  ```
  set full_context [ get_context -full]
  ```

  To restore the full context, use this command:

  ```
  set_context {*}$full_context
  ```

  The {*} prefix breaks the list of arguments into separate arguments for the set_context command.

- -ijtag

  An option that returns 1 when "set_context -ijtag" is specified.

- -logic_bist

  An option that returns 1 when "set_context -scan" is specified. When set_system_mode analysis is run and a Logic BIST configuration is detected, the -logic_bist sub-context is set to 1.

- -no_rtl

  An option that returns 1 when "set_context -no_rtl" is either specified or inferred. It is inferred in all patterns contexts and in the dft -scan and -edt contexts.

- -no_sub_context

  An option that returns 1 when the set_context was specified with no sub_context options such as -scan or -edt. The -rtl/-no_rlt options on the set_context command have no impact on the get_context -no_sub_context value.

- -rtl

  An option that returns 1 when "set_context -rtl" is specified.

- -scan

  An option that returns 1 when "set_context -scan" is specified.

- -scan_diagnosis

  An option that returns 1 when "set_context -scan_diagnosis" is specified.

- -scan_retargeting

  An option that returns 1 when "set_context -scan_retargeting" is specified.

- -silicon_insight

  An option that returns 1 when "set_context -silicon_insight" is specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -ssn

  An option that returns 1 when there are active SSH instances in analysis mode in one of the following contexts:

  - patterns -scan
  - patterns -scan_diagnosis
  - patterns -scan_retargeting

- -test_points

  An option that returns 1 when "set_context -test_points" is specified.

Return Values

Returns dft or patterns when no option is specified. Returns 0 or 1 when an options is specified.

Examples

Example 1

In the following example, a script introspects the current context and displays an appropriate message.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
if {[get_context] == "dft"} {
   if {[get_context -scan]} {
      puts "in dft scan context"
   } elseif {[get_context -edt]} {
      puts "in dft edt context"
   } else {
      if {[get_context -rtl]} {
         puts "in dft rtl context"
      } else {
         puts "in dft no_rtl context"
      }
   }
} else { ;# patterns -> get_context is patterns
   if {[get_context -scan]} {
      if {[get_context -atpg]} {
         if {[get_context -edt]} {
            puts "in patterns scan edt atpg context"
         } else {
            puts "in patterns scan atpg context"
         }
      } else {;# logic_bist -> get_context -logic_bist is 1
         puts "in patterns scan logic_bist context"
      }
   } elseif {[get_context -ijtag]} {
      if {[get_context -extraction]} {
         puts "in ijtag icl_extraction context"
      } else {
         puts "in ijtag retargeting context"
      }
   } else {;# scan_diagnostic
      puts "in scan_diagnostic context"
   }
}
```

Example 2

This demonstrates the result of the -full switch.

set_context dft -rtl
get_context -full

```
dft -rtl -design_identifier "rtl"
```

Related Topics

report_context

set_context

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_current_design

Context: all contexts

Mode: all modes

Returns a collection of one element that specifies the top-level design when previously set.

Usage

dft -no_rtl and patterns -no_rtl Contexts

get_current_design [-icl | -created_from_tcd | -created_from_icl]

dft -rtl and patterns -rtl Contexts

get_current_design [-icl | -top_config_dictionary | -created_from_tcd | -created_from_icl]

Description

Returns a collection of one element that specifies the top-level design when previously set.

The get_current_design command returns an empty collection when the top-level design has not been previously defined using the set_current_design.

Arguments

- -icl

  An optional switch that returns the name of an ICL module object if that object exists. The switch returns an error if there is no ICL module associated with the current design.

- -top_config_dictionary (“set_context *context* -rtl” only)

  An optional switch that returns the name and library of the configuration of VHDL top module. You use this switch to determine if the design was elaborated using a top-level configuration or using the top module directly. When the top module was not set via a configuration, it returns a null.

  The format for the returned dictionary is:

  ```
  config_name <name> library_name <library_name>
  ```

- -created_from_tcd

  An optional switch that returns true if the *current_design* is created from a Tessent Core Description (TCD), otherwise it returns false.

- -created_from_icl

  An optional switch that returns true if the *current_design* is created from an already-loaded ICL module that is associated with the *current_design*, otherwise it returns false.

Examples

Example 1

get_current_design

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{}

read_verilog top.v
set_current_design top
get_current_design

{top}

puts [get_current_design]

@45

puts [get_single_name [get_current_design]]

top

if { [get_single_name [get_current_design]]== "top" } { puts "yes" }

yes
```

The first get_current_design command returns an empty collection because no current design has been set. The second get_current_design command returns a collection consisting of the name of the top-level design that was previously set. The first “puts” command returns only the pointer (@45) to a collection in memory. The second “puts” command shows how to use the get_single_name command to return the name of the module object returned by get_current_design.

Example 2

This example shows how to use the -top_config_dictionary switch to determine if the design was elaborated using a top-level configuration or using a top module directly.

```
set top_config_dictionary [get_current_design -top_config_dictionary]
if {$top_config_dictionary ne ""} {
  set top_config_name [dict get $top_config_dictionary config_name]
  set top_library_name [dict get $top_config_dictionary library_name]
} else {
  set top_module [get_current_design]
}
```

Related Topics

get_name_list

set_current_design

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_current_mode

Context: patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis, patterns -failure_mapping

Mode: setup, analysis

Returns the current test mode, as specified by the set_current_mode command.

Usage

get_current_mode [-type]

Description

Returns the current test mode, as specified by the set_current_mode command.

If you do not include the -type switch, the command returns the name of the test mode. If you include the -type switch, this command returns the mode type.

Arguments

- -type

  An optional switch that returns the mode type. Possible return values are: unwrapped, internal, external, and retargeting. For more information about these test modes, refer to the set_current_mode command description.

Examples

This example returns the test mode as “unwrapped.”

get_current_mode

```
unwrapped
```

Related Topics

set_current_mode

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_current_physical_block

Context: dft -edt, dft -scan, dft -test_points, patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: setup, analysis

Returns the current physical block specified with the set_current_physical_block command.

Usage

get_current_physical_block

Description

Returns the current physical block specified with the set_current_physical_block command.

Arguments

None.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_current_silicon_insight_setup

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Returns the currently selected SiliconInsight setup as specified in the TSI setup spec.

Usage

get_current_silicon_insight_setup

Arguments

None

Examples

get_current_silicon_insight_setup

```
Sid(simdut)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_current_simulation_context

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Returns the name of the current simulation context.

Usage

get_current_simulation_context

Description

Returns the name of the current simulation context.

The get_current_simulation_context command returns the current simulation context. For a list of predefined simulation contexts, refer to Table 8-6 on page 2434.

Arguments

None.

Examples

The following example shows how to set and return the current simulation context:

SETUP> get_ current_simulation_context

```
stable_after_setup
```

SETUP> set_current_simulation_context stable_load_unload
SETUP> get_ current_simulation_context

```
stable_load_unload
```

Related Topics

add_simulation_context

copy_simulation_context

delete_simulation_contexts

get_simulation_context_list

report_simulation_contexts

set_current_simulation_context

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_default_ssn_datapath_configuration

Context: all contexts

Mode: all modes

Retrieves an override setting of an ICL primary target on a SSN datapath.

Usage

get_default_ssn_datapath_configuration [-instance *design_instance*]
-target *icl_pin_port_or_reg*

Description

Retrieves a PDL value overriding a default setting of an ICL primary target on a SSN datapath.

Refer to the command "set_default_ssn_datapath_configuration" on page 2441 to learn more about the purpose and the use model of the default SSN datapath configuration and how set_default_ssn_datapath_configuration assigns settings to the PDL values of ICL targets to override their default state.

Arguments

- -instance *design_instance*

  An optional switch and string pair that specifies the design instance portion of the instance/target pair from which to retrieve the PDL value. If you omit this switch, the tool treats the -target specification as relative to the current design.

- **-target *icl_pin_port_or_reg***

  A required switch and value pair that specifies a primary ICL input port, an ICL pin, or an ICL ScanRegister. The icl_pin_port_or_reg parameter can also specify a sub-range. Specify the target in the exact same way as assigned by the set_default_ssn_datapath_configuration command.

Examples

The following example sets two PDL values of two ICL targets and then retrieves their values.

```
SETUP> set_default_ssn_datapath_configuration -target cfginst.din -value 0b1101xx10
SETUP> set_default_ssn_datapath_configuration -target ssnMuxCtrl -value Upper

SETUP> get_default_ssn_datapath_configuration -target cfginst.din
0b1101xx10
SETUP> get_default_ssn_datapath_configuration -target ssnMuxCtrl
Upper
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_defaults_value

Context: dft, patterns, unspecified

Mode: setup, analysis, insertion

Obtains the default value of a property as specified in one of the DefaultsSpecification(company | user | group)/DftSpecification or DefaultsSpecification(company | user | group)/ PatternsSpecification wrappers.

Usage

get_defaults_value ***property_path_name***
[-policy *policy*]
[-time_unit *time_unit*] [-number_format *format*]
[-is_specified] [-tolower] [-toupper]

Description

Obtains the default value of a property as specified in one of the DefaultsSpecification(company | user | group)/DftSpecification or DefaultsSpecification(company | user | group)/ PatternsSpecification wrappers.

When the -policy option is not specified, it returns the value specified in the DefaultsSpecification(user) policy, the value specified in the group policy if not specified in the user policy, the value specified in the company policy if not specified in the user or group policy, or the tool default if not specified in any policy.

When automatically resolving the priority, the default value of the property is accessed.

Arguments

- ***property_path_name***

  A required string that defines the path to a property without the “DefaultsSpecification()/” prefix. For the DftSpecification, this path starts with “DftSpecification/”. For the PatternsSpecification, this path starts with “PatternsSpecification/”.

- -policy *policy*

  An optional switch and value pair that defines from which DefaultsSpecification wrapper the default value is requested. If not specified, the command returns the value specified in the user policy, the value specified in the group policy if not specified in the user policy, the value specified in the company policy if not specified in the user or group policy, or the tool default if not specified in any policy.

- -time_unit *time_unit*

  An optional switch and value pair that is only relevant when obtaining the value of a property of type time. This option enables you to request the time expressed in the units you want. For more information see the description of the get_config_value -time_unit option.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -number_format *format*

  An optional switch and value pair that is only relevant when obtaining the value of a property of type binary. This option enables you to request the binary number expressed in the format you want. For more information see the description of the get_config_value -number_format option.

- -is_specified

  An optional Boolean option that specifies that you are not requesting the actual value of the property but instead requesting status on whether the property is specified or not in the given policy. If the -policy option is not specified, the tool returns 1 if it is specified in any of the policy wrappers. If the -policy option is not specified and the tool returns 0, then you know you are getting the tool default value.

- -tolower

  An optional switch and value pair that is only relevant when obtaining the value of a property of type string. This option enables you to request the string with all upper-case letters converted to lower case.

- -toupper

  An optional switch and value pair that is only relevant when obtaining the value of a property of type string. This option enables you to request the string with all lower-case letters converted to upper case.

Examples

The following example queries the default value of a property and then checks which policy specified it:

```
set p DftSpecification/IjtagNetwork/HostScanInterface/Block/scan_in?

get_defaults_value $p

my_scan_in

foreach policy {user group company} {
  if {[get_defaults_value $p -policy $policy -is_specified]} {
    break
  }
}
puts "'$p' is specified by policy $policy'."
'DftSpecification/IjtagNetwork/HostScanInterface/Block/scan_in' is specified by policy
'group'.
```

Related Topics

create_dft_specification

set_defaults_value

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_design_level

Context: dft

Mode: setup, analysis, insertion

Returns the design level previously defined with the set_design_level command.

Usage

get_design_level

Description

Returns the design level previously defined with the set_design_level command.

When a new design is elaborated, the level goes back to being unspecified. In this case, the get_design_level command returns a null string. This commands returns “chip” when you used “set_design_level top” or “set_design_level chip”.

Arguments

None

Examples

The following example queries the level of the current design and stores the result in a variable.

```
set design_level [get_design_level]
```

Related Topics

set_current_design

set_design_level

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_design_objects

Context: All

Mode: All

Returns a collection of design objects associated to the object_ids you supply.

Usage

get_design_objects {**-from_ids** ***object_ids*** | **-from_objects** ***from_objects***}
[-map_pseudo_ports]

Description

This command converts a list of ***object_ids*** or objects to a collection of objects. Each design object type (module, instance, port, pin, net, and pseudo_port) has a unique ***object_id***, as described in the "Hierarchical Design Data Model" section.

The value of these attributes consists of a string followed by a colon and a unique integer per object type. The string is the name of the object type. For example, the object_id of a module is module:integer. Use this unique string to construct a dictionary for objects, using the ***object_id*** as the key. Use the ***object_ids*** with the "get_design_objects -from_ids" command to get back to the objects. Using the ***object_id*** rather than the name of the object is more efficient and more robust because it is much shorter and does not change during synthesis.

The object names found inside a generate block in RTL change during synthesis. For example, the instance u1 inside generate block b1 has a name "b1.u1" in RTL but "\b1.u1 " after quick synthesis. When you build a dictionary in analysis mode with the synthesized name as a key, you cannot find the object with that name in insertion mode because the RTL view is restored in insertion mode. When you store the ***object_id***, you can find the object in either the synthesized view or the RTL view. See the example later for an illustration of this behavior.

Arguments

- **-from_ids** ***object_ids***

  A required switch-value pair that is used to supply a Tcl list of ***object_ids***. The ***object_ids*** of the objects are obtained using the "get_attribute_value_list -name ***object_id***" command. An ***object_id*** can only be used to get back the objects for as long as the design remains elaborated in memory. There is no guarantee that an ***object_id*** for a given object remains the same the next time the design is elaborated. It is exactly like the gate_id of a gate_pin object that is only valid while the flat model is in memory. Never save a dictionary to disk containing ***object_ids*** and expect the ***object_ids*** to have meaning when you reload the dictionary at a later time. The ***object_ids*** are only stored in memory and reused while the design remains in memory, as shown in the example.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-from_objects *from_objects***

  A required switch-value pair that is used to supply a Tcl list of objects. The ***from_objects*** may contain ports, pseudo_ports, pins, nets, and gate_pins. A port, pin, or net is passed through to the result collection.

- -map_pseudo_ports

  An optional switch that specifies a pseudo_port is mapped to the corresponding pins or nets. Otherwise, it passes through to the result collection. A gate_pin is mapped to the corresponding port, pseudo_port, pin, or net. If it maps to a pseudo_port and you specify "-map_pseudo_ports", that pseudo_port is further mapped to the corresponding pins or nets as described above.

Examples

Example 1

The following example shows that the name of an object changes with synthesis but the ***object_id*** does not. The ***object_id*** stored in analysis from the synthesized view is valid to get the RTL object in insertion mode.

```
SETUP> set design_mods [get_modules -of_type design]
SETUP> set_attribute_value $design_mods -name synthesize_before_analysis
SETUP> set_system_mode analysis

//  ----------------------------------------------------------------------
//  Begin RTL synthesis.
//  --------------------
//  Module 'FF' synthesized.
//  Module 'elt' synthesized.
//  Module 'subBlock1' synthesized.
//  Module 'subBlock2' synthesized.
//  RTL synthesis completed, synthesized modules=4, Time=1.83 secs.
//  ----------------------------------------------------------------------

ANALYSIS> set subBlock1_instances [get_instance -of_module subBlock1]
ANALYSIS> set subBlock1_instance1 [index_collection $subBlock1_instances 1]
ANALYSIS> set name [get_single_name $subBlock1_instance1]
ANALYSIS> set object_id [get_attribute_value_list $subBlock1_instance1 \
                         -name object_id]
ANALYSIS> puts [get_single_name [get_instance [list $name]]]

option1.subBlock2_I2 /subBlock1


ANALYSIS> puts [get_single_name [get_design_objects -from_ids $object_id]]

option1.subBlock2_I2 /subBlock1

ANALYSIS> set_system_mode insertion

# The object name changed when restoring the RTL view but the
# object_id remained the same

INSERTION> puts [get_single_name [get_instance [list $name]]]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Warning: No result.
```

INSERTION> puts [get_single_name [get_design_objects -from_ids $object_id]]

```
option1.subBlock2_I2/subBlock1
```

Example 2

The following example shows how to map pseudo ports and gate_pins to port, pin, and net objects. Any pin, port, or net objects found in $col1 passes through unchanged. Any gate_pin object is mapped to its associated port, pin, or net object. If the gate_pin object is inside a cell and has no associated port, pin, or net objects, it is silently deleted. Finally, any pseudo_port object found in $col1 is mapped to its associated port, pin, or net object because the -map_pseudo_ports option is used. When that switch is not used, the pseudo_port objects pass through unchanged.

set design_object [get_design_objects -from_objects Scol1 -map_pseudo_ports]

Related Topics

get_attribute_value_list

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_design_sources

Context: unspecified, all contexts

Mode: setup, analysis, insertion

Returns the pathnames or file extensions previously specified with the set_design_sources command.

Usage

get_design_sources [-format {verilog | icl | tcd_bscan | tcd_scan | tcd_fbox | tcd_memory | tcd_memory_cluster}]
[-path_list | -extension_list | -file_dictionary | -search_design_load_path]

Description

Returns the pathnames or file extensions previously specified with the set_design_sources command.

By default, the get_design_sources command returns a list of paired values that specify the pathnames to all design sources. The first item in the pair is -v or -y to indicate whether the pathname is a file or a directory, respectively. This is useful when you want to add additional design sources because any invocation of the set_design_sources command overwrites previous invocations.

Arguments

- -format {verilog | icl | tcd_bscan | tcd_scan | tcd_fbox | tcd_memory | tcd_memory_cluster}

  An optional switch and literal pair that enables you to choose the output format. By default, the command specifies Verilog modules.

- -path_list

  An optional switch that returns a list of pathnames. This is the default.

- -extension_list

  An optional switch that returns a list of file extensions for the specified format.

- -file_dictionary

  An optional switch that returns a Tcl dictionary that contains information about the design files that the tool has read and written. This can be useful, for example, if you need to create a script for a downstream tool that needs to read the correct design files with the correct options in the correct sequence. For a description of the file dictionary format, see the description of the *<design_name>.design_source_dictionary* component in dft_inserted_designs. You can use this switch only when you also specify “-format verilog”.

- -search_design_load_path

  An optional switch that returns a Boolean value (on or off) that indicates whether the set_design_sources -search_design_load_path option is enabled. You can use this switch only when you also specify “-format icl”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

This example clears previous set_design_sources commands before setting new design sources, and then returns the list of pathnames and the list of extensions:

```
SETUP> set_design_sources -clear
SETUP> set_design_sources

//  No -v/-y files/directories specified.

SETUP> set_design_sources -v  MODB.v  MODC.v -y  /u/joe/testcases -extensions .v .vhd
SETUP> get_design_sources -path_list -v MODB.v -v MODC.v -y /u/joe/testcases
SETUP> get_design_sources -extension_list .v .v.gz .vhd .vhd.gz
```

Related Topics

report_design_sources

set_design_sources

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_dfm_rules

Context: patterns -scan_diagnosis

Mode: analysis

Returns a TCL list of the DFM rules contained in the layout database.

Usage

get_dfm_rules [-header | -col *columns*...]

Description

Returns a TCL list of the DFM rules contained in the layout database.

By default, for each rule the tool returns the id, rule name, layer, and type.

For more information about DFM rules, see "Diagnosis for Design for Manufacturability Analysis."

Arguments

- -header

  An optional switch that returns a list of available column names.

- -col *columns*...

  An optional switch that specifies the column names you want to use for the DFM rule information in the list.

Examples

Example 1

The following example sets the available column names of the DFM report table into a TCL variable and displays the column names.

```
open_layout MyLayoutDB
set columns [ get_dfm_rules -header ]
puts "columns: $columns"
```

```
columns: id rule layer type
```

Example 2

The following example creates a custom report by inspecting the DFM rules:

```
foreach { id rule_name layer_name type_name }  [ get_dfm_rules ] {
    puts "$id $rule_name $layer_name $type_name"
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
1 minimum_space_4_line_route1 route_1 BRIDGE
2 metal_cross_edge_route1 route_1 OPEN
3 low_density_min_space_route2 route_2 BRIDGE
4 half_comb_route3 route_2 OPEN
5 metal_cross_edge_route1 route_3 OPEN
6 corner_to_interior_metal4 route_4 BRIDGE
7 divergent_line_with_jog_metal5 route_5 BRIDGE
8 metal_cross_edge_route1 route_5 OPEN
9 route_5_barbell route_5 OPEN
10 route_2_between_vias route_2 OPEN
```

Example 3

The following example customizes the columns for the report:

```
foreach { r t }  [ get_dfm_rules -col rule type ] {
    puts "$r $t"
}
```

```
minimum_space_4_line_route1  BRIDGE
metal_cross_edge_route1  OPEN
low_density_min_space_route2  BRIDGE
half_comb_route3  OPEN
metal_cross_edge_route1  OPEN
corner_to_interior_metal4  BRIDGE
divergent_line_with_jog_metal5  BRIDGE
metal_cross_edge_route1  OPEN
route_5_barbell  OPEN
route_2_between_vias  OPEN
```

Related Topics

report_dfm_rules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_dft_cell

Context: dft

Mode: all modes

Retrieves the named sets of cells to use when test hardware must be inserted (as a Tcl list of the currently available cell selection names), or a particular cell given the name and required cell function of a specified set of cells.

Usage

get_dft_cell -available_cell_selection_names | -default_cell_selection_name | *function_name*
[-with_asynch_enable] [-with_asynch_disable] [-with_set] [-with_reset] [-length *length*]
[-cell_selection_name *dft_cell_selection_name*] [-inputs *number*]
[-with_inverted_inputs *number_inverted_inputs*] [-silent]

Description

Retrieves the named sets of cells to use when test hardware must be inserted (as a Tcl list of the currently available cell selection names), or a particular cell given the name and required cell function of a specified set of cells.

If only one set (one cell_selection) exists, or the "set_cell_library_options -default_cell_selection_name" command was used to set the default cell selection, then the -cell_selection_name switch is optional for retrieving a cell of a given function. Otherwise, -cell_selection_name must be used to identify a specific cell selection.

This command is very useful in enabling you to write a technology-independent command. After you have retrieved the cell module using the get_dft_cell command, you can use the function attribute on the ports of the module to find the ports you want to connect to.

For more information on Tessent cell library attributes, see the *Tessent Cell Library Manual*. More specifically, refer to the dft_cell_selection wrapper description described in the Cell Selection section as well as the function attribute described in Pin Attributes.

You can see a list of the available DFT cell functions by using the "help get_dft_cell" command. Each of the listed functions corresponds one for one with the elements documented in the dft_cell_selection wrapper found in the Cell Selection section of the *Tessent Cell Library Manual*. Many basic functions are learned and automatically assigned to a dft_cell_selection wrapper named default when your library does not contain a dft_cell_selection wrapper.

Note

If your Tessent cell library contains only ATPG model information but you have a *cell.lib* file in the original LogicVision format, you can load it natively using the read_cell_library command, and this populates the dft_cell_selection wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- -available_cell_selection_names

  An optional switch that specifies for the tool to return a Tcl list of names of the currently available DFT cell selections. If none are parsed and one is learned by default, it returns a list with the single element “default”. If there is no cell selection parsed and no default is created (for example, if no library is loaded) it returns null.

- -default_cell_selection_name

  An optional switch that returns null unless there is only one cell selection or the set_cell_library_options -default_dft_cell_selection_name command was used to define a cell selection as the default.

- *function_name*

  An optional string that specifies the DFT cell function from the specified or default dft_cell_selection to return. If the specified function does not exist in the specified dft_cell_selection, the tool reports an error and lists the dft_cell_selection names that contains the requested function_name.

- -with_asynch_enable

  An optional switch that applies to clock_gating_and, clock_gating_or, and clock_shaper cell types. It specifies to return a cell with an asynchronous input to enable clocking at the cell output.

- -with_asynch_disable

  An optional switch that applies to clock_gating_and, clock_gating_or, and clock_shaper cell types. It specifies to return a cell with an asynchronous input to disable clocking at the cell output.

- -with_set

  An optional switch that applies to active_high_latch, active_low_latch, posedge_dff, negedge_dff, posedge_scan_cell, negedge_scan_cell, posedge_synchronizer_cell or negedge_synchronizer_cell and specifies to return a cell with an asynchronous input to set the cell’s state to “1” (high). For all types except posedge_synchronizer_cell and negedge_synchronizer cell, if a cell with exactly what is requested does not exist, the command returns null. However, for the synchronizer cells, the command returns a cell if it has all the features requested and can be tied off outside to produce the requested cell. For example, if a synchronizer_cell with no asynchs is requested (no -with_set or -with_reset switches) and none exists, but a sync cell with one or more asynchs does exist, the command returns it. Similarly, if a sync_cell -with_set is requested, and none with only a set exists, but one with both asynchs does exist, the command returns it.

- -with_reset

  An optional switch that applies to active_high_latch, active_low_latch, posedge_dff, negedge_dff, posedge_scan_cell, negedge_scan_cell, posedge_synchronizer_cell or negedge_synchronizer_cell and specifies to return a cell with an asynchronous input to reset the cell’s state to “0” (low). For all types except posedge_synchronizer_cell and

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

negedge_synchronizer cell, if a cell with exactly what is requested does not exist, the command returns null. However, for the synchronizer cells, the command returns a cell if it has all the features requested, and can be tied off outside to produce the requested cell. For example, if a synchronizer_cell with no asynchs is requested (no -with_set or -with_reset switches) and none exists, but a sync cell with one or more asynchs does exist, the command returns it. Similarly, if a sync_cell -with_reset is requested, and none with only a reset exists, but one with both asynchs does exist, the command returns it.

- -length *length*

  A switch and integer pair that specifies the length of a posedge_synchronizer_cell or a negedge_synchronizer_cell. The length is the number of DFFs in the synchronizer cell and can be specified using a single number or a number:number range. If the dft_cell_selection does not contain a synchronizer_cell of the length specified, the command returns an empty string.

- -cell_selection_name *dft_cell_selection_name*

  A switch and string pair that specifies the name of a currently available cell selection to retrieve cells from. This could be a technology name, for example. This switch is optional if only one dft_cell_selection wrapper is currently loaded. If this switch is not specified and more than one dft_cell_selection wrapper exists, the tool issues an error. Similarly, if an invalid technology is specified, the tool issues an error and lists all available dft_cell_selection names.

- -inputs *number*

  An optional switch and integer pair that specifies the number of inputs on the model. The tool ignores the -inputs switch except in the case of symmetrical primitives {and, nand, or, nor, xor}. The default for symmetrical primitives is 2 unless the -inputs number pair specifies an integer greater than 2; in this case the tool returns a cell with the specified number of inputs if one exists in the specified technology.

- -with_inverted_inputs *number_inverted_inputs*

  An optional switch and integer pair that specifies the number of inverted inputs of the total inputs specified with the -inputs switch. At least one input must be non-inverting. This switch can only be used when *function_name* is “and” or “or”.

- -silent

  An optional switch that specifies to suppress the error message and returns an empty collection.

Return Values

Returns the module objects when no errors are present.

Examples

Example 1

In the following example, the command queries which cell model is available in the tech_name_1 dft_cell_selection wrapper to be used as a clock_buffer.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_dft_cell clock_buffer -cell_selection_name tech_name_1

```
{my_buff}
```

Example 2

In the following example, the command queries which 4-input cell model is available to be used as an and gate.

get_dft_cell and -input 4

```
{and_4}
```

Example 3

In the following example, multiple dft_cell_selection wrappers exist but the -cell_selection_name switch is not specified. As a result, the tool returns an error and lists the available dft_cell_selection names that exist in the library.

get_dft_cell and

```
//  Error: Multiple cell_selection(<selection_name>) exist.
//         Must specify -cell_selection_name followed by one of :
//         tech_name_1
//         tech_name_2
//         tech_name_3
```

Example 4

In the following example, an invalid technology is specified. As a result, the tool returns an error and lists the available technologies.

get_dft_cell and -cell_selection_name foo

```
//  Error: cell_selection(foo) does not exist.
//         -cell_selection_name must currently be followed by one of :
//         tech_name_1
//         tech_name_2
//         tech_name_3
```

Example 5

In the following example, a procedure called intercept_with_cell_body is created to enable intercepting ports, nets, or pins that are either sinks or drivers. The second argument is a cell_selection type string that uses the get_dft_cell command to obtain the corresponding cell from the library, thereby making this command technology-independent. To understand the purpose of each section, read the comments in the commented section headers.

Note, in Section 3, the port attribute called ‘function’ is used to determine the port names to connect on the selected dft cell. You can also refer to the example for the register_tcl_command to see how you can create a cleaner external interface to this command with proper option parsing and help messages.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
#####################################################################
### This array is to not repeat interception into repeated module
### instances when auto_uniquify_edited_modules is off
#####################################################################

variable insertion_parent_leaf_instance_array
array set insertion_parent_leaf_instance_array {}
proc intercept_with_cell_body {args} {

   # Intitialize the allArgs array with the default values of the optional
   # options
   array set allArgs {
      input2 ""
      select ""
      level 1
      clear 0
   }

   #Override the default values of the arguments with the actual values
   # received from
   array set allArgs $args
   set node $allArgs(node)
   set cell_sel $allArgs(cell_sel)
   set in2 $allArgs(input2)
   set sel $allArgs(select)
   set level $allArgs(level)
   set clear $allArgs(clear)

   #####################################################################
   ### Section 1: Defines a global array to know if an interception was
   ### already done inside repeated module instance. clear args set to 1
   ### resets the array
   #####################################################################

   variable insertion_parent_leaf_instance_array

   if {$clear == 1} {
      array unset insertion_parent_leaf_instance_array
      array set insertion_parent_leaf_instance_array {}
   }
   set dft_cell [get_dft_cell $cell_sel]
   #####################################################################
   ### Section 2: Validates node argument and determine if it is a sink or
   ### driver interception
   #####################################################################

   set node_object [check_obj_spec $node <node>]
   set node_type   [get_attribute_value_list $node_object \
                     -name object_type]
   if {$node_type eq "port" } {
      if {[get_attribute_value_list $node_object -name direction] \
            eq "input"} {
         set insertion_type "driver"
      } else {
         set insertion_type "sink"
      }
      set leaf_name [lindex  [get_attribute_value $node_object \
            -name name] 0]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
      set instance_object ""
      set leaf_block_id ""
   } elseif {$node_type eq "pin"} {
      if {[get_attribute_value_list $node_object -name direction] \
            eq "output"} {
         set insertion_type "driver"
      } else {
         set insertion_type "sink"
      }
      set instance_object [get_instances -of_pins $node_object]
      set leaf_name [lindex [get_attribute_value $node_object \
            -name leaf_name] 0]
      set leaf_block_id [lindex [get_attribute_value $instance_object \
            -name leaf_block_id] 0]
   } else {
      set insertion_type "driver"
      set instance_object [get_instances -of_nets $node_object]
      set leaf_name [lindex [get_attribute_value $node_object \
            -name leaf_name] 0]
      set leaf_block_id [lindex [get_attribute_value $node_object \
            -name leaf_block_id] 0]
   }

   ######################################################################
   ### Section 3: Determine if in2 and sel are needed
   ######################################################################

   set connect_in2 0
   set connect_sel 0
   if {$cell_sel == "mux" | $cell_sel == "clock_mux"} {
      set input0 "[get_single_name [get_ports -of_modules $dft_cell \
               -filter {function == mux_in0}]]"
      set input1 "[get_single_name [get_ports -of_modules $dft_cell \
               -filter {function == mux_in1}]]"
      set select "[get_single_name [get_ports -of_modules $dft_cell \
               -filter {function == mux_select}]]"
      set output "[get_single_name [get_ports -of_modules $dft_cell \
               -filter {function == data_out}]]"
      set connect_in2 1
      set connect_sel 1
   } elseif {$cell_sel == "buffer" | $cell_sel == "clock_buffer" | \
               $cell_sel == "inverter" } {
         set input0 "[ get_single_name [get_ports -of_modules $dft_cell \
               -filter {function == data_in}]]"
         set output "[ get_single_name [get_ports -of_modules $dft_cell \
               -filter {function == data_out}]]"
   } else {
      set input0 "[lindex [get_name_list [get_ports -of_modules $dft_cell\
         -filter {function == data_in}]] 0]"
      set input1 "[lindex [get_name_list [get_ports -of_modules $dft_cell\
         -filter {function == data_in}]] 1]"
      set output "[get_single_name [get_ports -of_modules $dft_cell \
         -filter {function == data_out}]]"
      set connect_in2 1
   }

   if {$connect_in2} {
      if {$in2 != ""} {
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
      set in2_object [check_obj_spec $in2 "-input2"]
   } else {
      return -code error "-input2 must be specified when \
         -cell_function_name is two input cell"
   }
}

if {$connect_sel} {
   if {$sel != ""} {
      set sel_object [check_obj_spec $sel "-select"]
   } else {
      return -code error "-select must be specified when \
         -cell_function_name is mux or clock_mux"
   }
}

#####################################################################
### Section 4: Determines location of the interception gate and if it
### was already done for the specified node inside its parent module.
#####################################################################

if {[sizeof_collection $instance_object] ne 0} {
   if {$node_type eq "net"} {
      set leaf_instance_name ""
      set parent_module [get_modules -of_instances $instance_object]
      set parent_instance_name [get_single_name $instance_object]
   } else { # it is a pin
      set leaf_instance_name [lindex [get_attribute_value_list \
         $instance_object -name leaf_name_hash] 0]
      set parent_instance [get_instances -parent_of_instances \
         $instance_object]
      set parent_instance_name [get_single_name $parent_instance]
      if {[sizeof_collection $parent_instance] eq 0} {
         set parent_module [get_current_design]
      } else {
         set parent_module [get_modules -of_instances \
            $parent_instance]
      }
   }
} else {
   set parent_instance_name ""
   set leaf_instance_name ""
   set parent_module [get_current_design]
}
set parent_mod_name [get_single_name $parent_module]
set parent_lib_name [lindex [get_attribute_value_list $parent_module \
   -name library_name] 0]
set parent_arch_name [lindex [get_attribute_value_list $parent_module \
   -name vhdl_architecture] 0]

#######################################################################
# This key matches for insertion made inside repeated instances of
# a module or inside a generate loop block. When it match, the
# interception is skipped for the next invocation
#######################################################################

set key "Level${level}-Cell(${cell_sel})-"
append key \
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
   "${parent_lib_name}::${parent_mod_name}(${parent_arch_name})"
if {$leaf_instance_name ne ""} {
   append key "/${leaf_instance_name}"
}
append key "/${leaf_name}"

#####################################################################
### Section 5: Retrieves leaf instance name of gate if node is a
### repeated module instance and skips the interception
#####################################################################

if {[info exists insertion_parent_leaf_instance_array($key)] && \
      [get_insertion_option -auto_uniquify_edited_modules] eq "off"} {
   set leaf_cell_inst "$insertion_parent_leaf_instance_array($key)"
   if {$leaf_block_id ne ""} {
      set cell_inst [get_instance \
         "${parent_instance_name}/${leaf_block_id}.${leaf_cell_inst}"]
   } else {
      set cell_inst [get_instance \
         "${parent_instance_name}/${leaf_cell_inst}"]
   }
} else {
   if {$leaf_block_id ne ""} {
      set cell_inst_name \
      "${parent_instance_name}/${leaf_block_id}.insertion_${cell_sel}"
   } else {
      set cell_inst_name \
         "${parent_instance_name}/insertion_${cell_sel}"
   }
   no_transcript {
      set cell_inst [create_instance $cell_inst_name -of_modules \
         $dft_cell -allow_instance_uniquification]
   }

   set leaf_name [lindex [get_attribute_value_list $cell_inst -name \
      leaf_name] 0]
   set leaf_block_id [lindex [get_attribute_value_list $cell_inst \
      -name leaf_block_id] 0]
   if {$leaf_block_id ne ""} {
      # Remove the block_id part from the leaf instance name
      set leaf_name [string range $leaf_name \
         [string length $leaf_block_id] end]
   }
   set insertion_parent_leaf_instance_array($key) $leaf_name

   if {$insertion_type == "sink"} {
      no_transcript {
         move_connections -from $node_object \
            -to [get_pins ${input0} -of_instances  $cell_inst]
         create_connections [get_pins ${output} -of_instances \
            $cell_inst] $node_object
      }
   } else {
      no_transcript {
         move_connections -from $node_object -to [get_pins ${output} \
            -of_instances $cell_inst]
         create_connections $node_object [get_pins ${input0} \
            -of_instances  $cell_inst]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
            }
         }
      }

      if {$connect_in2} {
         no_transcript {
            create_connections $in2_object [get_pins $input1 -of_instances \
               $cell_inst]
         }
      }

      if {$connect_sel} {
         no_transcript {
            create_connections $sel_object [get_pins $select -of_instances \
               $cell_inst]
         }
      }
      return $cell_inst
   }

   ##########################################################################
   # This proc checks that the value is either an single element collection
   # of nets, ports, or pins or the name of a single net, port, or pin.
   ##########################################################################

   proc check_obj_spec {obj_or_name id} {
      if {[is_collection $obj_or_name]} {
         if {[sizeof_collection $obj_or_name] ne 1} {
            display_message "Supplied collection for ${id} has \
               more than one element."
            return -code error
         } else {
            set obj_type [lindex [get_attribute_value_list $obj_or_name \
               -name object_type] 0]
            set object $obj_or_name
            if {$obj_type ne "port" && $obj_type ne "pin" && \
               $obj_type ne "net"} {
               display_message "Specified value for ${id} has \
                  type '$obj_type' instead of 'pin', 'port' or 'net'"
               return -code error
            }
         }
      } else {
         set net_object [get_nets  $obj_or_name -silent]
         if {[sizeof_collection $net_object] eq 0} {
            display_message "Specified value for ${id} is not an \
               existing 'pin', 'port' or 'net' name."
            return -code error
         } elseif {[sizeof_collection $net_object] ne 1} {
            display_message "Specified value for ${id} is more \
               than one bit wide"
            return -code error
         }
         set port_object [get_ports $obj_or_name -silent]
         set pin_object [get_pins  $obj_or_name -silent]
         if {[sizeof_collection $port_object] ne 0} {
            #If the name is also a port, pick the port object
            set object $port_object
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
         } elseif {[sizeof_collection $pin_object] ne 0} {
            #If the name is also a pin, pick the pin object
            set object $pin_object
         } else {
            #The net is not a port nor a pin
            set object $net_object
         }
      }
      return $object
   }
```

Related Topics

report_context

set_context

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_dft_cell_selection_mapping

Context: dft

Mode: all

Returns the library cell selection associated to the specified power domain.

Usage

get_dft_cell_selection_mapping -power_domain_name *power_domain_name*

Description

Returns the library cell selection associated to the specified power domain.

Arguments

- -power_domain_name *power_domain_name*

  A required switch and string pair that specifies the value of power_domain_name attribute for which the associated library cell selection is to be returned.

Related Topics

report_dft_cell_selection_mapping

set_dft_cell_selection_mapping

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_dft_info_dictionary

Context: dft, patterns

Mode: setup, analysis

Creates a Tcl dictionary containing the information about the DFT inserted in the RTL that must be considered during scan insertion.

Usage

get_dft_info_dictionary [[-example_usage_script]
[-process_modules_with_chains_not_active_by_default]]

Description

This command creates a Tcl dictionary containing the information about the DFT inserted in the RTL that must be considered during scan insertion. When you are using Tessent Scan for scan insertion, the information about the DFT logic you inserted into the RTL is automatically understood and used during scan insertion. If you are performing the scan insertion as part of synthesis, the dictionary created by this command can be used to import the information into the synthesis tool.

Note that the DFT info dictionary is automatically stored into the dft_inserted_designs directory when the extract_icl command is called with the -write_in_tsdb switch set or defaulting to on.

The format of the dictionary is illustrated in the syntax summary below. The version, dft_signals, modules_with_chains, modules_with_chains_not_active_by_default, non_scannable_instance_list and edt_instances keys always exist even if their values are empty.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set tessent_dft_info_dict {
  version 1
  dft_signals {
    dft_signal_name {
      connection_node_name node_name
      connection_node_type port|pin
      forced_value_in_pre_scan_drc 0|1
    }
  }
  modules_with_chains {
    module_name {
      pre_scan_drc_on_boundary_only 0|1
      module_type                   normal|memory|occ
      is_hard_module                0|1
      internal_scan_only            0|1
      allow_scan_out_retiming       0|1
      instance_list                 {inst_name ...}
      scan_en_ports|ltest_en_ports|set_reset_ports {
        port_name {
          active_polarity 0|1
        }
      }
      clock_ports {
        port_name {
          off_state 0|1
        }
      }
      clock_out_ports_or_pins {
        port_or_pin_name {}
      }
      scan_chains {
        scan_chain_name {
          length                    auto|int
          scan_in_port              port_name
          scan_in_clock_name        port_name
          scan_in_clock_inversion   0|1
          scan_out_port             port_name
          scan_out_clock_name       port_name
          scan_out_clock_inversion  0|1
        }
      }
    }
  }
  modules_with_chains_not_active_by_default {
    // structure identical to modules_with_chains wrapper
  }
  non_scannable_instance_list {inst_name ...}
  edt_instances {
    instance_name {
      edt_module_name module_name
      scan_chains {
        chain_name {
          scan_in_port  port_name
          scan_out_port port_name
        }
      }
    }
  }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

}

Arguments

- -example_usage_script

  An optional switch used to request an example usage script that you can use as a starting example to convert the dictionary into the specific commands of your synthesis tool. The places you are expected to enter tool-specific commands are identified with the ### comments.

- -process_modules_with_chains_not_active_by_default

  An optional switch that you can specify along with the -example_usage_script switch to generate an example usage script that includes processing the modules_with_chains_not_active_by_default key

Examples

The following example shows the content of an example usage script to be used with the DFT info dictionary created as part of the extract_icl command:

process_dft_specification
extract_icl

```
puts [get_dft_info_dictionary -example_usage_script \
     -process_modules_with_chains_not_active_by_default]
puts "Processing DFT signals"
set dft_signals_dict [dict get $tessent_dft_info_dict dft_signals]
puts "Setting up Static Dft Signals"
foreach dft_signal [dict keys $dft_signals_dict] {
  if {[dict exists $dft_signals_dict $dft_signal forced_value_in_pre_scan_drc]} {
    set connection_node_name [dict get $dft_signals_dict $dft_signal connection_node_name]
    set connection_node_type [dict get $dft_signals_dict $dft_signal connection_node_type]
    set forced_value_in_pre_scan_drc [dict get $dft_signals_dict $dft_signal \
       forced_value_in_pre_scan_drc]
    puts "--- Static Dft Signal $dft_signal ---"
    if {$connection_node_type eq "pin"} {
      ### Command to cut and force created pseudo port ###
    } else {
      ### Command to force port ###
    }
  }
}

puts "Setting Dynamic Dft Signals"
array set scan_signal_map {ltest_en -ten shift_capture_clock -tclk}
foreach dft_signal {scan_en ltest_en shift_capture_clock} {
  if {[dict exists $dft_signals_dict $dft_signal]} {
    set connection_node_name [dict get $dft_signals_dict $dft_signal connection_node_name]
    set connection_node_type [dict get $dft_signals_dict $dft_signal connection_node_type]
    ### Command to declare signal as the source of the DFT function goes here    ###
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
puts "Declare the Non-Scannable instances"
set non_scannable_instance_list \
 [dict get $tessent_dft_info_dict non_scannable_instance_list]
if {[llength $non_scannable_instance_list] > 0} {
  ### Command to declare the Non-Scannable instances  ###
}

puts "EDT Scan chain connections"
set chain_count 0
set edt_instances_dict [dict get $tessent_dft_info_dict edt_instances]
foreach edt_instance [dict keys $edt_instances_dict] {
  set scan_chains_dict [dict get $edt_instances_dict $edt_instance scan_chains]
  foreach scan_chain [dict keys $scan_chains_dict] {
    set scan_in_port [dict get $scan_chains_dict $scan_chain scan_in_port]
    set scan_out_port [dict get $scan_chains_dict $scan_chain scan_out_port]
    ### Command to declare scan scan chains SI/SO locations    ###
    incr chain_count
  }
}

puts "Declare sub-chains and extract OCC clocks"
set occ_clock_pin_list [list]
set modules_with_chains_dict [dict get $tessent_dft_info_dict modules_with_chains]
foreach module_with_chain [dict keys $modules_with_chains_dict] {
  set instance_list [dict get $modules_with_chains_dict $module_with_chain instance_list]
  set module_type [dict get $modules_with_chains_dict $module_with_chain module_type]
  if {$module_type eq "occ"} {
    set clock_out_ports_or_pins_dict [dict get $modules_with_chains_dict \
       $module_with_chain clock_out_ports_or_pins]
    foreach clock_out [dict keys $clock_out_ports_or_pins_dict] {
      foreach instance $instance_list {
        lappend occ_clock_list "${instance}/${clock_out}"
      }
    }
  }

   set scan_en_port_list [list]
  set scan_en_ports_dict [dict get $modules_with_chains_dict $module_with_chain \
                           scan_en_ports]
  foreach scan_en_port [dict keys $scan_en_ports_dict] {
    set active_polarity [dict get $scan_en_ports_dict $scan_en_port active_polarity]
    ### Command to declare the Scan enable ports    ###
  }

   set ltest_en_port_list [list]
  if {[dict exists $modules_with_chains_dict $module_with_chain ltest_en_ports]} {
    set ltest_en_ports_dict [dict get $modules_with_chains_dict $module_with_chain \
       ltest_en_ports]
    foreach ltest_en_port [dict keys $ltest_en_ports_dict] {
      set active_polarity [dict get $ltest_en_ports_dict $ltest_en_port active_polarity]
      ### Command to declare the ltest enable ports      ###
    }
  }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
    set set_reset_port_list [list]
    if {[dict exists $modules_with_chains_dict $module_with_chain set_reset_ports]} {
      set set_reset_ports_dict [dict get $modules_with_chains_dict $module_with_chain \
         set_reset_ports]
      foreach set_reset_port [dict keys $set_reset_ports_dict] {
        set active_polarity [dict get $set_reset_ports_dict $set_reset_port active_polarity]
        ### Command to declare the ltest enable ports      ###
      }
    }

    set clock_port_list [list]
    if {[dict exists $modules_with_chains_dict $module_with_chain clock_ports]} {
      set clock_ports_dict [dict get $modules_with_chains_dict $module_with_chain clock_ports]
      foreach clock_port [dict keys $clock_ports_dict] {
        set off_state [dict get $clock_ports_dict $clock_port off_state]
        ### Command to declare the input clock ports      ###
      }
    }

    set chain_cnt 0
    set scan_chains_dict [dict get $modules_with_chains_dict $module_with_chain scan_chains]
    foreach scan_chain [dict keys $scan_chains_dict] {
      if {![dict exists $scan_chains_dict $scan_chain scan_in_clock_name] || \
          ![dict exists $scan_chains_dict $scan_chain scan_out_clock_name]} {
          set msg "// Warning: Ignoring Chain '$scan_chain' in Module '$module_with_chain' "
          append msg "because its scan_in_clock or scan_out_clock ports are not define " \
                     "and and this tool is not able to extract it."
          puts $msg
        continue
      }
      set length [dict get $scan_chains_dict $scan_chain length]
      if {$length eq "auto"} {
         set msg "Warning: Using length of 8 for Chain $scan_chain of Module "
         append msg "$module_with_chain because it had a value of auto " \
                    "and this tool does not support sub chain tracing to extract it. " \
                    "It may affect chain balancing."
         puts $msg
         set length 8
      }
      set scan_in_port             [dict get $scan_chains_dict $scan_chain scan_in_port]
      set scan_in_clock_name       [dict get $scan_chains_dict $scan_chain scan_in_clock_name]
      set scan_in_clock_inversion  [dict get $scan_chains_dict $scan_chain \
                                    scan_in_clock_inversion]
      set scan_out_port            [dict get $scan_chains_dict $scan_chain scan_out_port]
     set scan_out_clock_name      [dict get $scan_chains_dict $scan_chain scan_out_clock_name]
      set scan_out_clock_inversion [dict get $scan_chains_dict $scan_chain \
                                    scan_out_clock_inversion]
        ### Command to declare the sub chain    ###
       incr chain_cnt
    }
  }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set modules_with_chains_dict [dict get $tessent_dft_info_dict \
                              modules_with_chains_not_active_by_default]
foreach module_with_chain [dict keys $modules_with_chains_dict] {
  set instance_list [dict get $modules_with_chains_dict $module_with_chain instance_list]
  set module_type [dict get $modules_with_chains_dict $module_with_chain module_type]
  if {$module_type eq "occ"} {
    set clock_out_ports_or_pins_dict [dict get $modules_with_chains_dict \
                                      $module_with_chain clock_out_ports_or_pins]
    foreach clock_out [dict keys $clock_out_ports_or_pins_dict] {
      foreach instance $instance_list {
        lappend occ_clock_list "${instance}/${clock_out}"
      }
    }
  }

   set scan_en_port_list [list]
  set scan_en_ports_dict [dict get $modules_with_chains_dict $module_with_chain \
                          scan_en_ports]
  foreach scan_en_port [dict keys $scan_en_ports_dict] {
    set active_polarity [dict get $scan_en_ports_dict $scan_en_port active_polarity]
    ### Command to declare the Scan enable ports    ###
  }

   set ltest_en_port_list [list]
  if {[dict exists $modules_with_chains_dict $module_with_chain ltest_en_ports]} {
    set ltest_en_ports_dict [dict get $modules_with_chains_dict $module_with_chain \
                             ltest_en_ports]
    foreach ltest_en_port [dict keys $ltest_en_ports_dict] {
      set active_polarity [dict get $ltest_en_ports_dict $ltest_en_port active_polarity]
      ### Command to declare the ltest enable ports      ###
    }
  }

   set set_reset_port_list [list]
  if {[dict exists $modules_with_chains_dict $module_with_chain set_reset_ports]} {
    set set_reset_ports_dict [dict get $modules_with_chains_dict $module_with_chain \
                              set_reset_ports]
    foreach set_reset_port [dict keys $set_reset_ports_dict] {
      set active_polarity [dict get $set_reset_ports_dict $set_reset_port active_polarity]
      ### Command to declare the ltest enable ports      ###
    }
  }

   set clock_port_list [list]
  if {[dict exists $modules_with_chains_dict $module_with_chain clock_ports]} {
    set clock_ports_dict [dict get $modules_with_chains_dict $module_with_chain clock_ports]
    foreach clock_port [dict keys $clock_ports_dict] {
      set off_state [dict get $clock_ports_dict $clock_port off_state]
      ### Command to declare the input clock ports      ###
    }
  }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
    set mode_en_port_list [list]
   if {[dict exists $modules_with_chains_dict $module_with_chain mode_en_ports]} {
     set mode_en_ports_dict [dict get $modules_with_chains_dict $module_with_chain \
                            mode_en_ports]
     foreach mode_en [dict keys $mode_en_ports_dict] {
       set active_polarity [dict get $mode_en_ports_dict $mode_en active_polarity]
       ### Command to declare the input mode enable ports      ###
     }
   }

  set chain_cnt 0
  set scan_chains_dict [dict get $modules_with_chains_dict $module_with_chain scan_chains]
  foreach scan_chain [dict keys $scan_chains_dict] {
     if {![dict exists $scan_chains_dict $scan_chain scan_in_clock_name] || \
         ![dict exists $scan_chains_dict $scan_chain scan_out_clock_name]} {
           set msg "// Warning: Ignoring Chain '$scan_chain' in Module '$module_with_chain' \
                    because "
           append msg "its scan_in_clock or scan_out_clock ports are not define and " \
                      "and this tool is not able to extract it."
     puts $msg
     }
     set length [dict get $scan_chains_dict $scan_chain length]
     if {$length eq "auto"} {
       set msg "Warning: Using length of 8 for Chain $scan_chain of Module \
                $module_with_chain "
       append msg "because it had a value of auto and this tool does not support " \
                  "sub chain tracing to extract it. It may affect chain balancing."
       puts $msg
       set length 8
     }
     set scan_in_port             [dict get $scan_chains_dict $scan_chain scan_in_port]
     set scan_in_clock_name       [dict get $scan_chains_dict $scan_chain scan_in_clock_name]
     set scan_in_clock_inversion  [dict get $scan_chains_dict $scan_chain \
                                   scan_in_clock_inversion]
     set scan_out_port            [dict get $scan_chains_dict $scan_chain scan_out_port]
     set scan_out_clock_name      [dict get $scan_chains_dict $scan_chain scan_out_clock_name]
     set scan_out_clock_inversion [dict get $scan_chains_dict $scan_chain \
                                     scan_out_clock_inversion]
       ### Command to declare the sub chain    ###
      incr chain_cnt
   }
 }

 puts "Declaring OCC output clocks"
 foreach occ_clock $occ_clock_list {
   ### Command to declare clocks with off state of 0  ###
 }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_dft_signal

Context: dft, patterns

Mode: setup, analysis, insertion

Obtains the source of a DFT signal or various aspects about the DFT signal.

Usage

get_dft_signal {**-list** | **-source_node** ***pin_port_spec***
| ***name*** [-at *at_value* | -exists | -list_of_parent_instances
| -scan_mode_type] [-parent_instance *parent_instance*]}
[-usage *usage*] [-silent]

Description

Obtains the source of a DFT signal or various aspect about the DFT signal. Between the time a DFT signal was added and process_dft_specification has completed its run, the existence of a DFT signal can be checked with the -exists switch, but the signal's location cannot be obtained until it is actually created.

You can query the location of a DFT signal within the process_dft_specification.post_insertion callback as the DFT signal is created before the proc is called. As you can see in the example below, the get_dft_signal command is useful at several stages of the flow to obtain the location of a previously added DFT signal.

Arguments

- **-list**

  A Boolean switch that specifies to return the list of existing or to-be-created DFT signals. The switch can be combined with the -usage switch to limit the list to a single usage as shown in the example.

  When the -list, -exists, -list_of_parent_instances, and -scan_mode_type switches are not specified, the command returns a collection containing the object on which the DFT signal exists.

- **-source_node** ***pin_port_spec***

  A switch-value pair used to specify the name of a single port or pin object or a collection containing a single port or pin object. When used, the command returns the name of the DFT signal added to the specified node. If no DFT signals were added on the specified node, an error is generated unless the -silent switch is used, in which case the error message is suppressed, and an empty string is returned.

- ***name***

  A string argument used to specify the name of a DFT signal for which you want its location or other aspect such as its existence, or the type of scan_mode it is.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -at *at_value*

  A switch-value pair that you use to access a DFT signal location at specified point in the DFT signal path. The DFT signal can have several instruments on its path, such as pads or LBIST, and SSH. When you do not use this switch, the command returns the location at the end of the DFT signal path.

  The *at_value* can be one of the following values:

  source

  Returns the pin or port at the start of the DFT signal path.

  source_pad_dft_driver

  Returns the location of the DFT signal at the output of boundary logic (pad, auxiliary pin, or pad followed by an auxiliary pin).

  lbist_dft_driver

  Returns the DFT signal location at the output of an LBIST instrument instance where the signal is available.

  ssh_dft_driver

  Returns the DFT signal location on the output of ScanHost instrument instance where the signal is available.

  current_dft_driver

  Returns the end location of the DFT signal path, just as when you omit the -at switch.

  After SSH insertion, current_dft_driver points to ssh_dft_driver.

- -exists

  A Boolean switch used to specify that a 1 should be returned if the specified DFT signal name exist and when the -usage switch is also specified that the DFT signal is of the specified usage. A “0” is returned otherwise.

- -list_of_parent_instances

  A Boolean switch that requests the list of parent instances in which the DFT signal exists. A null value in the list represents the top module. This option is only valid when combined with the name edt_clock and shift_capture_clock as those are the only two signals that may have several copies located in different parent instances. See the add_dft_signals -parent_instance switch description for more details.

- -scan_mode_type

  A Boolean switch used to request the type of DFT signal having the usage scan_mode. The returned values are unwrapped, internal, external, or retargeting when the DFT signal has the usage scan_mode. The return value is a null string when the DFT signal has a usage other than scan_mode.

- -parent_instance *parent_instance*

  A switch-value pair used to specify a parent instance in which the DFT signal is to be located. This option is only valid when combined with the name edt_clock and

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

shift_capture_clock as those are the only two signals that may have several copies located in different parent instances. See the add_dft_signals -parent_instance switch description for more details.

- -usage *usage*

  A switch-value pair that is used to limit the DFT signals to a specified usage. The five permitted usages are: global_dft_control, logic_test_control, scan_mode, dynamic, static_ijtag. The usage static_ijtag means the DFT signal was added with -type static_dft_control and with either the "-create_with_tdr" switch or with the "-source_node *port* -make_ijtag_port" switches, which makes it controllable by ijtag.

  When combined with the -list switch, it restricts the list to the DFT signal of the specified usage. When combined with the -exists switch, the command only returns "1" if the DFT signal exists and is of the specified usage.

- -silent

  A Boolean switch that suppresses the error message that is normally generated when the specified DFT signal does not exist or is not of the requested usage. Instead of generating an error, the tool returns an empty collection.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example illustrates the usage of the various switches:

```
set_context dft -rtl

read_verilog ../data/corea.v

set_current_design corea

set_design_level physical_block

add_dft_signals ltest_en int_ltest_en ext_ltest_en \
            memory_bypass_en async_set_reset_static_disable \
            int_mode ext_mode

add_dft_signals scan_en test_clock edt_update \
            -source_nodes {my_scan_en my_test_clock my_edt_update}

add_dft_signals edt_clock shift_capture_clock \
            async_set_reset_dynamic_disable \
            -create_from_other_signals

check_design_rules

create_dft_specification

process_dft_specification

foreach usage {global_dft_control logic_test_control \
             scan_mode dynamic} {
   foreach dft_signal [get_dft_signal -list -usage $usage] {
     puts "get_dft_signal $dft_signal -exists ="
     puts "[get_dft_signal $dft_signal -exists]"
     puts "get_dft_signal $dft_signal ="
     puts "[get_single_name [get_dft_signal $dft_signal]]"
     if {$usage eq "scan_mode"} {
      puts "get_dft_signal $dft_signal -scan_mode_type ="
      puts "[get_dft_signal $dft_signal -scan_mode_type]"
     }
   }
 }
get_dft_signal async_set_reset_static_disable -exists =

1

get_dft_signal async_set_reset_static_disable =

corea_rtl_tessent_tdr_sri_ctrl_inst/async_set_reset_static_disable

get_dft_signal ext_ltest_en -exists =

1

get_dft_signal ext_ltest_en =

corea_rtl_tessent_tdr_sri_ctrl_inst/ext_ltest_en
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_dft_signal int_ltest_en -exists =

```
1
```

get_dft_signal int_ltest_en =

```
corea_rtl_tessent_tdr_sri_ctrl_inst/int_ltest_en
```

get_dft_signal ltest_en -exists =

```
1
```

get_dft_signal ltest_en =

```
corea_rtl_tessent_tdr_sri_ctrl_inst/ltest_en
```

get_dft_signal memory_bypass_en -exists =

```
1
```

get_dft_signal memory_bypass_en =

```
corea_rtl_tessent_tdr_sri_ctrl_inst/memory_bypass_en
```

get_dft_signal ext_mode -exists =

```
1
```

get_dft_signal ext_mode =

```
corea_rtl_tessent_tdr_sri_ctrl_inst/ext_mode
```

get_dft_signal ext_mode -scan_mode_type =

```
external
```

get_dft_signal int_mode -exists =

```
1
```

get_dft_signal int_mode =

```
corea_rtl_tessent_tdr_sri_ctrl_inst/int_mode
```

get_dft_signal int_mode -scan_mode_type =

```
internal
```

get_dft_signal async_set_reset_dynamic_disable -exists =

```
1
```

get_dft_signal async_set_reset_dynamic_disable =

```
tessent_persistent_cell_async_set_reset_dynamic_disable/y
```

get_dft_signal edt_clock -exists =

```
1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_dft_signal edt_clock =

```
tessent_persistent_cell_edt_clock/clkg
```

get_dft_signal edt_update -exists =

```
1
```

get_dft_signal edt_update =

```
my_edt_update
```

get_dft_signal input_wrapper_scan_en -exists =

```
1
```

get_dft_signal input_wrapper_scan_en =

```
tessent_persistent_cell_input_wrapper_scan_en/y
```

get_dft_signal output_wrapper_scan_en -exists =

```
1
```

get_dft_signal output_wrapper_scan_en =

```
tessent_persistent_cell_output_wrapper_scan_en/y
```

get_dft_signal scan_en -exists =

```
1
```

get_dft_signal scan_en =

```
my_scan_en
```

get_dft_signal shift_capture_clock -exists =

```
1
```

get_dft_signal shift_capture_clock =

```
tessent_persistent_cell_shift_capture_clock/clkg
```

get_dft_signal test_clock -exists =

```
1
```

get_dft_signal test_clock =

```
my_test_clock
```

Error message when the requested signal does not exist:

catch {get_dft_signal junk}

```
//  Error: The DFT signal named 'junk' does not exist. Run the
// 'report_dft_signals' command for more information.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Error message when the requested signal is of the wrong usage:

catch {get_dft_signal ltest_en -usage scan_mode}

```
//  Error: The DFT signal named 'ltest_en' is for usage
//  'logic_test_control' instead of 'scan_mode'.
```

Error message suppressed when -silent is used:

get_dft_signal ltest_en -usage scan_mode -silent

The -unmapped_across_pad parameter usage and comparison:

add_dft_signals scan_en -source_node gpio3

get_dft_signals scan_en

```
{gpio3_pad/o}
```

get_dft_signal scan_en -unmapped_across_pad

```
{gpio3}
```

Related Topics

add_dft_signals

process_dft_specification

report_dft_signals

set_static_dft_signal_values

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_dft_signals_option

Context: dft -rtl, dft -no_rtl

Mode: setup, analysis, insertion

Returns the current setting specified by the set_dft_signals_options command.

Usage

get_dft_signals_option **-disable_for_functional_safety**

Description

Returns the current setting specified by the corresponding option and the set_dft_signals_options command.

Arguments

- **-disable_for_functional_safety**

  A required switch that returns the current setting for this option. See "set_dft_signals_options" on page 2464 for possible return values.

Examples

The following example indicates that gating and monitoring logic for single-event upsets (SEUs) in test data registers (TDRs) is not created. The TDR registers fan out directly to functional logic.

```
SETUP> get_dft_signals_option -disable_for_functional_safety
off
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_dft_specification_requirement

Context: dft patterns

Mode: all modes

Returns the requirements set using the set_dft_specification_requirements command.

Usage

get_dft_specification_requirement {-memory_bist | -tck_injection | -memory_bisr_chains | -memory_bisr_controller | -boundary_scan | -ac_boundary_scan | -bsdl_extraction | -bonding_configuration | -logic_test | -add_test_clock_on_ssh | -memory_bisr_host_list | -bisr_segment_order_file | -host_scan_interface_type | -design_type}

Description

Returns the requirements set using the set_dft_specification_requirements command.

Arguments

- -memory_bist

  An option that requests the value of the set_dft_specification_requirements -memory_bist option.

- -tck_injection

  An option that requests the value of the set_dft_specification_requirements -tck_injection option.

- -memory_bisr_chains

  An option that requests the value of the set_dft_specification_requirements -memory_bisr_chains option.

- -memory_bisr_controller

  An option that requests the value of the set_dft_specification_requirements -memory_bisr_controller option.

- -boundary_scan

  An option that requests the value of the set_dft_specification_requirements -boundary_scan option.

- -ac_boundary_scan

  An option that requests the value of the set_dft_specification_requirements -ac_boundary_scan option.

- -bsdl_extraction

  An option that requests the value of the set_dft_specification_requirements -bsdl_extraction option.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -bonding_configuration

  An option that requests the value of the set_dft_specification_requirements -bonding_configuration option.

- -logic_test

  An option that requests the value of the set_dft_specification_requirements -logic_test option.

- -add_test_clock_on_ssh

  An option that requests the value of the set_dft_specification_requirements -add_test_clock_on_ssh option.

- -memory_bisr_host_list

  An option that requests the value of the set_dft_specification_requirements -memory_bisr_host_list option.

- -bisr_segment_order_file

  An option that requests the value of the set_dft_specification_requirements -bisr_segment_order_file option.

- -host_scan_interface_type

  An option that requests the value of the set_dft_specification_requirements -host_scan_interface_type option.

- -design_type

  An option that requests the value of the set_dft_specification_requirements -design_type option.

Examples

The following example queries the value of the -memory_test option.

get_dft_specification_requirement -memory_bist

```
off
```

set_dft_specification_requirement -memory_test on

get_dft_specification_requirement -memory_bist

```
auto
```

Related Topics

set_dft_specification_requirements

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_drc_handling

Context: unspecified, all contexts

Mode: all modes

Retrieves the DRC handling status for the DRC with the specified name.

Usage

get_drc_handling ***drc_name*** [-auto_fix]

Arguments

- ***drc_name***

  A required string that specifies the name of the of the DRC you for which you want the DRC handling status.

- -auto_fix

  An optional switch that enables you to see if the DRC supports auto fixing of the violation. If you registered your own custom DRC using the register_drc command and you used the -allow_auto_fix switch, this switch returns 1 unless you explicitly disabled auto fixing using the "set_drc_handling -auto_fix" command. It always returns 0 for a DRC that does not permit auto fixing. Use this command in the body of your custom DRC to know if the violation should be reported or auto fixed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_equivalent_editable_node

Context: dft

Mode: setup, analysis, insertion

Checks if a node is editable. If the node is not editable, then returns the node's editable equivalent node if it exists.

Usage

get_equivalent_editable_node ***node_spec***

Description

The get_equivalent_editable_node command accepts a pin, port, net, pseudo_port or gate_pin objects as its argument.

If the object is a pseudo_port or a gate_pin, the tool maps it to its corresponding port, pin, or net object. If the object is a pin that is on or inside an instance created by RTLC, or inside a hard module or a cell or if the object is a net that is inside an instance created by RTLC, or inside a hard module or a cell, the tool maps it to an equivalent node in the editable region if one exists.

If the object cannot be mapped to an editable object, an empty collection is returned.Use this command when you are using the synthesized view to perform some analysis and then you want to perform edits on the RTL view.

Arguments

- ***node_spec***

  A required string used to specify the name of a pin, port, net, pseudo_port or gate_pin object or a collection containing a single pin, port, net, pseudo_port or gate_pin object.

Examples

The following example shows the use of the command to map various nodes to its editable equivalent:

Pin on a synthesized instance:

```
set map [get_equivalent_editable_node ex2_cell/rtlc0_1/sel_0]
set type [get_attribute_value_list $map -name object_type]
puts "$type [get_single_name $map]"
```

```
pin ex2_cell/c
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Pin on a normal instance:

```
set map [get_equivalent_editable_node ex2_cell/a]
set type [get_attribute_value_list $map -name object_type]
puts "$type [get_single_name $map]"
```

```
pin ex2_cell/a
```

Pseudo_ports are mapped to their associated design objects:

```
puts [get_single_name [get_equivalent_editable_node pseudo_a]]
```

```
ex1_i1/a
```

A gate_pin is mapped to its pseudo port then to the pin:

```
puts [get_single_name [get_equivalent_editable_node
```

```
ex1_i1/y
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_fanins

Context: all contexts

Mode: all modes

Prerequisites: The current design must be set.

Returns a collection of all requested objects found in the fan-in of the specified pin, net, or port object.

Usage

get_fanins ***obj_spec***
[-stop_on {cell_pin | net | pin}]
[-direction {inward | outward}]
[-silent]

Description

Returns a collection of all requested objects found in the fan-in of the specified pin, net, or port object.

When <*obj_spec*> is an output or inout pin of a cell, the command reports all input or inout pins on the same cell that are in the combinational fan-in of the output or inout pin.

Note

When you specify the "-stop_on pin" option, the fan-in direction of an inout pin is toward the inside of the instance.

The -stop_on option specifies object types upon which the tracing should stop. It does not mean that the returned collections contain only objects of the specified type because the tracing may stop before reaching the specified type. For example, a pin may be connected to a dangling net. The net is reported as the object on the fan-in of the pin even if -stop_on is specified as cell_pin. If the net had not been dangling and it had a driver, the get_fanins command would not have stopped on the net and would have continued to walk the fan-in path.

The get_fanins command returns an empty collection when there is no fan-in. The fan-in of an input port is null. When the fan-in of an input pin is an empty collection, the pin is either unconnected or tied. Use the tie_value attribute to see it if is tied to 0 or 1 or unconnected.

The get_fanins command does not use simulated values and only considers connectivity of the hierarchical data model. You can use the trace_flat_model command to perform tracing on the flat model with simulation conditions considered.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***obj_spec***

  A required value that specifies a Tcl list of one or more net, pin, port, or pseudo_port names; or a collection of one or more net, pin, port, or pseudo_port objects.

  Tip

  If your ***object_spec*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- -stop_on {cell_pin | net | pin}

  An optional switch and literal pair that specifies the following:

  - When -stop_on is set to cell_pin (the default), the fan-in walk traverses nets and design pins but stops on pins of cells or primitives in the fan-in of the object. The walk may still, however, stop on a net or a design pin when the net or design pin does not have a fan-in.
  - When -stop_on is set to net, the fan-in walk traverses through one level of "assign" statements and stops on the net connected to the start pin.
  - When -stop_on is set to pin, the fan-in walk traverses nets and reports the pin in the fan-in of the object. This could be an input or inout pin above the object or an output or inout pin on child instances. The walk may still, however, stop on a net if the net does not fan-in to pins.

- -direction {inward | outward}

  An optional switch and literal pair to specify the trace direction for inout pins. This switch is only relevant to inout pin objects, not ports or nets, and must be used in conjunction with the "-stop_on pin" option. The default is inward.

- -silent

  An optional switch that suppresses the error that is normally generated if any object within obj_spec does not exist, in which case it is ignored.

Examples

Example 1

The following example shows a series of get_fanins command invocations of the circuit example shown in Figure 5-3.

```
# find net connected to pin u4/I5
get_fanins u4/I5 -stop_on net

{n3}

#Find cell_pin in the fan-in of u4/I5
get_fanins u4/I5
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{u3/Y}
```

#Find pin in the fan-in of u4/I5. Returns cell pins as there is no design pins before it
get_fanins u4/I5 -stop_on pin

```
{u3/Y}
```

#Find pin in the fan-in of u4/u1/A0
get_fanins u4/u1/A0 -stop_on pin

```
{u4/I1}
```

#Find all nets connected to the D pin of dff modules
get_fanins [get_pins -filter {module_name == “dff” && leaf_name == “D”}] -stop_on net

```
{I3 n6 u4/n2 u4/n3 u4/u5/n1 u4/u5/I3 u5/n1}
```

#Find pins in the fan-in of an output pin of a cell
get_fanins u1/Y

```
{u1/A0 u1/A1 u1/A2}
```

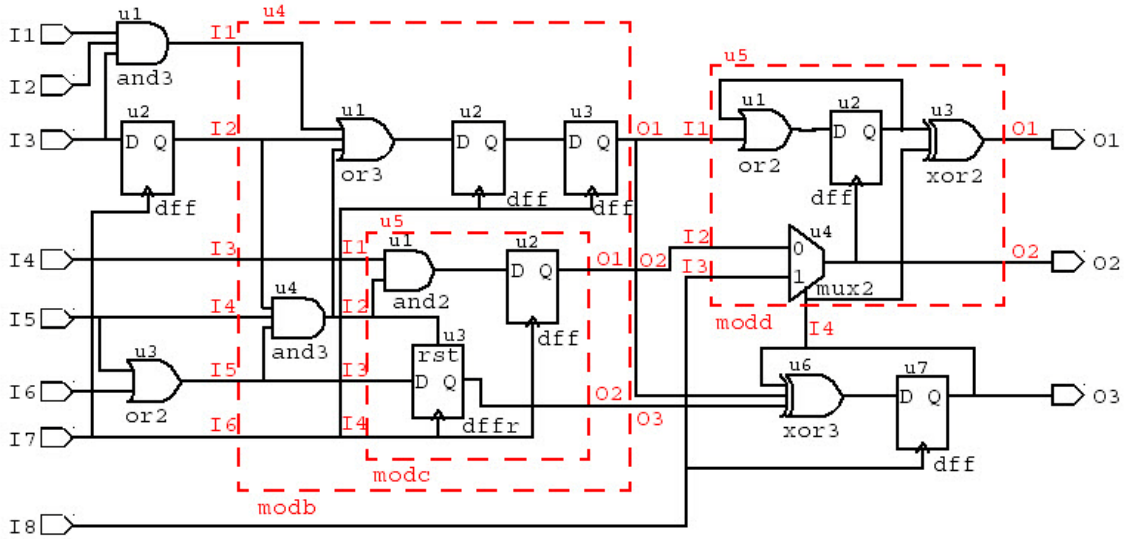
Example 2

The following example adds an internal primary input (pseudo_port). Based on the example circuit shown in Figure 5-3, it then returns the collection of objects found in the fan-in of both of the pseudo_port’s underlying pins.

add_primary_inputs {u4/I4 u4/I5} -internal -pseudo_port_name user_pi1
get_fanins user_pi1

```
{I5 u3/Y}
```

Figure 5-3. Circuit Illustrating the Usage of the get_fanins Command



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

The following example is based on a pad cell that is inside a wrapper module:

```
module top( inout wire gpio1 );
   wrapper wrap_inst(.gpio1(gpio1));
endmodule

module wrapper( inout wire gpio1 );
   iopad gpio1_pad(.io(gpio1));
endmodule

SETUP> get_fanins wrap_inst/gpio1 -stop_on pin
 {wrap_inst/gpio1_pad/io}

SETUP> get_fanins wrap_inst/gpio1 -stop_on pin -direction inward
 {wrap_inst/gpio1_pad/io}

SETUP> get_fanins wrap_inst/gpio1 -stop_on pin -direction outward
 {gpio1}
```

Related Topics

get_icl_ports

get_nets

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_fanouts

Context: all contexts

Mode: all modes

Prerequisites: The current design must be set.

Returns a collection of all requested objects found in the fanout of the specified pin, net, or port object.

Usage

get_fanouts ***obj_spec***
[-stop_on {cell_pin | net | pin}]
[-direction {inward | outward}]
[-silent]

Description

Returns a collection of all requested objects found in the fanout of the specified pin, net, or port object.

When <*obj_spec*> is an input pin of a cell, the command reports all output or inout pins on the same cell that are in the combinational fanout of the input pin.

Note

When you specify the "-stop_on pin" option, the fanout direction of an inout pin is toward the outside of the instance.

The -stop_on option is used to specify object types upon which the tracing should stop. Specifying this option does not mean that the returned collections only contain objects of the specified type, because the tracing may stop before reaching the specified type. For example, a pin may be connected to a dangling net. The net is reported as the object on the fanout of the pin even if -stop_on is specified as cell_pin. If the net had been connected to other pins, the get_fanouts command would not have stopped on the net and would have continued to walk the fanout path.

The get_fanouts command returns an empty collection when there is no fanout. The fanout of an output port is null.

The get_fanouts command does not use simulated values and only considers connectivity of the hierarchical data model. You can use the trace_flat_model command to perform tracing on the flat model with simulation conditions considered.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***obj_spec***

  A required value that specifies a Tcl list of one or more net, pin, port, or pseudo_port names; or a collection of one or more net, pin, port, or pseudo_port objects.

  Tip

  If your ***object_spec*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- -stop_on cell_pin | net | pin

  An optional switch and literal pair that specifies the following:

  - When -stop_on is set to cell_pin (the default), the fanout walk traverses nets and design pins but stops on pins of cells or primitives in the fanout of the object. The walk may still, however, stop on a net or a design pin when the net or design pin does not have a fanout.
  - When -stop_on is set to net, the fanout walk traverses through one level of "assign" statements and stops on the net connected to the start pin,.
  - When -stop_on is set to pin, the fanout walk traverses nets and reports the pin in the fanout of the object. This could be output or inout pins above the object or inputs or inout pins on child instances. The walk may still, however, stop on a net if the net does not fanout to pins.

- -direction {inward | outward}

  An optional switch and literal pair to specify the trace direction for inout pins. This switch is only relevant to inout pin objects, not ports or nets, and must be used in conjunction with the "-stop_on pin" option. The default is outward.

- -silent

  An optional switch that suppresses the error that is normally generated if any object within obj_spec does not exist, in which case it is ignored.

Examples

Example 1

The following example shows a series of get_fanouts command invocations of the circuit example shown in Figure 5-4.

#Find cell pins in the fanout of u4/I5
get_fanouts u4/I5

```
{u4/u4/A2 u4/u5/u3/D}
```

#Find pins in the fanout of u4/u1/Y. Returns a cell pin as there is no design pin before it
get_fanouts u4/u1/Y -stop_on pin

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{u4/u2/D}
```

#Find pins in the fanout of u4/u4/Y
get_fanouts u4/u4/Y -stop_on pin

```
{u4/u1/A2 u4/u5/I2}
```

#Find all nets connected to the Q pin of dff modules
get_fanouts [get_pins -filter {module_name == “dff”&& leaf_name == “Q”}] -stop_on net

```
{n2 O3 u4/n3 u4/O1 u4/u5/O1 u4/u5/O2 u5/n2}
```

#Find net in fanout of a cell output pin
get_fanouts u1/Y -stop_on net

```
{n1}
```

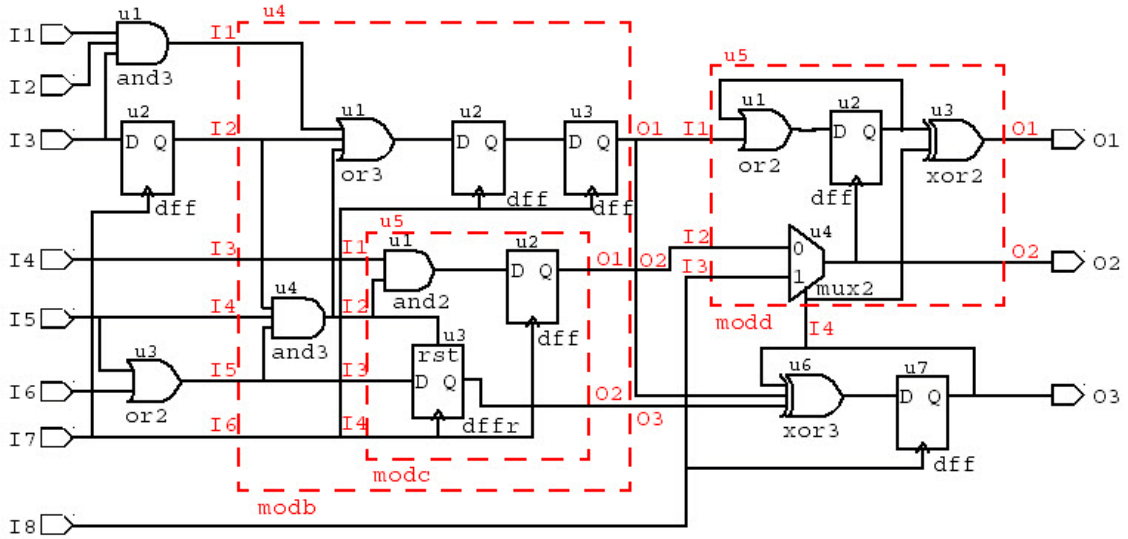
Example 2

The following example adds an internal primary output (pseudo_port). Based on the example circuit shown in Figure 5-4, it then returns the collection of objects found in the fanout of the pseudo_port’s underlying pin.

add_primary_outputs {u4/I5} -internal
get_fanouts [get_ports u4/I5 -of_type pseudo]

```
{u4/u4/A2 u4/u5/u3/D}
```

Figure 5-4. Circuit Illustrating the Usage of the get_fanouts Command



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

The following example is based on a pad cell that is inside a wrapper module:

```
module top( inout wire gpio1 );
   wrapper wrap_inst(.gpio1(gpio1));
endmodule

module wrapper( inout wire gpio1 );
   iopad gpio1_pad(.io(gpio1));
endmodule

SETUP> get_fanouts wrap_inst/gpio1 -stop_on pin
 {gpio1}

SETUP> get_fanouts wrap_inst/gpio1 -stop_on pin -direction outward
 {gpio1}

SETUP> get_fanouts wrap_inst/gpio1 -stop_on pin -direction inward
 {wrap_inst/gpio1_pad/io}
```

Related Topics

get_fanins

get_icl_ports

get_nets

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_fault_type

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: all modes

Returns the current fault model type, as previously specified by the set_fault_type command.

Usage

get_fault_type

Description

Returns the current fault model type, as previously specified by the set_fault_type command.

The command returns one of the following strings: stuck, toggle, iddq, transition, path_delay, bridge, udfm.

Arguments

None

Examples

Example 1

The following example shows how to set the fault model type and then introspect the fault model:

```
ANALYSIS> set_fault_type bridge
ANALYSIS> get_fault_type
```

```
bridge
```

Example 2

The following example loads an SDC file if the current fault model type is “transition”:

```
if {[get_fault_type] eq "transition" } {
   read_sdc myDesign.sdc
} else {
   puts "No SDC file read. Fault type is [get_fault_type]."
}
```

Related Topics

set_fault_type

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_flat_model_option

Context: unspecified, all contexts

Mode: all

Retrieves the current setting of the specified flat model option.

Usage

get_flat_model_option {**-remove_unused_library_pins** |
-eligible_cell_port_names_threshold_for_pin_removal |
-gate_count_reduction_threshold_for_pin_removal |
-emulate_loadboard_loopback_pairs}

Description

The get_flat_model_option command returns information about a specified flat model option previously configured with the set_flat_model_options command. Use an argument whenever you run this command.

Arguments

- **-remove_unused_library_pins**

  A required switch that specifies to return the setting that controls which read_cell_library models have unused ports removed from the flat model file.

- **-eligible_cell_port_names_threshold_for_pin_removal**

  A required switch that specifies the flat model option to retrieve the threshold at which the tool removes unused ports from the flat model.

- **-gate_count_reduction_threshold_for_pin_removal**

  A required switch that specifies to retrieve the threshold of the flat model gate reduction percentage that must occur before the tool removes any unused ports.

- **-emulate_loadboard_loopback_pairs**

  A required switch that specifies to retrieve the status of external loadboard loopback emulation of the flat model.

Examples

The following example returns the current setting of the "set_flat_model_options -remove_unused_library_pins" command.

```
get_flat_model_option -remove_unused_library_pins
```

Related Topics

set_flat_model_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_gate_pins

Context: all contexts

Mode: setup, analysis

Prerequisites: Although you can use this command in all system modes, you can only use it with a flattened netlist. In Tessent Shell, netlist flattening typically happens when you first attempt to enter analysis mode or when you run the create_flat_model command.

Returns a collection of gate_pin objects found below the current design in the flat model.

Usage

get_gate_pins [*gate_pin_id_list* | *hier_pin_name_patterns*]
[-filter *attribute_equation*]
[-regexp]
[-nocase]
[-silent]
[-of_instances *instance_objects*
| -below_instances *instance_objects*
| -of_pins *pin_objects*
| -of_ports *port_objects*
| -of_nets *net_objects*
| -of_gates {*gate_id_list* | *gate_pin_objects* | -from_objects *from_objects*}]
[-of_gate_type {*sequential* | *primitive_name_list*}]
[-direction {source | destination}]
[-output_strength {strong | weak}]

Description

Returns a collection of gate_pin objects found below the current design in the flat model.

The command supports arbitrary filtering based on attribute values. It also supports those options for frequently used filters.

Arguments

- *gate_pin_id_list* | *hier_pin_name_patterns*

  An optional string that specifies a list of gate_pin objects. The *gate_pin_id_list* is a list of gate_pin IDs (two integers separated by a period, the first integer representing a gate and the second representing a gate_pin). No wildcards are permitted in the *gate_pin_id_list*. If the second integer is not present, the ID specifies all pins on the gate.

  The *hier_pin_name_patterns* is a list of strings (wildcard patterns or regular expressions) matching hierarchical pin names. Using the *hier_pin_name_patterns* is equivalent to filtering on the hier_pin_name attribute.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

When the get_gate_pins command operates on the flat model, it removes all backslashes and white spaces from escaped identifiers. You must not include backslashes or white spaces in the *hier_pin_name_patterns* string. If you want to access the corresponding gate_pin, use the gate_pin_id attribute of the hierarchical pin object. If you want to access the gate_pin by name, use the get_name_list -remove_escaping command and switch.

- -filter *attribute_equation*

  An optional switch and string that specify to filter results based on the expression you specify with the *attribute_equation* string. The attributes used within the equation must exist for the gate_pin object type. See "Attribute Filtering Equation Syntax" on page 3832 for more details. The =~ and !~ matching operators use simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option or not.

- -regexp

  An optional switch that directs the tool to interpret the value of the *hier_pin_name_patterns* argument as a regular expression instead of as a simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for gate_pins matching the value of the *hier_pin_name_patterns* argument.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *hier_pin_name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

  Note

  You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

- -of_instances *instance_objects*

  An optional switch and value pair that constrains the command to return only gate_pins associated with the specified *instance_objects*.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to return only gate_pins associated with the circuitry located underneath the specified *instance_objects*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the specified *instance_spec* is an empty collection, all instances below the current design are considered. See the description of the -below_instances switch of the get_instances command description for the justification of this behavior.

- -of_pins *pin_objects*

  An optional switch and value pair that constrains the command to return only gate_pins associated with the specified *pin_objects*.

  If the pin is an inout pin, you can get up to three gate_pins if there is also a weak driver, such as a pull resistor on a tri-state inout buffer cell. At a minimum, you will get two gate pins for each inout pin. If you get the gate_pin to pass the trace_flat_model command, you must use the "get_gate_pins -of_pins -direction source | destination" command and the -output_strength option to get the gate pin you want to trace from.

  - If you want to trace the outside fanout or the inside fan-in of the inout pin, use "-direction source".
  - If you want to trace the outside fan-in or the inside fanout of the inout pin, use "-direction destination".

  Note

  Specifying the -output_strength option as strong or weak makes no difference when specifying "-direction destination" or when the pin has direction input.

- -of_ports *port_objects*

  An optional switch and value pair that constrains the command to return only gate_pins associated with the specified *port_objects* or *pseudo_port* objects.

- -of_nets *net_objects*

  An optional switch and value pair that constrains the command to return only gate_pins associated with the specified *net_objects*. If a gate_pin name directly matches the net name, the matching gate_pin is returned; otherwise, the driving gate_pin object of the net is returned.

- -of_gates *{gate_id_list | gate_pin_objects}*

  An optional switch and value pair that constrains the command to return only gate_pins associated with the specified gates. The specified gates can be a list of gate IDs, gate_pin IDs, gate_pin objects or hierarchical pin name patterns.

- -from_objects *from_objects*

  An optional switch and value pair that constrains the command to return only objects specified with the *from_objects* parameter, which may contain ports, pseudo_ports, pins, nets, and gate_pins. A port, pseudo_port, pin or net is mapped to the corresponding gate_pins in the same way the existing -of_ports, -of_pins or -of_nets options do. A gate_pin is just passed through to the result collection. This new option is convenient if you have a collection of design objects of different types that you want to map to gate_pins.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -of_gate_type {*sequential* | *primitive_name_list*}

  An optional switch and value pair that constrains the command to return only gate_pins associated with the gates of the specified types.

  When the gate type *sequential* is specified, the command returns all the gate_pin objects associated with the gates whose primitive_name attribute value is DFF, DLAT, RAM, or ROM.The permitted values for the *primitive_name_list* are all the primitive_name attribute values (see Table 11-6, Built-In Primitives and Primitive Port Names). The value of *primitive_name_list* can be either a single string or a Tcl list of strings such as {PI PO}.

- -direction {source | destination}

  An optional switch and literal pair that you can use to specify the direction of the gate_pin objects. This switch is useful when introspecting gate pins belonging to inout pins and ports to get the correct gate pin for tracing with the trace_flat_model command.

  source — A source is a gate_pin that can drive a value. For a port on the current design, the source is the gate_pin corresponding to an input port or the input branch of an inout port. For an intermediate pin, the source pin corresponds to an output pin or the output branch of an inout pin.

  destination — A destination is a gate_pin that gets a driven value. The destination is the gate_pin corresponding to an output port or the output branch of an inout port. For an intermediate pin, the destination gate_pin corresponds to an input pin or the input branch of an inout pin. This is the default.

- -output_strength {strong | weak}

  An optional switch and literal pair that instructs the tool to only include output gate_pins of the specified drive strength. The switch has no impact on input gate_pins. By default, the tool returns both strong and weak output drivers when they exist. See the -of_pins switch description for when to use this switch.

Examples

Example 1

The following example filters the collection of all gate_pin objects based on an attribute_equation of user-defined attributes:

get_gate_pins -filter {myType1 || myType2 == "ABC"}

```
{/u1/U2/b 16.0 16.1 16.2 24.0}
```

Example 2

The following example returns the collection of all gate_pin objects matching a list of case-insensitive hierarchical pin name patterns. For more examples of pattern matching with regular expressions, refer to "Glob and Regular Expression Pattern Matching Syntax" on page 3834.

get_gate_pins {u1/u1/a u*/b} -nocase

```
{/u1/U1/a /u1/U1/b /u1/U2/b /u1/U3/b /u2/U1/b}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

The following example returns a collection of all gate_pin objects on the gate with ID=12, the output gate_pin object (pin_index=0) on the gate with ID=14, and the gate_pin object with pin_index=2 on the gate with ID=11:

get_gate_pins { 12 14.0 11.2 }

```
{/u2/U1/b 12.0 14.0 /u1/U3/c}
```

Example 4

The following equivalent get_gate_pins commands return a collection of all gate_pin objects where the corresponding hierarchical pin name matches the wildcard pattern /u1/*/a or /u2/U1*:

get_gate_pins { /u1/*/a /u2/U1* }

```
{/u1/U1/a /u1/U2/a /u1/U3/a /u2/U1/b /u2/U1/c /u2/U1/a /u2/U1/y /u2/U1/z}
```

get_gate_pins -filter "hier_pin_name =~ /u1/*/a || hier_pin_name =~ /u2/U1*"

```
{/u1/U1/a /u1/U2/a /u1/U3/a /u2/U1/b /u2/U1/c /u2/U1/a /u2/U1/y /u2/U1/z}
```

Example 5

The following example adds to the fault list all of the gate_pin objects corresponding to the hierarchical pins matching the wildcard pattern u1/*:

add_faults [get_gate_pins u1/*]
report_faults

```
type    code    pin_pathname
----   ------   ------------------------------------
  0     UC      /u2/U1/b
  1     UC      /u2/U1/b
  0     UC      /u2/U1/c
  1     UC      /u2/U1/c
  0     UC      /u2/U1/a
  1     UC      /u2/U1/a
  0     UC      /u2/U1/y
  1     UC      /u2/U1/y
  0     UC      /u2/U1/z
  1     UC      /u2/U1/z
```

Example 6

The following examples are based on the circuit in Figure 5-3 on page 1012. This example returns all of the gate pins for the specified instance u4/u2:

SETUP> get_gate_pins -of_instances u4/u2

```
{u4/u2/Q u4/u2/QB u4/u2/CLK u4/u2/D}
```

This example returns all of the gate pins named D that are found below the u4 level of hierarchy:

SETUP> get_gate_pins */D -below_instances u4

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{u4/u2/D u4/u3/D u4/u5/u2/D u4/u5/u3/D}
```

Example 7

The following examples return all the gate pins in gates of type DFF:

get_gate_pins -filter “primitive_name == DFF”

```
{7.0 7.1 7.2 DFF/CLK DFF/D}
```

get_gate_pins -of_gate_type DFF

```
{7.0 7.1 7.2 DFF/CLK DFF/D}
```

This example returns gate pins named D in all gates of type DFF:

get_gate_pins -of_gate_type DFF -filter "primitive_port_name == D"

```
{DFF/D}
```

Related Topics

get_common_parent_instance

get_fanins

get_modules

get_nets

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_extraction_options

Context: all contexts

Mode: all modes

Provides access to the settings specified by the set_icl_extraction_options command.

Usage

get_icl_extraction_options
    -extract_learned_muxes | -extract_library_muxes | -extract_scan_interfaces

Description

Provides access to the settings specified by the set_icl_extraction_options command.

Arguments

- -extract_learned_muxes

  A switch that instructs the tool to return the extract_learned_muxes setting of the ICL extraction options. Possible values are 0 and 1.

- -extract_library_muxes

  A switch that instructs the tool to return the extract_library_muxes setting of the ICL extraction options. Possible values are 0 and 1.

- -extract_scan_interfaces

  A switch that instructs the tool to return the extract_scan_interfaces setting of the ICL extraction options. Possible values are 0 and 1.

Examples

The following command returns the setting specified by the -extract_library_muxes option of set_icl_extraction_options command:

get_icl_extraction_options -extract_library_muxes

```
1
```

Related Topics

report_icl_extraction_options

set_icl_extraction_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_fanins

Context: all contexts

Mode: all modes

Prerequisites: The current design must be set.

Returns a collection of ICL objects found in the fan-in of the specified ICL objects.

Usage

get_icl_fanins ***object_spec*** [{-stop_on none | pin} | -scan_path] [-silent]

Description

Returns a collection of ICL objects found in the fan-in of the specified ICL objects.

The types of the objects you can define in the ***object_spec*** and the types of the objects in the result collection depend on the usage of the -scan_path switch. If you omit this switch, the only possible object types in the object_spec and in the result collection are icl_port and icl_pin. If you use the -scan_path switch, the possible object types in the object_spec and in the result collection are icl_port, icl_pin, icl_scan_mux, icl_scan_register, and icl_one_hot_scan_group. Use the -stop_on option to specify if the command should look for the furthest objects in the fan-in or the closest. Use “none” in the former case and “pin” in the latter case. The -stop_on and -scan_path switches are mutually exclusive.

Arguments

- ***object_spec***

  A required value that specifies a single name, a list, or a collection of ICL objects, the fan-in of which will be analyzed. The types of the objects in the object_spec depend on the usage of the -scan_path switch, as discussed in the description earlier in this topic.

- -stop_on none | pin

  An optional switch and literal pair that specifies the following:

  none — The fan-in walk traverses the connections and the ICL pins in the fan-in of the object. The trace reports icl_pin objects with no fan-in, icl_pin objects without any icl_pin or icl_port objects in their fan-in, or icl_port objects reached in the fan-in walk. This is the default.

  pin — The fan-in walk traverses the connection and reports the icl_pin and icl_port objects in the fan-in of the object; this could be an ICL input pin above the object or an ICL output pin on a child ICL instance. The walk may still, however, stop on an ICL port if the ICL pin does not fan-in to ICL pins.

- -scan_path

  An optional switch that limits the scope of the tracing to scan paths but also allows the usage of all object types of elements that can be encountered on a scan path: icl_port, icl_pin, icl_scan_mux, icl_scan_register, and icl_one_hot_scan_group. When you use this switch, the object_spec and the returned collection can contain objects of these object types. The

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

tracing always stops immediately at the next encountered element. Tracing backwards from a ScanMux considers the scan data inputs only, but not the select inputs. Tracing backwards from a ScanOutPort considers the Source specification only, but not the Enable specification. Tracing backwards from a ScanRegister considers the ScanInSource specification only, but not the CaptureSource specification.

- -silent

  An optional switch that suppresses the error that is normally generated if any object within object_spec does not exist.

Examples

The following example shows command invocations with and without the -stop_on pin option to illustrate its effect.

```
# find true fan-in of pin block1.sib1.si
```

get_icl_fanins block1.sib1.si

```
{si}

# Went through block1.si as -stop_on was defaulted to none.
# Reached ICL port si on the current_design find first pin in fan-in of
# block1.sib1.si
```

get_icl_fanins block1.sib1.si -stop_on pin

```
{block1.si}

# Stopped on block1.si as -stop_on was specified as pin.
```

Related Topics

get_icl_ports

get_icl_fanouts

get_icl_fanins_in_module

get_icl_fanouts_in_module

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_fanins_in_module

Context: all contexts

Mode: all modes

Prerequisites: The ICL module that contains the specified objects must be elaborated.

Returns a collection of objects found in the fan-in of the specified icl_pin_in_module or icl_port objects.

Usage

get_icl_fanins_in_module ***obj_spec*** [-silent]

Description

Returns a collection of all icl_pin_in_module and icl_port objects found in the fan-in of the specified icl_pin_in_module or icl_port objects.

While the command get_icl_fanins can only be used when the current design is set, the command get_icl_fanins_in_module can be used without a current design being present. There is only one prerequisite: the ICL module containing the elements in the ***obj_spec*** must be elaborated. This can be achieved by means of the command elaborate_icl_modules.

Arguments

- ***obj_spec***

  A required value that specifies a collection of one or more icl_pin_in_module or icl_port objects.

- -silent

  An optional switch that suppresses all warnings and all error messages associated with the command. If the -silent switch is used, the command returns an empty collection instead of issuing an error in situations that would normally result in an error.

Examples

elaborate_icl_module chip

set top_tdo [get_icl_ports tdo -of_module chip]

```
{tdo}
```

get_icl_fanins_in_module $top_tdo

```
{tap_I1.tdo}
```

set tap_tdi [get_icl_objects tap_I1.tdi -object_type icl_pin_in_module -in_module chip]

```
{tap_I1.tdi}
```

get_icl_fanins_in_module $tap_tdi

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{tdi}
```

Related Topics

elaborate_icl_modules

get_icl_fanins

get_icl_fanouts

get_icl_objects

get_icl_fanouts_in_module

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_fanouts

Context: all contexts

Mode: all modes

Prerequisites: The current design must be set.

Returns a collection of ICL objects found in the fanout of the specified ICL objects.

Usage

get_icl_fanouts ***object_spec*** [{-stop_on none | pin} | -scan_path] [-silent]

Description

Returns a collection of ICL objects found in the fanout of the specified ICL objects.

The types of the objects in the ***object_spec*** and the result collection depend on the usage of the -scan_path switch. If you omit this switch, the only possible object types in the object_spec and result collection are icl_port and icl_pin. If you use the -scan_path switch, the possible object types in the object_spec and the result collection are icl_port, icl_pin, icl_scan_mux, icl_scan_register, and icl_one_hot_scan_group. Use the -stop_on option to specify if the command should look for the furthest objects in the fanout or the closest. Use “none” in the former case and “pin” in the latter case. The -stop_on and -scan_path switches are mutually exclusive.

Arguments

- ***object_spec***

  A required value that specifies a single name, a list, or a collection of ICL objects, the fanout of which will be analyzed. The allowed types of the objects in the object_spec depend on the usage of the -scan_path switch, as discussed in the description earlier in this topic.

- -stop_on none | pin

  An optional switch and literal pair that specifies the following:

  none — The fanout walk traverses the connections and the ICL pins in the fanout of the object. The trace reports icl_pin objects with no fanout, icl_pin objects without any icl_pin or icl_port objects in their fanout, or icl_port objects that were reached in the fanout walk. This is the default.

  pin — The fanout walk traverses the connection and reports the icl_pin and icl_port objects in the fanout of the object. This could be an ICL output pin above the object or an ICL input pin on child ICL instances. The walk may still, however, stop on an ICL port if the ICL pin does not fanout to ICL pins.

- -scan_path

  An optional switch that limits the scope of the tracing to scan paths, but also allows the usage of all object types of elements that can be encountered on a scan path: icl_port, icl_pin, icl_scan_mux, icl_scan_register, and icl_one_hot_scan_group. When you use this switch, the object_spec and the returned collection can contain objects of the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

aforementioned object types. The tracing always stops immediately at the next encountered element. Tracing forward from a ScanRegister considers the path from the rightmost bit to another element on the scan path only, but not the parallel outputs of the update stage.

- -silent

  An optional switch that suppresses the error that is normally generated if any object within object_spec does not exist..

Examples

The following example shows the get_icl_fanouts command invocations with and without the -stop_on pin option to illustrate its effect.

```
# find true fanout of pin block1.sib1.so
```

get_icl_fanouts block1.sib1.so

```
{so}

# Went through block1.so as -stop_on was defaulted to none. Reached ICL
# port so on the current_design find first pin in fanout of block1.sib1.so
```

get_icl_fanouts block1.sib1.so -stop_on pin

```
{block1.so}

# Stopped on block1.so as -stop_on was specified as pin.
```

Related Topics

get_fanins

get_icl_ports

get_icl_fanins

get_icl_fanins_in_module

get_icl_fanouts_in_module

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_fanouts_in_module

Context: all contexts

Mode: all modes

Prerequisites: The ICL module that contains the specified objects must be elaborated.

Returns a collection of objects found in the fanout of the specified icl_pin_in_module or icl_port objects.

Usage

get_icl_fanouts_in_module ***obj_spec*** [-silent]

Description

Returns a collection of all icl_pin_in_module and icl_port objects found in the fanout of the specified icl_pin_in_module or icl_port objects.

While the command get_icl_fanouts can only be used when the current design is set, the command get_icl_fanouts_in_module can be used without a current design being present. There is only one prerequisite: the ICL module containing the elements in the ***obj_spec*** must be elaborated. This can be achieved by means of the command elaborate_icl_modules.

Arguments

- ***obj_spec***

  A required value that specifies a collection of one or more icl_pin_in_module or icl_port objects.

- -silent

  An optional switch that suppresses all warnings and all error messages associated with the command. If the -silent switch is used, the command returns an empty collection instead of issuing an error in situations that would normally result in an error.

Examples

elaborate_icl_module chip

set top_tdi [get_icl_ports tdi -of_module chip]

```
{tdi}
```

get_icl_fanouts_in_module $top_tdi

```
{sib_I1.si tap_I1.tdi block1_I1.si}
```

set tap_tdo [get_icl_objects tap_I1.tdo -object_type icl_pin_in_module -in_module chip]

```
{tap_I1.tdo}
```

get_icl_fanouts_in_module $tap_tdo

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{tdo}
```

Related Topics

elaborate_icl_modules

get_icl_fanins

get_icl_objects

get_icl_fanouts

get_icl_fanins_in_module

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_instances

Context: all contexts

Mode: all modes

Returns a collection of all ICL instances instantiated relative to the current design that match the specified <**name_patterns*> list or -filter expression.

Usage

Usage 1: Without the -hierarchical option

get_icl_instances [*name_patterns*] [-of_modules *icl_module_objects*]
[-parent_of_instances *instance_objects* | -below_instances *instance_objects* |
-of_pins *icl_pin_objects* | -of_instances *design_instance_objects*]
[-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Usage 2: With the -hierarchical option

get_icl_instances [*leaf_name_patterns*] [-of_modules *icl_module_objects*]
[-parent_of_instances *instance_objects* | -below_instances *instance_objects*]
[-hierarchical] [-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Usage 3: With no name pattern

get_icl_instances [-of_modules *icl_module_objects*]
[-parent_of_instances *instance_objects* | -below_instances *instance_objects* |
-of_pins *icl_pin_objects*]
[-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Description

Returns a collection of all ICL instances instantiated relative to the current design that match the specified <**name_patterns*> list or -filter expression.

- **Usage 1** — The name patterns are expressed as hierarchical instance path names relative to the current design. You can have any number of wildcards or regular expressions in the names between the hierarchy separator symbols as shown in the examples for this command. You can use the dot or the slash as the hierarchy separator when not using the -regexp option. You can only use the slash as the hierarchy separator when using the -regexp option because the dot is the wildcard symbol when using regular expressions.
- **Usage 2** — The name patterns are leaf instance names with any number of wildcards or regular expressions. Unlike in Usage 1, the leaf instance name pattern is searched relative to all ICL instances found below the current design.
- **Usage 3** — A name_patterns argument is not specified so that all ICL instances are selected and then filtered by the other criteria specified by the other optional arguments.

If you issue this command but have not previously loaded any icl files, it returns an empty collection.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of ICL instances. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no name_patterns are specified, the tool searches all ICL instances that match the results returned by the -filter attribute_equation filter as shown in Usage 3. In Usage 2, described previously, the name patterns cannot include slashes or dots, which are hierarchy separator symbols. In Usage 1, you can use the dot or slash as the hierarchy separator when not using the -regexp option. You can only use the slash as the hierarchy separator when using the -regexp option because the dot is the wildcard symbol when using regular expressions.

  See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a complete description of the pattern syntax when the -regexp option is used and not used.

- -of_modules *icl_module_objects*

  An optional switch and value pair that constrains the command to return only instances of ICL modules specified by icl_module_objects. The icl_module_objects is a Tcl list of one or more ICL module names, or a collection of one or more ICL module objects as returned by the get_icl_modules command.

  Note

  This command also considers prefixes and suffixes defined with the set_module_matching_options command. If the base name of a design module matches an ICL module, the tool adds the related module to compute the results.

- -parent_of_instances *instance_objects*

  An optional switch and value pair that constrains the command to return a collection of ICL instance objects that are the parent instances of the specified instance_objects. The returned collection is filtered to only return a given parent instance once. When this option is specified, the optional name_patterns option is a string that is matched against the leaf instance name of every parent instance of the collection. An error is generated if the specified name_patterns contain more than one level of hierarchy when the -parent_of_instances option is used.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for ICL instances below the specified *instance_objects*. The specified *name_patterns* are searched relative to each instance found in the *instance_objects* list.

  If the specified *instance_spec* is an empty collection, all instances below the current design are considered. See the description of the -below_instances switch of the get_instances command description for the justification of this behavior.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -of_pins *icl_pin_objects*

  An optional switch and value pair that constrains the command to return the ICL instance to which the ICL pins belong. The returned collection is filtered to return any given ICL instance only once, even if the instance belongs to many ICL pins in the icl_pin_objects list. When this option is specified, the optional name_patterns must be leaf_name patterns that are checked against the leaf name of the ICL instances. An error is generated if the specified name_patterns contains more than one level of hierarchy when the -of_pins option is used.

- -of_instances *design_instance_objects*

  An optional switch and value pair that constrains the command to return the ICL instances corresponding to the design instances specified by *design_instance_objects*. The *design_instance_objects* is a Tcl list of design instances or a collection of one or more design instances as returned by the get_instances command.

  Note

  This command also considers prefixes and suffixes defined with the set_module_matching_options command. If the base name of a design module matches an ICL module, the tool adds the related module to compute the results.

- -hierarchical

  An optional string that specifies to perform pattern matching relative to all ICL instances found below the current design. When you specify this option, the name_patterns argument cannot include slashes or dots, which are hierarchy separator symbols.

- -filter *attribute_equation*

  An optional switch and string pair that specifies to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the ICL instance object type. See section "Attribute Filtering Equation Syntax" on page 3832 for more details on the filtering attribute equation format.

- -regexp

  An optional switch that directs the tool to interpret the value of the name_patterns argument and the arguments of the =~ and !~ -filter expressions as regular expressions instead of as simple wildcard patterns. You must properly escape the bracket "[]" symbols so that they are interpreted as part of the regular expression and not interpreted by the Tcl shell. See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a description of the regular expression syntax. You must use the slash as a hierarchy separator when using the -regexp option.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for instances matching the name_patterns.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

Note

You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

Examples

Example 1

The following example illustrates Usage 3 and returns the collection of all instances of ICL modules modb and modd found anywhere below the current design:

get_icl_instances -of_modules {modb modd}

```
{u2 u1/u5}
```

Example 2

The following example illustrates Usage 3 in which the collection of all instances is filtered based on an attribute_equation of user-defined attributes.

get_icl_instances -filter {myType1 || myType2 == "ABC"}

```
{u2 u1.u5}
```

Example 3

The following example illustrates Usage 1 in which the collection of all instances matching a list of hierarchical name patterns is returned. The names u1/* and u*/u* are only searched relative to the current design.

get_icl_instances {u1/* u*/u*}

```
{u4.u5 u1.u7}
```

Example 4

The following example illustrates Usage 2 in which the collection of all instances matching a leaf name pattern is returned. The leaf instance name u3* searches for all instances found below the current design.

get_icl_instances u3* -hierarchical

```
{u3 u4.u30 u4.u5.u31 u5.u3}
```

Example 5

The following example shows how to use the -filter switch in Usage 3, with the search criteria specified by the optional introspection commands, tessent_instrument_type, and tessent_instrument_subtype.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_instances

```
{DATA_MEM_1 DATA_MEM_1_interface_inst
PROG_MEM_2 PROG_MEM_2_interface_inst
aaa_bbb_ccc_processor_core_rtl2_tessent_occ_intclk_inst
aaa_bbb_ccc_processor_core_rtl2_tessent_occ_intclk_inst.tdr_sib
add_clock_inst_0
lfxt_c_bbb_ccc_processor_core_rtl2_tessent_occ_lfxt_clk_int_inst
lfxt_c_bbb_ccc_processor_core_rtl2_tessent_occ_lfxt_clk_int_inst.tdr_sib
lfxt_mod_new_mem lfxt_mod_new_mem_interface_inst
processor_core_rtl1_tessent_mbist_bap_inst
processor_core_rtl1_tessent_mbist_c1_controller_inst
processor_core_rtl1_tessent_mbist_c2_controller_inst
processor_core_rtl1_tessent_sib_mbist_inst
processor_core_rtl1_tessent_sib_sti_inst
processor_core_rtl2_tessent_edt_c1_inst
processor_core_rtl2_tessent_edt_c1_tdr_inst
processor_core_rtl2_tessent_occ_dco_clk_inst
processor_core_rtl2_tessent_occ_dco_clk_inst.tdr_sib
processor_core_rtl2_tessent_occ_lfxt_clk_inst
processor_core_rtl2_tessent_occ_lfxt_clk_inst.tdr_sib
processor_core_rtl2_tessent_sib_edt_inst
processor_core_rtl2_tessent_sib_occ_inst
processor_core_rtl2_tessent_sib_sri_inst
processor_core_rtl2_tessent_tdr_sri_ctrl_inst submod_new_mem
submod_new_mem_interface_inst}
```

get_icl_instances -filter tessent_instrument_type==mentor::edt

```
{processor_core_rtl2_tessent_edt_c1_inst}
```

get_icl_instances -filter {tessent_instrument_type==mentor::memory_bist && tessent_instrument_subtype==controller}

```
{processor_core_rtl1_tessent_mbist_c1_controller_inst
processor_core_rtl1_tessent_mbist_c2_controller_inst}
```

Related Topics

get_icl_modules

get_icl_ports

read_icl

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_modules

Context: all contexts

Mode: all modes

Returns a collection of all ICL modules that match the specified name_patterns list or -filter expression.

Usage

get_icl_modules [*name_patterns*] [-of_instances *instance_objects* |
-below_instances *instance_objects* | -of_modules *design_module_objects* |
-parameterized_views_of_modules *icl_module_objects*]
[-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Description

Returns a collection of all ICL modules that match the specified *name_patterns* list or -filter expression.

By default, this command returns all ICL modules below the current design. You can use the -of_instances argument to constrain the search.

You can have any number of wildcards or regular expressions in the name patterns.

If you issue this command but have not previously loaded any icl files, it returns an empty collection.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of ICL modules. If no name_patterns are specified, the tool searches all ICL modules that match the results returned by the -filter attribute_equation filter.

- -of_instances *instance_objects*

  An optional switch and value pair that constrains the command to return only ICL modules of the specified *instance_objects*. The instance_objects is a Tcl list of one or more ICL instance names, or a collection of one or more ICL instance objects as returned by the get_icl_instances command. You could use the module_name attribute of the instance objects to perform a similar filtering but this method is more direct and efficient.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to return only ICL modules corresponding to instances found below the specified *instance_objects*.

  If the specified *instance_spec* is an empty collection, all instances below the current design are considered. See the description of the -below_instances switch of the get_instances command description for the justification of this behavior.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -of_modules *design_module_objects*

  An optional switch and value pair that constrains the command to return a collection of ICL modules corresponding to the design modules specified by design_module_objects. The design_module_objects is a Tcl list of one or more design modules, or a collection of one or more design modules as returned by the get_modules command.

  Note

  This command also considers prefixes and suffixes defined with the set_module_matching_options command. If the base name of a design module matches an ICL module, the tool adds the related module to compute the results.

- -parameterized_views_of_modules *icl_module_objects*

  An optional switch and value pair that constrains the command to return a collection of ICL modules corresponding to all the parameterized views of the ICL module objects specified. The *icl_module_objects* can be specified as a Tcl list of ICL module names or a collection of ICL modules. Whether *icl_module_objects* contains a parameterized view or the master copy of an ICL module, this switch instructs the command to find all parameterized views and return them. Using this switch is equivalent to using the master_name attribute of an ICL module and requesting all ICL modules having the same master_name value. Using this switch, however, is more efficient. You use this switch to determine whether an ICL module is uniquely instantiated or not and to find out if an ICL module has multiple parameterized views.

- -filter *attribute_equation*

  An optional switch and string pair that specifies to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the ICL module object type. See section "Attribute Filtering Equation Syntax" on page 3832 for more details on filtering attribute equation format. The =~ and !~ matching operator use simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option.

- -regexp

  An optional switch that directs the tool to interpret the value of the name_patterns argument and the arguments of the =~ and !~ -filter expressions as regular expressions instead of as simple wildcard patterns. See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for modules matching the name_patterns.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

name_patterns disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

Note

You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

Examples

Example 1

The following example returns ICL modules whose name matches a* or b*.

get_icl_modules {a* b*}

```
{ab abc b bcd}
```

Example 2

The following example returns ICL modules having one or two characters.

get_icl_modules {[:alnum:]{1,2}} -regexp

```
{ab b}
```

Example 3

This example shows how to report the names of the parameterized views for a module.

puts “[get_name_list [get_icl_modules -parameterized_views_of_modules block]]”

```
block block@PV1 block@PV2
```

Related Topics

get_icl_instances

get_icl_ports

read_icl

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_module_parameter_list

Context: all contexts

Mode: all modes

Returns the list of parameters in an ICL module.

Usage

get_icl_module_parameter_list ***module_object*** [-overridden]

Description

Returns the list of parameters in an ICL module.

Arguments

- ***module_object***

  A required string that specifies the parameterized ICL module objects whose parameters are to be returned.

- -overridden

  An optional switch that specifies to only list those parameters whose default values are overridden during an instantiation.

Return Values

List of parameters.

Examples

n the following example, this command is used to obtain the list of parameters overridden in the specific instantiation. Note how the get_icl_instances command is used to find the ICL instance *u1/myinst_i1*; this ensures that the get_attribute_value_list command returns the module_name attribute of the ICL instance and not the design instance in case a design instance of the same name exists.

```
set icl_inst [get_icl_instance u1/myinst_i1]

set params [get_icl_module_parameter_list [get_attribute_value_list $icl_inst \
                -name module_name] -overridden]
```

Related Topics

get_icl_module_parameter_value

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_module_parameter_value

Context: all contexts

Mode: all modes

Returns the value of the parameter on the specified module and indicates whether it is the default value or is overridden.

Usage

get_icl_module_parameter_value ***parameter_name*** [-of_module *icl_module_object*] [-format {icl | tcl}]

Description

Returns the value of the parameter on the specified module and indicates whether it is the default value or is overridden.

Arguments

- ***parameter_name***

  A required string that specifies the name of the parameter whose value is to be returned.

- -of_module *icl_module_object*

  An optional switch and string pair that specifies the ICL module whose parameter value is to be returned. When omitted, the parameters of the current design are introspected.

- -format {icl | tcl}

  An optional switch and literal pair to specify which format the tool uses to return binary and hexadecimal numbers.

  icl — Use this switch if you want the tool to return binary or hexadecimal numbers in the ICL format. This is the default.

  ```
  <size>'b<num>
  <size>'h<num>
  ```

  Note

  The tool only specifies the <size> part of the format if the parameter is a sized binary or hexadecimal number.

  tcl — Use this switch if you want the tool to return binary or hexadecimal numbers in the Tcl format:

  ```
  0b<num>
  0x<num>
  ```

Return Values

The value of the requested ICL module parameter.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

In the following example, the values of all overridden parameters are echoed in the transcript. Note how the get_icl_instances command is used to find the ICL instance *u1/myinst_i1*; this ensures that the get_attribute_value_list command returns the module_name attribute of the ICL instance and not the design instance in case a design instance of the same name exists.

```
set icl_inst [get_icl_instance u1/myinst_i1]

set icl_mod [get_attribute_value_list $icl_inst -name module_name]

set params [get_icl_module_parameter_list $icl_mod -overridden]

foreach param $params {
    puts "parameter $param = [get_icl_module_parameter_value $param \
        -of_mod $icl_mod]"
}
```

Related Topics

get_icl_module_parameter_list

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_network

Context: patterns -silicon_insight -ijtag

Mode: setup

Returns various values for the specified option.

Usage

get_icl_network { *node_name* {-node_type | -inputs | -outputs | -current_input_index } | -current_scan_path_length | -is_current_scan_path_testable | -current_scan_path_nodes | -nodes_of_type *node_type* | -nodes_of_status *diagnosis_status* }

Description

The command has two major forms: retrieving information about a given node or retrieving general ICL network-related data.

Use this command when manually debugging IJTAG networks as described in "Diagnosing and Debugging IJTAG Networks" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

Arguments

- *node_name* {-node_type | -inputs | -outputs | -current_input_index }

  An optional variable and literal pair that returns information about the specified node. Options are:

  -node_type — The node type: port, SIB, scan mux, or scan register.

  -inputs — Input nodes for the specified node.

  -outputs — Output nodes for the given node.

  -current_input_index — Input index for the currently active scan path.

- -current_scan_path_length

  An optional switch that returns the length of the current scan path in terms of the number of flops from the input signals to the output signals.

- -is_current_scan_path_testable

  An optional switch that returns true or false depending on whether the current scan path is testable or not. A scan path is deemed untestable if one or more elements in the scan path are untestable. A scan element may be untestable if the following attributes are set to on: tessent_ignore_during_icl_verification and explicit_iwrite_only.

- -current_scan_path_nodes

  An optional switch that returns a list of scan element names in the current scan path from TDI to TDO.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -nodes_of_type *node_type*

  An optional switch and variable pair that returns the list of nodes of the specified type in the IJTAG network.

- -nodes_of_status *diagnosis_status*

  An optional switch and variable pair that returns a list of nodes of the specified diagnosis status, based on the last performed diagnosis. Options are:

  suspect

  bad

  good

  untested

Examples

get_icl_network -current_scan_path_nodes

```
tdo tap_I1.IRMux@0 tap_I1.DRMux@1 sib_I4@0 sib_I3@0 sib_I2@0 sib_I1@0 tdi
```

Related Topics

analyze_icl_suspect

assert_icl_sib

create_icl_flush_pattern

create_icl_setup_pattern

deassert_icl_sib

get_icl_scan_mux

isolate_icl_node

report_icl_network

set_icl_network

set_icl_scan_mux

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_objects

Context: all contexts

Mode: all modes

Returns a collection of ICL object matching the requested criteria.

Usage

get_icl_objects [*name_patterns*] **-object_types *object_types***
[-below_instances *icl_instance_spec* | -parent_of_instances *icl_instance_spec* | -in_modules *icl_module_spec* | -of_objects *icl_object_specs*]
[-hierarchical]
[-direction *direction*]
[-function *function*]
[-filter *attribute_equation*]
[-regexp]
[-nocase]
[-silent]

Description

This command combines the complete usage of get_icl_modules, get_icl_instances, get_icl_ports, and get_icl_pins plus provides introspection into other ICL object types as well as the ability to see objects inside a module before it is elaborated. Table 5-4 lists the available object types and against which object they can be requested from. The object types icl_instance, icl_pin, icl_scan_interface, icl_scan_register, icl_scan_mux and icl_one_hot_scan_group are only available when ICL is elaborated. You can get access to their _of_module/_in_module counterpart without requiring the ICL to be elaborated, but you are limited to the object directly declared in the ICL module. This difference is illustrated in Example 2 below.

Arguments

- *name_patterns*

  An optional Tcl list of one or more strings that specifies patterns to be used to filter the returned list of ICL object based on their name attributes. If no *name_patterns* are specified, the tool searches all ICL object that match the other criteria.

- **-object_types *object_types***

  A required Tcl list of one or more strings that specifies ICL object types to consider in the search. The tool supports the following ICL object types:

  - icl_module
  - icl_port
  - icl_instance
  - icl_instance_in_module

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

  - icl_pin
  - icl_pin_in_module
  - icl_scan_interface
  - icl_scan_interface_of_module
  - icl_scan_register
  - icl_scan_register_in_module
  - icl_scan_mux
  - icl_scan_mux_in_module
  - icl_one_hot_scan_group
  - icl_one_hot_scan_group_in_module
  - icl_alias
  - icl_alias_in_module
  - icl_sib

- -below_instances *icl_instance_spec*

  An optional switch-value pair used to specify a Tcl list of one or more names of icl_instance objects or a collection containing one or more icl_instance objects. This option can only be used when the -object_type switch specifies at least one of the following object types:

  - icl_instance
  - icl_pin
  - icl_scan_interface
  - icl_scan_register
  - icl_scan_mux
  - icl_one_hot_scan_group
  - icl_alias

  If the specified *instance_spec* is an empty collection, all instances below the current design are considered. See the description of the -below_instances switch of the get_instances command description for the justification of this behavior

- -parent_of_instances *icl_instance_spec*

  An optional switch-value pair used to specify a Tcl list of one or more names of icl_instance objects or a collection containing one or more icl_instance objects. This option can only be used when the -object_type switch specifies “icl_instance”. When you specify this option, the optional *name_patterns* must be leaf_name patterns that are checked against the leaf

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

names of the ICL instances. The tool generates an error if the specified *name_patterns* contain more than one level of hierarchy when you use the -parent_of_instances option.

- -in_modules *icl_module_spec*

  An optional switch-value pair used to specify a Tcl list of one or more names of ICL modules or a collection contain one or more ICL module objects. The permitted types in the -object_types switch value are “icl_instance_in_module”, “icl_pin_in_module”, “icl_scan_register_in_module”, “icl_scan_mux_in_module”, “icl_one_hot_scan_group_in_module” and “icl_alias_in_module” when the -in_modules switch is used.

- -of_objects *icl_object_spec*

  An optional switch-value pair used to specify a Tcl list of one or more names of ICL objects or a collection containing one or more ICL objects. Table 5-4 shows the permitted type for the *icl_object_spec* objects based on the -object_types requested.

Table 5-4. ICL Objects and Applicable Options

icl_object types in the -of_objects icl_object_spec	permitted types in the -object_types list
icl_alias	icl_instance
icl_alias_in_module	icl_module
icl_instance	icl_alias icl_module icl_one_hot_scan_group icl_pin icl_scan_interface icl_scan_mux icl_scan_register
icl_instance_in_module	icl_module icl_pin_in_module
icl_module	icl_instance icl_port icl_scan_interface_of_module
icl_one_hot_scan_group	icl_instance
icl_one_hot_scan_group_in_module	icl_module
icl_pin	icl_instance icl_port icl_scan_interface

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-4. ICL Objects and Applicable Options (cont.)

icl_object types in the -of_objects icl_object_spec	permitted types in the -object_types list
icl_pin_in_module	icl_instance_in_module icl_port
icl_port	icl_module icl_pin icl_scan_interface_of_module
icl_scan_interface	icl_instance icl_pin
icl_scan_interface_of_module	icl_module icl_port
icl_scan_mux	icl_instance
icl_scan_mux_in_module	icl_module
icl_scan_register	icl_instance
icl_scan_register_in_module	icl_module
icl_sib	icl_sib

- -hierarchical

  An optional string that specifies to perform pattern matching relative to all ICL instances found below the current design. The *name_patterns* argument cannot include more than one dot when the -hierarchical option is specified and the -object_types switch specifies one of the following: icl_pin, icl_scan_interface, icl_scan_register, icl_scan_mux, icl_one_hot_scan_group, icl_alias or icl_sib. The dot is the hierarchy separator symbols in ICL.When any other object type is specified the name_patterns argument must not contain any hierarchy separator.

- -direction *direction*

  An optional switch and literal pair to filter results based on the value of the direction attribute on the objects of type icl_port, icl_pin, or icl_pin_in_module. When you omit this argument, the tool considers pins or ports with any direction.

- -function ***function_list***

  An optional switch and value pair to filter results based on the value of the ijtag_function attribute on the objects of type icl_port, icl_pin, or icl_pin_in_module. When you omit this argument, the tool considers ports or pins with any ijtag_function value. The ijtag_function values are the ICL port function as found in the ICL description, where multiple words are separated by underscores instead of using camel casing without the "_port" termination. For example, a port declared as a ScanInPort in ICL uses the function name scan_in in the ICL data model.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -filter *attribute_equation*

  An optional switch and string pair to filter results based on the *attribute_equation* string. The *attribute_equation* string filters icl_object attributes reported by report_attributes. The attributes used within the equation must exist for all the object types listed in the -object_type switch. See section "Attribute Filtering Equation Syntax" on page 3832 for more details on the filtering attribute equation format. The =~ and !~ matching operators use simple wildcard matching unless Posix extended regular expression is specified by the -regexp option.

- -regexp

  An optional switch that directs the tool to interpret the value of the *name_patterns* argument and the arguments of the =~ and !~ -filter expressions as regular expressions instead of as simple wildcard pattern.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for ICL object names matching *name_patterns*.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

  Note

  You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

Examples

Example 1

The following example illustrate how you can query objects of multiple types using a single invocation of the get_icl_objects command. This cannot be done when using get_icl_ports and get_icl_pins command. The example usage queries for ICL ports and ICL pins with ijtag_function equal to capture_en within the elaborate ICL design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
get_icl_objects -object_type {icl_pin icl_port} -function capture_en
{corea_rtl1_tessent_mbist_bap_inst.capture_en
 corea_rtl1_tessent_sib_mbist_inst.ijtag_ce
 corea_rtl1_tessent_sib_sti_inst.ijtag_ce
 corea_rtl2_tessent_edt_c1_tdr_inst.ijtag_ce
 corea_rtl2_tessent_occ_child_clkb_inst.ijtag_ce
 corea_rtl2_tessent_occ_child_clkb_inst.tdr_sib.capture_en
 corea_rtl2_tessent_occ_child_nocg_clka_inst.ijtag_ce
 corea_rtl2_tessent_occ_child_nocg_clka_inst.tdr_sib.capture_en
 corea_rtl2_tessent_occ_child_nocg_clkc_inst.ijtag_ce
 corea_rtl2_tessent_occ_child_nocg_clkc_inst.tdr_sib.capture_en
 corea_rtl2_tessent_sib_edt_inst.ijtag_ce
 corea_rtl2_tessent_sib_occ_inst.ijtag_ce
 corea_rtl2_tessent_sib_sri_inst.ijtag_ce
 corea_rtl2_tessent_tdr_sri_ctrl_inst.ijtag_ce ijtag_ce}
```

Example 2

The following example illustrates the difference between the object types ending with "_of_module" and those that do not. The icl_scan_interface object_type is a hierarchical object_type that exists in the current design and on ICL instances. Their names are the full hierarchical paths of their associated ICL instances followed by the leaf_name of the scan interface. They only exist when the ICL is elaborated. The icl_scan_interface_of_module object_type is a non-hierarchical object_type that exists on ICL modules. Their names are the leaf_name of the scan interface. They exist even if ICL is not elaborated. The top part of the example uses the hierarchical object types to find specific instances of an ICL scan interface and follows to extract the ICL pins with function scan_in from it. The second part of the example uses the non-hierarchical object types to find the ICL scan interface of a module and follow the interface to extract the ICL port with function scan_in from it. The second method is useful when you want to extract information about ICL modules and the top level ICL is not yet extracted and elaborated. It is also useful to introspect ICL modules not part of the current ICL elaboration tree.

```
# This part shows introspecting the ICL pins of ICL scan interfaces when ICL is elaborated

set icl_scan_int [get_icl_objects -object_type icl_scan_interface \
    -of_objects [get_icl_instances corea_rtl2_tessent_sib_sri_inst]]

{corea_rtl2_tessent_sib_sri_inst.client
 corea_rtl2_tessent_sib_sri_inst.host}

set icl_pins [get_icl_objects -object_type icl_pin \
                    -of_object $icl_scan_int \
                    -function scan_in]

{corea_rtl2_tessent_sib_sri_inst.ijtag_si
 corea_rtl2_tessent_sib_sri_inst.ijtag_from_so}

# This part shows introspecting the ICL pins of ICL scan interfaces when ICL is elaborated

set icl_scan_int [get_icl_objects -object_type icl_scan_interface \
    -of_objects [get_icl_instances corea_rtl2_tessent_sib_sri_inst]]

{corea_rtl2_tessent_sib_sri_inst.client
 corea_rtl2_tessent_sib_sri_inst.host}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set icl_pins [get_icl_objects -object_type icl_pin \
                    -of_object $icl_scan_int \
                    -function scan_in]
```

```
{corea_rtl2_tessent_sib_sri_inst.ijtag_si
 corea_rtl2_tessent_sib_sri_inst.ijtag_from_so}
```

```
# This part shows introspecting the ICL ports of ICL scan interfaces of ICL modules when
# ICL is NOT elaborated

set icl_scan_int_of_mod [get_icl_objects -object_type icl_scan_interface_of_module \
    -of_objects [get_icl_modules corea_rtl2_tessent_sib_1]]
```

```
{client host}
```

```
set icl_port [get_icl_objects -object_type icl_port -of_objects $icl_scan_int_of_mod
    -function scan_in]
```

```
{ijtag_si ijtag_from_so}
```

Related Topics

get_common_parent_instance

get_icl_ports

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_pins

Context: all contexts

Mode: all modes

Returns a collection of all hierarchical ICL pins instantiated relative to the current design that match the specified name_patterns list or -filter expression.

Usage

Usage 1: Without the -hierarchical option

get_icl_pins *name_patterns* [-direction input | output] [-function *function*]
[-of_instances *instance_objects* | -below_instances *instance_objects* |
-of_pins *design_pin_objects*]
[-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Usage 2: With the -hierarchical option

get_icl_pins *leaf_name_patterns* | *leaf_instance/leaf_name_patterns* -hierarchical
[-below_instances *instance_objects*] [-filter *attribute_equation*]
[-direction input | output] [-function *function*]
[-regexp] [-nocase] [-silent]

Usage 3: With no name pattern

get_icl_pins [-filter *attribute_equation*] [-direction input | output] [-function function] [-regexp]
[-nocase] [-silent]

Description

Returns a collection of all hierarchical ICL pins instantiated relative to the current design that match the specified *name_patterns* list or -filter expression.

- **Usage 1** — The name patterns are expressed as hierarchical pin pathnames relative to the current design. You can have any number of wildcards or regular expressions in the names between the hierarchy separator symbols as shown in the examples that follow. You can use the dot or the slash as the hierarchy separator when not using the -regexp option. You can only use the slash as the hierarchy separator when using the -regexp option because the dot is the wildcard symbol when using regular expressions.
- **Usage 2** — The name patterns are leaf pin names or leaf instance name/leaf pin names with any number of wildcards or regular expressions. Unlike in Usage 1, the leaf name pattern is searched relative to all instances found below the current design. There can only be one hierarchy separator when using the -hierarchical option as shown in Example 3.
- **Usage 3** — A *name_patterns* argument is not specified so all ICL pins are selected and then filtered by the other criteria specified by the other optional arguments.

If you issue this command but have not previously loaded any icl files, it returns an empty collection.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- *name_patterns* | *leaf_instance/leaf_name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of pins. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no *name_patterns* are specified, the tool searches all ICL pins that match the results returned by the -filter *attribute_equation* filter as shown in Usage 3. In Usage 2, described previously, the name patterns are not permitted to include more than one slash or dot, which are hierarchy separator symbols. See section "Attribute Filtering Equation Syntax" on page 3832 for a complete description of the pattern syntax when the -regexp option is used or not.

- -direction input | output

  An optional switch and literal pair that specifies to filter results based on the value of the direction attribute on the pin object. When omitted, pins with any direction are considered.

- -of_instances *instance_objects*

  An optional switch and value pair that constrains the command to return only ICL pins of the specified instance_objects. The instance_objects is a Tcl list of one or more ICL instance names, or a collection of one or more ICL instance objects as returned by the get_icl_instances command. You could use the module_name attribute of the instance objects to perform a similar filtering but this method is more direct and efficient.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for ICL pins below the specified *instance_objects*. Specified *name_patterns* are searched relative to the specified *instance_objects* when this option is used.

  If the specified *instance_objects* is an empty collection, all instances below the current design are considered. See the description of the -below_instances switch of the get_instances command description for the justification of this behavior.

- -of_pins *design_pin_objects*

  An optional switch and value pair that constrains the command to return only ICL pins corresponding to the design pins specified by design_pin_objects. The design_pin_objects is a Tcl list of one or more design pins, or a collection of one or more design pins as returned by the get_pins command.

  The tessent_design_rtl_ports and tessent_design_gate_ports attributes are used to retrieve the ICL pins from design pins. The tessent_design_rtl_ports attribute is used when in RTL mode and the current view is an RTL view. Otherwise, the tessent_design_gate_ports attribute is used.

  To use this command in no RTL mode when the current view is a netlist view, the tessent_design_gate_ports attribute must be up to date. In most cases, this is true after the set_current_design command is invoked. For sub-blocks, it is only true after write_design or update_icl_attributes_from_design is invoked. See "Post-Synthesis Update" in the *Tessent Shell User's Manual* for further information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

This command also considers prefixes and suffixes defined with the set_module_matching_options command. If the base name of a design module matches an ICL module, the tool adds the related module to compute the results.

- -hierarchical

  An optional string that specifies to perform pattern matching relative to all ICL instances found below the current design. When this option is specified, the *name_patterns* argument cannot include more then one slash or dot, which are the hierarchy separator symbols.

- -function function

  An optional switch and literal pair that specifies to filter results based on the value of the function attribute on the ICL pin object. When omitted, pins with any function are considered. The value function is a Tcl list of function names derived from the ICL port function where multiple words are separated by underscores instead of using camel casing. For example, a port declared as a ScanInPort in ICL has function scan_in in the ICL data model.

- -filter *attribute_equation*

  An optional switch and string pair that specifies to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the ICL pin object type. See section "Attribute Filtering Equation Syntax" on page 3832 for more details on filtering attribute equation format. The =~ and !~ matching operators use simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option.

- -regexp

  An optional switch that directs the tool to interpret the value of the *name_patterns* argument and the arguments of the =~ and !~ -filter expressions as regular expressions instead of as simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for ICL pin names matching *name_patterns*.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

Examples

Example 1

The following example illustrates Usage 3 where the collection of all ICL pins is filtered based on an attribute_equation of user-defined attributes.

get_icl_pins -filter {myType1 || myType2 == "ABC"}

```
{u1.Y u4.u4.A0 u7.D}
```

Example 2

The following example illustrates Usage 1 where the collection of all ICL pins matching a list of hierarchical name patterns is returned. The names u1/Y and u*/Q are only searched relative to the current design. See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for examples of pattern matching with regular expressions.

get_icl_pins {u1/Y u*/Q}

```
{u1.Y u2.Q u7.Q}
```

Example 3

The following example illustrates Usage 2 where the collection of all ICL pins matching a leaf instance_name/leaf pin name pattern is returned. The tool searches for the leaf instance name u* in all ICL instances found below the current design, and then searches for the leaf pin names matching Y* on those instances.

get_icl_pins u*/Y* -hierarchical

```
{u1.Y u3.Y u4.u1.Y u4.u4.Y u4.u5.u1.Y u5.u1.Y u5.u3.Y u5.u4.Y u6.Y}
```

Example 4

The following example gets all ICL pins with function clock or to_clock in the ICL data base.

get_icl_pins -function {clock to_clock}

```
{PLL1.ref PLL1.vco mbist.clk}
```

Related Topics

get_common_parent_instance

get_icl_ports

read_icl

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_ports

Context: all contexts

Mode: all modes

Returns a collection of all ICL ports that match the specified name_patterns list or -filter expression.

Usage

get_icl_ports [*name_patterns*] [-of_modules *icl_module_objects* |
-of_ports *design_port_objects*] [-direction input | output]
[-function *function*] [-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Description

Returns a collection of all ICL ports that match the specified *name_patterns* list or -filter expression.

You can have any number of wildcards or regular expressions in the name patterns.

If you issue this command but have not previously loaded any icl files, it returns an empty collection.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of ports. If no *name_patterns* are specified, the tool searches all ports that match the results returned by the -filter *attribute_equation* filter, if one is specified.

- -of_modules *icl_module_objects*

  An optional switch and value pair that constrains the command to return only ICL ports on modules specified by *icl_module_objects*. The *icl_module_objects* is a Tcl list of ICL module names or a collection of one or more module objects as returned by the get_icl_modules command.

- -of_ports *design_port_objects*

  An optional switch and value pair that constrains the command to return only ICL ports corresponding to the design ports specified by *design_port_objects*. The *design_port_objects* is a Tcl list of design ports or a collection of one or more design ports as returned by the get_ports command.

  The tessent_design_rtl_ports and tessent_design_gate_ports attributes are used to retrieve the ICL ports from design ports. The tessent_design_rtl_ports attribute is used when in RTL mode and the current view is an RTL view. Otherwise, the tessent_design_gate_ports attribute is used.

  To use this command in no RTL mode when the current view is a netlist view, the tessent_design_gate_ports attribute must be up to date. In most cases this will be true after

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

the set_current_design command is invoked. For sub-blocks it will only be true after write_design or update_icl_attributes_from_design is invoked. See "Post-Synthesis Update" in the *Tessent Shell User's Manual* for further information.

- -direction input | output

  An optional switch and literal pair that specifies to filter results based on the value of the direction attribute on the port object. When omitted, ports with any direction are considered.

- -function function

  An optional switch and literal pair that specifies to filter results based on the value of the function attribute on the ICL port object. When omitted, ports with any function are considered. The value function is a Tcl list of function names derived from the ICL port function where multiple words are separated by underscores instead of using camel casing. For example, a port declared as a ScanInPort in ICL has function scan_in in the ICL data model.

- -filter *attribute_equation*

  An optional switch and string that specifies to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the ICL port object type. See section "Attribute Filtering Equation Syntax" on page 3832 for more details on filtering attribute equation format. The =~ and !~ matching operators used simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option or not.

- -regexp

  An optional switch that directs the tool to interpret the value of the *name_patterns* argument and the arguments of the =~ and !~ -filter expressions as regular expressions instead of as simple wildcard patterns. See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for ports matching the *name_patterns*.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

  Note

  You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

This example looks for ICL ports on the current design whose name match I* or O*.

get_icl_ports {I* O*}

```
{I1 I2 I3 I4 I5 I6 I7 I8 O1 O2 O3}
```

Example 2

This example looks for scalar ports having one or two characters on module ModA.

get_icl_ports {[[:alnum:]]{1,2}} -regexp -of_module ModA

```
{ab a b}
```

Related Topics

get_common_parent_instance

read_icl

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_scan_interface_chain_list

Context: all contexts

Mode: all modes

Returns the names of the explicitly defined scan chains for the specified client scan interface in an existing ICL top module, or the names of the explicitly defined scan chains for the specified client scan interface that are created in the new ICL top module during ICL extraction.

Usage

get_icl_scan_interface_chain_list **-name *name*** [-as_set]

Description

Returns the names of the explicitly defined scan chains for the specified client scan interface in an existing ICL top module, or the names of the explicitly defined scan chains for the specified client scan interface that are created in the new ICL top module during ICL extraction.

You must set the current design with set_current_design before using the get_icl_scan_interface_chain_list command.

Arguments

- **-name *name***

  A required switch and value pair that specifies the name of a scan interface, or a collection that contains exactly one element of type either “icl_scan_interface” or “icl_scan_interface_of_module” that specifies the scan interface, for which the list of explicitly defined scan chains is returned.

- -as_set

  An optional switch that instructs the tool to return the specifications done by means of the add_icl_scan_interfaces command, even if there is already an ICL module for the current design.

Return Values

If the ICL top module exists and the -as_set switch has been omitted, this command returns the names of the explicitly defined scan chains for the specified ScanInterface.

If the ICL top module does not exist or the -as_set switch has been used, the command returns the names of the new scan chains for the ScanInterface specified with the -chains argument of the add_icl_scan_interfaces command, which are created during ICL extraction.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_scan_interface_list

Context: all contexts

Mode: all modes

Returns the names of the scan interfaces in an existing ICL top module, if the ICL top module exists.

Usage

get_icl_scan_interface_list [-control_of object_spec | -as_set]

Description

Returns the names of the scan interfaces in an existing ICL top module, if the ICL top module exists. If the ICL top module does not exist, it returns the names of the scan interfaces that are created in the new ICL top module by ICL extraction after the add_icl_scan_interfaces command has been used to specify the scan interfaces.

You must set the current design with set_current_design before using the get_icl_scan_interface_list command.

Arguments

- -control_of *object_spec*

  An optional string that specifies an object_spec with at most one element, which is either an object of type icl_scan_interface or an object of type icl_scan_register.

  The -control_of option cannot be used in the ICL extraction flow when there is no top level ICL.

- -as_set

  An optional switch that instructs the tool to return the specifications done by means of the add_icl_scan_interfaces command, even if there is already an ICL module for the current design.

Return Values

When this command is used without the -control_of option, then if the ICL top module exists and the -as_set switch has been omitted, the command returns the ScanInterface names in the ICL top module. If the module does not have an explicit ScanInterface definition in the ICL module, but only an implicitly defined anonymous scan interface, the command returns “unnamed”. If the ICL top module does not exist or the -as_set switch has been used, the command returns the new ScanInterface names specified with the add_icl_scan_interfaces command, which are created during ICL extraction.

When the -control_of option is specified, the command returns a list of names (not a collection) of top level client scan interfaces, through which the given scan interface / the given scan register can be accessed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example returns the ScanInterface names PG1 and PG2 in the ICL module pass1:

```
Module pass1 {
   ScanInPort sin1;
   ScanInPort sin2;
   ScanOutPort sout;
   ShiftEnPort sen;
   CaptureEnPort cen;
   SelectPort select1;
   SelectPort select2;
   UpdateEnPort uen;
   ScanInterface PG1 {
        Port sin1; Port sout; Port sen; Port cen; Port select1; Port uen;
   }
   ScanInterface PG2 {
        Port sin2; Port sout; Port sen; Port cen; Port select2; Port uen;
   }
}
```

set_current_design pass1

get_icl_scan_interface_list

```
PG1 PG2
```

Related Topics

add_icl_scan_interfaces

delete_icl_scan_interfaces

set_icl_scan_interface_ports

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_scan_interface_port_list

Context: all contexts

Mode: all modes

Returns the names of the ports for the specified scan interface in an existing ICL top module, or the names of the ports for the specified scan interface that are created in the new ICL top module during ICL extraction.

Usage

get_icl_scan_interface_port_list **-name *name*** [-chain *chain_name*] [-as_set]

Description

Returns the names of the ports for the specified scan interface in an existing ICL top module, or the names of the ports for the specified scan interface that are created in the new ICL top module during ICL extraction.

You must set the current design with set_current_design before using the get_icl_scan_interface_port_list command.

Arguments

- **-name *name***

  A required switch and value pair that specifies the name of a scan interface, or a collection that contains exactly one element of type either “icl_scan_interface” or “icl_scan_interface_of_module” that specifies the scan interface, for which the list of ports is returned. If the module does not have an explicit ScanInterface definition in the ICL module, but only an implicitly defined anonymous scan interface, then use the string "unnamed" for the scan interface name to get the ports of the anonymous scan interface.

- -chain *chain_name*

  An optional switch and value pair that specifies the name of the scan chain for which the list of ports is returned. This switch is necessary for a scan interface that contains multiple scan chains that were specified in the -chains argument of the “add_icl_scan_interfaces” command.

- -as_set

  An optional switch that instructs the tool to return the specifications done by means of the set_icl_scan_interface_ports command, even if there is already an ICL module for the current design.

Return Values

If the ICL top module exists and the -as_set switch has been omitted, this command returns the names of the ports for the specified ScanInterface.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the ICL top module does not exist or the -as_set switch has been used, the command returns the names of the new ports for the ScanInterface specified with the set_icl_scan_interface_ports command, which are created during ICL extraction.

If a scan interface has multiple scan chains, calling the "get_icl_scan_interface_port_list" command without the -chain switch returns all ports apart from those of type scan_in or scan_out. When specifying the name of the scan chain with the -chain switch, only the ports of type scan_in and scan_out of the corresponding chain are returned.

Examples

Example 1

This example returns the list of ports for the ScanInterface named default in the ICL module pass1:

```
Module pass1{
   ScanInPort sin;
   ScanOutPort sout;
   SelectPort select;
   CaptureEnPort cen;
   ShiftEnPort shiftEn;
   UpdateEnPort updateEn;
   ScanInterface default {
      Port sin; Port sout; Port select; Port cen; Port shiftEn; Port
updateEn; DefaultLoadValue 3'b0;
   }
```

set_current_design pass1

get_icl_scan_interface_port_list -name default

```
sin sout select cen shiftEn updateEn
```

Example 2

This example returns the list of ports for the anonymous scan interface in the ICL module pass2:

```
Module pass2{
   ScanInPort sin;
   ScanOutPort sout;
   SelectPort select;
   CaptureEnPort cen;
   ShiftEnPort shiftEn;
   UpdateEnPort UpdateEn;
   }
```

set_current_design pass2

get_icl_scan_interface_port_list -name unnamed

```
sin sout select cen shiftEn updateEn
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

add_icl_scan_interfaces

delete_icl_scan_interfaces

get_icl_scan_interface_list

set_icl_scan_interface_ports

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_scan_mux

Context: patterns -silicon_insight -ijtag

Mode: setup

Returns various information about the specified scan mux node.

Usage

get_icl_scan_mux ***mux_name*** {-selected_input | -number_of_inputs | -selector}

Description

Returns various information about the specified scan mux node.

Use this command when manually debugging IJTAG networks as described in "Diagnosing and Debugging IJTAG Networks" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

Arguments

- ***mux_name***

  A required string that specifies the name of a scan mux.

- -selected_input

  An optional switch that returns the programmed selected input index for the specified scan mux.

- -number_of_inputs

  An optional switch that returns the number of input signals for the specified scan mux.

- -selector

  An optional switch that returns the selected signal(s) for the specified scan mux.

Examples

get_icl_scan_mux tap_I1.DRMux -selected_input

```
1
```

get_icl_scan_mux block3_I1.M1 -number_of_inputs

```
4
```

get_icl_scan_mux tap_I1.DRMux -selector

```
tap_I1.IR[1] tap_I1.IR[0]
```

Related Topics

analyze_icl_suspect

assert_icl_sib

create_icl_flush_pattern

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_icl_setup_pattern

deassert_icl_sib

get_icl_network

isolate_icl_node

report_icl_network

set_icl_network

set_icl_scan_mux

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_scope

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan" and, "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Introspects the current ICL scope.

Usage

get_icl_scope [-iCall | -iPrefix]

Description

When run without an option, this command returns the instance name prefix that is appended to the beginning of any ICL object names used in the iCall, iWrite, and iRead commands. When run with the -iCall option, this command ignores the current iPrefix specification and returns the ICL scope against which the current iProc is running. When run with the -iPrefix option, this command returns the current value specified for the iPrefix command.

For more information on the ICL scope, see the iCall command reference page.

Arguments

- -iCall

  An optional literal that specifies to ignore the current iPrefix specification and return the ICL scope against which the current iProc is running. This path is useful to obtain the exact ICL instance the iProc was run against using the iCall command. You can use this value in an iNote as shown in the example below.

- -iPrefix

  An optional literal that specifies to return the current value specified for the iPrefix command.

Return Values

An ICL instance path name.

Examples

In the following example, the iProc gets the ICL scope in which it was run against with the iCall command to echo into the retargeted pattern using an iNote command:

```
iProc myProc {} {
    iNote "Running on Instance [get_icl_scope -iCall]"
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

iCall

iPrefix

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_ssn_datapath_list

Context: all contexts

Mode: all modes

Returns the SSN datapath specifications added with the add_icl_ssn_datapaths, or returns the list of SSN datapaths in the ICL module of the current design.

Usage

get_icl_ssn_datapath_list [-types *type_list*] [-as_set]

Description

This command has different effects depending on the existence of an ICL module for the current design.

If there is no ICL for the current design, the command returns the specifications that have been added with the add_icl_ssn_datapaths command. This behavior can be enforced by using the -as_set switch even if there is ICL for the current design.

If there is ICL for the current design, the command returns the union of all elements in the tessent_ssn_datapath_ids attributes of the ICL ports of the ICL top module.

Arguments

- -types *type_list*

  An optional switch and Tcl list pair that specifies the type or types of datapath identifiers to include in the returned list. The <*type_list*> may include the following values:

  - inout
  - input
  - output

- -as_set

  A switch that instructs the tool to return the specifications that have been done by means of the add_icl_ssn_datapaths command even if there is ICL for the current design.

Examples

```
SETUP> add_icl_ssn_datapaths {A B C}
SETUP> delete_icl_ssn_datapaths B
SETUP> get_icl_ssn_datapath_list

A C
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_ssn_datapath_ports

Context: all contexts

Mode: all modes

Returns the SSN datapath specifications specified with the set_icl_ssn_datapath_ports command, or returns the collection of the SSN datapaths ports of a particular type in the ICL module of the current design.

Usage

get_icl_ssn_datapath_ports -name *datapath_id*
{-clock_inputs | -clock_outputs | -bus_data_inputs | -bus_data_outputs}
[-as_set |-object_type {icl | design}]

Description

This command has different effects depending on the existence of an ICL module for the current design.

If there is no ICL for the current design, the command returns the specifications that have been done by means of set_icl_ssn_datapath_ports. This behavior can be enforced by using the -as_set switch even if there is ICL for the current design.

If there is ICL for the current design, the command returns the ports of the specified type that are associated with the specified SSN datapath. If the ICL ports have the tessent_ssn_datapath_bus_index attribute, the ports are returned in the correct order; the port with the highest bus index will be the first in the collection, the port with the smallest bus index will be the last in the collection. The -object_type switch lets you specify the type of the elements in the result collection. Depending on the choice, the result is either a collection of objects of type “icl_port” or a collection of objects of type “port”.

Arguments

- -name *datapath_id*

  The name of the SSN datapath the tool returns.

- -clock_inputs

  A switch that instructs the tool to return the collection of the clock input ports of the specified SSN datapath.

- -clock_outputs

  A switch that instructs the tool to return the collection of the clock output ports of the specified SSN datapath.

- -bus_data_inputs

  A switch that instructs the tool to return the collection of the bus data input ports of the specified SSN datapath.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -bus_data_outputs

  A switch that instructs the tool to return the collection of the bus data output ports of the specified SSN datapath.

- -as_set

  A switch that instructs the tool to return the specifications that have been done by means of set_icl_ssn_datapath_ports even if there is ICL for the current design.

- -object_type icl | design

  A switch and literal pair that specifies the object type for the elements in the result collection. If "icl" is used, the result will be a collection of objects of type "icl_port". This is the default. If "design" is used, the result will be a collection of objects of type "port".

Examples

```
SETUP> add_icl_ssn_datapaths {A B}
SETUP> set_icl_ssn_datapath_ports -name A -clock_inputs bus_clock \
          -bus_data_inputs bus_a_in[1:0] -bus_data_outputs bus_a_out[1:0]
SETUP> set_icl_ssn_datapath_ports -name B -clock_inputs bus_clock \
          -bus_data_inputs bus_b_in[3:0] -bus_data_outputs bus_b_out[3:0]

SETUP> get_icl_ssn_datapath_ports -name A -clock_inputs

bus_clock

SETUP> get_icl_ssn_datapath_ports -name A -clock_outputs
SETUP> get_icl_ssn_datapath_ports -name A -bus_data_inputs

{bus_a_in[1]} {bus_a_in[0]}

SETUP> get_icl_ssn_datapath_ports -name A -bus_data_outputs

{bus_a_out[1]} {bus_a_out[0]}

SETUP> get_icl_ssn_datapath_ports -name B -clock_inputs

bus_clock

SETUP> get_icl_ssn_datapath_ports -name B -clock_outputs
SETUP> get_icl_ssn_datapath_ports -name B -bus_data_inputs

{bus_b_in[3]} {bus_b_in[2]} {bus_b_in[1]} {bus_b_in[0]}

SETUP> get_icl_ssn_datapath_ports -name B -bus_data_outputs

{bus_b_out[3]} {bus_b_out[2]} {bus_b_out[1]} {bus_b_out[0]}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_iclock_list

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Returns a list of ICL port and pin names on which you have issued an iClock command.

Usage

get_iclock_list [-overridden] [-within_current_pattern_set]

Description

Returns a list of ICL port and pin names on which you have issued an iClock command.

When you use the -overridden option, this command lists the ICL port and pin names on which you have issued an iClockOverride command on. You can use this list to introspect the effective source and the effective frequency multiplication and division values of the ICL pin or port using the get_iclock_option [-overridden] command.

Arguments

- -overridden

  Specifies to return the ICL port and pin names on which an iClockOverride command was issued as opposed to the iClock command when the option is not specified.

- -within_current_pattern_set

  Restricts the result of the command to the iClock/iClockOverride specifications that have been done within the currently opened pattern set.

Return Values

A Tcl list of ICL port and pin names.

Examples

In the following example, a Tcl list of ICL port and pin names are obtained using the get_iclock_list command. Their effective source and their effective frequency multiplication and division values are then echoed to the transcript

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set iclock_list [get_iclock_list]
foreach iclock $iclock_list {
    puts “iClock pin or port: $iclock”
    puts “ source = [get_iclock_option $iclock -source]”
    puts “ freq_multiplier= [get_iclock_option $iclock -freq_multiplier]”
    puts “ freq_divider = [get_iclock_option $iclock -freq_divider]”
}
```

Related Topics

add_clocks

delete_clocks

get_iclock_option

iClock

iClockOverride

report_clocks

report_icl_modules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_iclock_option

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Returns the effective or specified source, source type, frequency multiplier, frequency divider, clock period and other information related to the iClock or iClockOverride commands for the specified icl_port or pin_name.

Usage

get_iclock_option ***icl_port_or_pin_name*** [-overridden]
{**-source** | **-source_type** | **-freq_multiplier** | **-freq_divider** |
-period [-in {as | fs | ps | ns | us | ms | s}] | **-traversed_pins** | **-error_list** }

Description

Returns the effective or specified source, source type, frequency multiplier, frequency divider, clock period and other information related to the iClock or iClockOverride commands for the specified icl_port or pin_name.

The specified icl_port_or_pin_name must have been the target of an iClock command when the -overridden option is not specified; it must have been the target of an iClockOverride command when the -overridden option is specified.

Arguments

- ***icl_port_or_pin_name***

  A required string that specifies the name of an ICL port or pin object on which an iClock or iClockOverride command has been issued.

- -overridden

  An optional switch that specifies that icl_port or pin_name must be an icl_port or icl_pin object on which an iClockOverride command has been issued. If this switch is used, the return value of get_iclock_option is derived from the most recent matching iClockOverride command, instead of the most recent matching iClock command.

- **-source**

  A required switch that specifies to return the effective source of an iClock ICL pin or port, or the specified source of an iClockOverride ICL pin or port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-source_type**

  A required switch that specifies to return the type of the effective source of an iClock ICL pin or port, or the specified source of an iClockOverride ICL pin or port. The result is one of the following: “primary”, “tck”, “internal” or “none”. The following table shows the meaning of the different source types:

Table 5-5. Source Types

type	without -override	with -override
primary	The effective source of the iClock pin or port is a ClockPort of the current design.	The specified source of the iClockOverride pin or port is a ClockPort of the current design.
tck	The effective source of the iClock pin or port is a TCKPort of the current design.	The specified source of the iClockOverride pin or port is a TCKPort of the current design.
internal	The effective source of the iClock pin or port is a ToClockPort with a “Period” property.	The iClockOverride command for the pin or port has a “-period” specification instead of a “-source” specification, or the “-source” specification refers to an internal ICL pin.
none	The specified ICL pin or port has not been subject to a successfully completed iClock command.	The specified ICL pin or port has not been subject to a successfully completed iClockOverride command.

  The -source_type switch is special, in the sense that it can be used without error even if the specified ICL pin or port has not been subject to an iClock or iClockOverride command before. The return value “none” denotes this situation.

- **-freq_multiplier**

  A required switch that specifies to return the effective frequency multiplier value of an iClock ICL pin or port, or the specified frequency multiplier value of an iClockOverride ICL pin or port.

- **-freq_divider**

  A required switch that specifies to return the effective frequency divider value of an iClock ICL pin or port, or the specified frequency divider value of an iClockOverride ICL pin or port.

- **-period** [-in {as | fs | ps | ns | us | ms | s}]

  A required switch that specifies to return the effective clock period of an iClock ICL pin or port, or the specified clock period of an iClockOverride ICL pin or port.You can optionally

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

use the -in switch and keyword to specify a particular time unit. For iClock specifications, the default time unit is derived from the time unit of the source clock. This is either the time unit used in the -period switch of the add_clocks command (in case of an asynchronous source clock) or the time unit of the tester period (in case of a synchronous source clock). For iClockOverride specifications, the default time unit is derived from the unit that has been used during the processing of the iClockOverride command.

- **-traversed_pins**

  A required switch that specifies to return a dictionary of pins that have been traversed during the processing of the iClock command. When you use the -traversed_pins switch, you cannot also use the -overridden switch. The structure of the result is a dictionary of dictionaries as follows:

```
{   <pin_name_1>   {  node_type        port|pin
                      inversion        0|1
                      freq_multiplier  <int>
                      freq_divider     <int>
                   }

    <pin_name_2>   {  node_type        port|pin
                      inversion        0|1
                      freq_multiplier  <int>
                      freq_divider     <int>
                   }

...

    <pin_name_n>   {  node_type        port|pin
                      inversion        0|1
                      freq_multiplier  <int>
                      freq_divider     <int>
                   }
}
```

  The order of pins is from source to sink (iClock pin or port). Frequency multipliers, frequency dividers and inversions are accumulated appropriately.

- **-error_list**

  A required switch that specifies to return a list of error messages associated with the iClock ICL pin or port. If the -overridden switch is used, the result is empty, because iClockOverride error messages are currently not collected.

Return Values

The value of the specified option for the specified icl_port_or_pin_name.

Examples

In the following example, a Tcl list of ICL port and pin names are obtained using the get_iclock_list command. Their effective source and their effective frequency multiplication and division values are then echoed to the transcript

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set iclock_list [get_iclock_list]
foreach iclock $iclock_list {
    puts “iClock pin or port: $iclock”
    puts “ source = [get_iclock_option $iclock -source]”
    puts “ freq_multiplier= [get_iclock_option $iclock -freq_multiplier]”
    puts “ freq_divider = [get_iclock_option $iclock -freq_divider]”
}
```

Related Topics

add_clocks

delete_clocks

get_iclock_list

iClock

iClockOverride

report_clocks

report_icl_modules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_ijtag_instances

Context: dft, patterns

Mode: all modes

Returns a collection of instances for which there is a matched ICL module.

Usage

get_ijtag_instances [***ijtag_instance_spec***] [-below_instances *instance_objects*] [-filter *attribute_equation*] [-map_to_icl_modules] [-silent]

Description

Returns a collection of instances for which there is a matched ICL module.

You can restrict the search to instances below a given collection of instances and filter the collection based on an arbitrary filtering equation involving other attributes registered against or inherited by the instance object type.

The -silent option suppresses the warning that is normally generated if no Ijtag instances are returned.

Arguments

- ***ijtag_instance_spec***

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of instances. The string is a Tcl list of patterns separated by spaces and enclosed in braces ({)}. If no *name_patterns* are specified, the tool searches all Ijtag instances.

  If you specify the -below_instances options, the name patterns are hierarchical name patterns relative to the specified instance objects; otherwise the name patterns are relative to the current design.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for instances below the specified *instance_objects*. The specified *name_patterns* are searched relative to each instance found in the *instance_objects* list.

  If the specified *instance_spec* is an empty collection, all instances below the current design are considered. See the description of the -below_instances switch of the get_instances command description for the justification of this behavior.

- -filter *attribute_equation*

  An optional switch and string that specify to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the instance object type. See section "Attribute Filtering Equation Syntax" on page 3832 for more details on filtering using the attribute equation format.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -map_to_icl_modules

  An optional switch that specifies to return only the collection of the ICL modules for the instances that are returned by the command when you do not specify this switch.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

  Note

  You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

Examples

The following example uses the get_ijtag_instances command to get all instances of a module below a given instance for which there is a matched ICL file.

get_ijtag_instances -below_instance subblockA_i1

```
{subblockA_i1/tdr1 subblockA_i1/tdr2}
```

Related Topics

get_ijtag_instance_option

report_ijtag_instances

set_ijtag_instance_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_ijtag_instance_option

Context: dft, patterns

Mode: all modes

Returns the value of the selected option for the specified Ijtag instance.

Usage

get_ijtag_instance_option ***ijtag_instance_spec*** [-use_in_dft_specification] [-connect_data_signals_already_connected_to_ports]

Description

Returns the value of the selected option for the specified Ijtag instance.

Arguments

- ***ijtag_instance_spec***

  A required string that is the name of an IJTAG instance or a collection containing one IJTAG instance as reported by get_instances or get_ijtag_instances.

  The instance object must have an ICL module matched to it.

  Tip

  If your ***ijtag_instance_spec*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- -use_in_dft_specification

  A Boolean option that specifies that the value of the -use_in_dft_specification option is required. This value is set using the set_ijtag_instance_options and is used by the create_dft_specification command.

- -connect_data_signals_already_connected_to_ports

  A Boolean option that specifies that the value of the -connect_data_signals_already_connected_to_ports option is required. This value is set using the set_ijtag_instance_options and is used by the create_dft_specification command.

Examples

The following example returns the value of the use_in_dft_specification option of all Ijtag instances.

```
foreach_in_collection inst [get_ijtag_instances] {
   set opt [get_ijtag_instance_option $inst -use_in_dft_spec]
   puts "inst = [get_single_name $inst], use_in_dft_spec = $opt"
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
inst = u1/trd1, use_in_dft_spec = auto
inst = u1/trd2, use_in_dft_spec = auto
inst = u2/trd1, use_in_dft_spec = off
```

Related Topics

get_ijtag_instances

report_ijtag_instances

set_ijtag_instance_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_ijtag_retargeting_options

Context: all contexts

Mode: all modes

Returns the value of the specified option of the previous set_ijtag_retargeting_options command or its default value if the set_ijtag_retargeting_options command was not run.

Usage

get_ijtag_retargeting_options ***getOption*** [-is_specified]

Description

Before using this command, you must load and elaborate the top-level ICL module. Otherwise, the tool prints an error message.

Arguments

- ***getOption***

  A required switch that can take one of the following values:

  -test_setup_network_end_state — A switch that returns the value of the -test_setup_network_end_state option (keep or reset) specified by the previous set_ijtag_retargeting_options command or returns the default value (reset) if the set_ijtag_retargeting_options command was not run.

  -test_setup_ireset — A switch that returns the value of the -test_setup_ireset option (off, first_icall or when_icl_present) specified by the previous set_ijtag_retargeting_options command or returns the default value (when_icl_present) if the set_ijtag_retargeting_options command was not run.

  -iapply_target_annotations — A switch that returns the value of the -iapply_target_annotations option (off, dense, or full) specified by the previous set_ijtag_retargeting_options command or returns the default value (dense) if the set_ijtag_retargeting_options command was not run.

  -max_operations_per_iapply — A switch that returns the value of the -max_operations_per_iapply option specified by the previous set_ijtag_retargeting_options command or returns the default value (64) if the set_ijtag_retargeting_options command was not run.

  -compare_constant_capture_values — A switch that returns the value of the -compare_constant_capture_values option (on or off) specified by the previous set_ijtag_retargeting_options command or returns the default value (on) if the set_ijtag_retargeting_options command was not run.

  -compare_constant_data_out_ports — A switch that returns the value of the -compare_constant_data_out_ports option for the primary DataOut ports (on or off) specified by the previous set_ijtag_retargeting_options command or returns the default value (off) if the set_ijtag_retargeting_options command was not run.

  -ignore_non_reset_end_state_implicit_compares — A switch that returns the value of the -ignore_non_reset_end_state_implicit_compares option (off, auto or on) specified

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

by the previous set_ijtag_retargeting_options command or returns the default value (auto) if the set_ijtag_retargeting_options command was not run.

-constrain_update_en_only — A switch that returns the value of the -constrain_capture_en_only option (auto or on) specified by the previous set_ijtag_retargeting_options command or returns the default value (auto) if the set_ijtag_retargeting_options command was not run.

-default_scan_out_strobe_point — A switch that returns the value of the -default_scan_out_strobe_point option (before_rising_edge_of_tck, before_falling_edge_of_tck, or auto) specified by the previous set_ijtag_retargeting_options command or returns the default value (auto) if the set_ijtag_retargeting_options command was not run.

-interface_type — A switch that returns the value of the -interface_type option (auto, ijtag or tpsp) specified by the previous set_ijtag_retargeting_options command or returns the default value (auto) if the set_ijtag_retargeting_options command was not run.

-merge_irunloop_only — A switch that returns the value of the -merge_irunloop_only option (on or off) specified by the previous set_ijtag_retargeting_options command or returns the default value (off) if the set_ijtag_retargeting_options command was not run.

-network_end_state_warnings — A switch that returns the value of the -network_end_state_warnings option (on or off) specified by the previous set_ijtag_retargeting_options command or returns the default value (off) if the set_ijtag_retargeting_options command was not run.

-disable_pio_operations — A switch that returns the value of the -disable_pio_operations option (on or off) specified by the previous set_ijtag_retargeting_options command or returns the default value (off) if the set_ijtag_retargeting_options command was not run.

-ignore_iwrite_when_value_already_present — A switch that returns the value of the -ignore_iwrite_when_value_already_present option (on or off) specified by the previous set_ijtag_retargeting_options command or returns the default value (off) if the set_ijtag_retargeting_options command was not run.

-allow_skip_rti — A switch that returns the value of the -allow_skip_rti option (on or off) specified by the previous set_ijtag_retargeting_options command or returns the default value (on) if the set_ijtag_retargeting_options command was not run.

-generated_timeplate_name — A switch that returns the timeplate base name value from the -generated_timeplate_name option specified by the previous set_ijtag_retargeting_options command or returns the default value if the set_ijtag_retargeting_options command was not run.

-inject_cycles — A switch that returns the value of the -inject_cycles option (Tcl list with an even number of arguments, each comprising a TAP state and the related number of injected cycles in alternating order) specified by the previous set_ijtag_retargeting_options command or returns the default value if the set_ijtag_retargeting_options command was not run.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

-tck_ratio — A switch that returns the value of the -tck_ratio option ("auto" or a power of 2) specified by the previous set_ijtag_retargeting_options command or returns the default value (auto) if the set_ijtag_retargeting_options command was not run.

The value of tck_ratio might depend on the maximum tck clock frequency that can be applied to the circuit. See the "IJTAG Network Performance Optimization" section in the *Tessent IJTAG User's Manual* showing how to maximize the frequency of the IJTAG network test clock.

-tck_period — A switch that returns the value of the -tck_period option specified by the previous set_ijtag_retargeting_options command or returns the default value (100ns) if the set_ijtag_retargeting_options command was not run.

-annotation_parameter_values — A switch that returns the value of the -annotation_parameter_values option (Tcl list with an even number of arguments, each comprising a key and a value in alternating order, describing the configuration of notes, comments and annotations in Tessent IJTAG) specified by the previous set_ijtag_retargeting_options command. If the set_ijtag_retargeting_options command was not run, then the tool returns the default value.

- -is_specified

  An optional switch that returns true if the retargeting option specified by the ***getOption*** switch was set by a previous set_ijtag_retargeting_options command.

Examples

The following example loads the ICL, turns off the reset of the ICL network after the last test_setup PDL command and then displays the value of the selected option:

```
SETUP> set_context patterns -ijtag -no_rtl
SETUP> read_icl ../data/chip.icl
SETUP> set_current_design

//  Note: Top design is 'chip'.

SETUP> set_ijtag_retargeting_options -test_setup_network_end_state keep
SETUP> get_ijtag_retargeting_options -test_setup_network_end_state

keep
```

Related Topics

report_ijtag_retargeting_options

set_ijtag_retargeting_options

reset_ijtag_retargeting_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_input_constraints

Context: dft, patterns

Mode: setup, analysis

Returns the present constraint status of primary input ports or pseudo ports.

Usage

get_input_constraints ***obj_spec*** { -equivalent_ports | -equivalent_inverted_ports | -constraint_value | -hold | -no_z | -slow_pad }

Description

The get_input_constraints command displays the constraint status of primary input pins to which you have applied constraints using the add_input_constraints command.

Arguments

- ***obj_spec***

  A required value that specifies a collection of ports and pseudo ports.

- -equivalent_ports

  An optional switch that specifies to return the ports/pseudo ports that are defined as the non-inverted equivalent of the specified port/pseudo port. The ***obj_spec*** collection must contain exactly one port or pseudo port object.

- -equivalent_inverted_ports

  An optional switch that specifies to return the ports/pseudo ports that are defined as the inverted equivalent of the specified port/pseudo port. The ***obj_spec*** collection must contain exactly one port or pseudo port object.

- -constraint_value

  An optional switch that specifies to return the constraint values specified for the given ports/pseudo ports. The returned constraint values can be any of the following:

  C0 — Constant 0

  C1 — Constant 1

  CX — Constant X (unknown)

  CZ — Constant Z (high-impedance)

  CR0 — Constant that returns to 0

  CR1 — Constant that returns to 1

  CT0 — Constant TIE0

  CT1 — Constant TIE1

  CTZ — Constant TIEZ

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The values are returned as a Tcl list in the same order as the port and pseudo port objects specified with ***obj_spec***. The returned Tcl list length equals the number of ports/pseudo ports specified with ***obj_spec***. For the specified ports and pseudo ports that do not have a constraint defined, an empty Tcl list is returned.

- -hold

  An optional switch that specifies to return a Tcl list of 0's and 1's depending on whether -hold was specified for the given port/pseudo ports. Default is 0. The values are returned as a Tcl list in the same order as the port and pseudo port objects specified with ***obj_spec***. The returned Tcl list length equals the number of ports/pseudo ports specified with ***obj_spec***. For the specified ports and pseudo ports that do not have a constraint defined, an empty Tcl list is returned.

- -no_z

  An optional switch that specifies to return a Tcl list of 0's and 1's depending on whether -no_z was specified for the given port/pseudo ports. Default is 0. The values are returned as a Tcl list in the same order as the port and pseudo port objects specified with ***obj_spec***. The returned Tcl list length equals the number of ports/pseudo ports specified with ***obj_spec***. For the specified ports and pseudo ports that do not have a constraint defined, an empty Tcl list is returned.

- -slow_pad

  An optional switch that specifies to return a Tcl list of 0's and 1's depending on whether -slow_pad was specified for the given port/pseudo ports. Default is 0. The values are returned as a Tcl list in the same order as the port and pseudo port objects specified with ***obj_spec***. The returned Tcl list length equals the number of ports/pseudo ports specified with ***obj_spec***. For the specified ports and pseudo ports that do not have a constraint defined, an empty Tcl list is returned.

Examples

Example 1

The following example adds constraints with the add_input_constraints command, then retrieves them with the report_input_constraints and get_input_constraints commands.

SETUP> add_input_constraints -equivalent A[2] B[1] B[0] -invert A[0] A[1]

SETUP> report_input_constraints

```
//  Primary Input  Equivalent  C-Type  Hold      No-Z      Slow-Pad
//  -------------  ----------  ------  ----  ------------  --------
//  A[2]                        NONE    No   Undetermined     No
 /                 ~A[1]        NONE    No   Undetermined     No
//                 ~A[0]        NONE    No   Undetermined     No
//                  B[0]        NONE    No   Undetermined     No
//                  B[1]        NONE    No   Undetermined     No
```

SETUP> get_input_constraints A[0] -equivalent_ports

```
{A[1]}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SETUP> get_input_constraints A[1] -equivalent_ports

```
{A[0]}
```

SETUP> get_input_constraints A[2] -equivalent_ports

```
{B[0] B[1]}
```

SETUP> get_input_constraints A[0] -equivalent_inverted_ports

```
{A[2] B[0] B[1]}
```

SETUP> get_input_constraints A[2] -equivalent_inverted_ports

```
{A[0] A[1]}
```

SETUP> get_input_constraints A[2] -hold

```
0
```

SETUP> add_input_constraints A[4] -C1
SETUP> get_input_constraints A[12:0] -constraint_value

```
{} {} {} {} {} {} {} {} C1 {} {} {} {}
```

SETUP> get_input_constraints A[2] -no_z

```
0
```

SETUP> add_input_constraints A[2] -slow_pad
SETUP> get_input_constraints A[2] -slow_pad

```
1
```

SETUP> get_input_constraints {A[0] A[1] A[2]} -slow_pad

```
0 0 1
```

SETUP> add_input_constraints B[1] -CT1 -no_z
SETUP> get_input_constraints B[1] -constraint_value

```
CT1
```

SETUP> get_input_constraints B[1] -no_z

```
1
```

Example 2

The following examples check if a constraint is defined:

```
set conValue [get_input_constraints xyz/n_123 -constraint_value]
if { [llength [lindex $conValue 0]] != 0 } {
   puts "Constraint is defined. "
} else {
   puts "Constraint not defined"
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set ports [get_ports]
set constraints [get_input_constraints $ports -constraint_value]
for {set i 0 } { $i < [llength $constraints] } { incr i }  {
 if { [llength [lindex $constraints $i]] != 0 } {
  puts "Constraint is defined for port [get_name_list [index_collection $ports $i]]"
   } else {
  puts "Constraint not defined for port [get_name_list [index_collection $ports $i]]"
   }
}
```

Related Topics

add_input_constraints

delete_input_constraints

report_input_constraints

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_insertion_option

Context: unspecified, all contexts

Mode: all modes

Returns the value of any insertion option that is specifiable with the set_insertion_options command.

Usage

```
get_insertion_option -auto_uniquify_edited_modules
    | -edited_module_prefix
    | -net_uniquification_suffix
    | -module_uniquification_suffix
    | -instance_uniquification_suffix
    | -allow_assigns | -use_buffer_cell
    | -inserted_object_comment_label
    | -treat_common_ancestor_as_unique
    | -preserve_paths_for_unrolled_v2001_loops
    | -vhdl_port_pin_net_type
    | -open_input_pins
    | -remove_inactive_sections_in_unrolled_generate_loops
    | -trace_across_module_boundaries_to_find_functional_drivers
    | -undriven_created_input_message_severity
    | -create_unique_net_names_in_module
    | -gen_loop_net_size_threshold
    | -edited_file_prefix
```

Description

Returns the value of any insertion option that is specifiable with the set_insertion_options command.

Only one option can be specified per get_insertion_option invocation. The return value is a string containing the current setting of the specified option. An error is generated if more than one option is specified, and the return value is null.

Arguments

- -auto_uniquify_edited_modules

  A switch that returns the current setting of -auto_uniquify_edited_modules option of the set_insertion_options command.

- -edited_module_prefix

  A switch that returns the current setting of the -edited_module_prefix option of the set_insertion_options command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -net_uniquification_suffix

  A switch that returns the current setting of the -net_uniquification_suffix option of the set_insertion_options command.

- -module_uniquification_suffix

  A switch that returns the current setting of the -module_uniquification_suffix option of the set_insertion_options command.

- -instance_uniquification_suffix

  A switch that returns the current setting of the -instance_uniquification_suffix option of the set_insertion_options command.

- -allow_assigns

  A switch that returns the current setting of the -allow_assigns option of the set_insertion_options command.

- -use_buffer_cell

  A switch that returns the current setting of the -use_buffer_cell option of the set_insertion_options command.

- -inserted_object_comment_label

  A switch that returns the current setting of the -inserted_object_comment_label option of the set_insertion_options command.

- -treat_common_ancestor_as_unique

  A switch that returns the current setting of the -treat_common_ancestor_as_unique option of the set_insertion_options command.

- -preserve_paths_for_unrolled_v2001_loops

  A switch that returns the current setting of the -preserve_paths_for_unrolled_v2001_loops option of the set_insertion_options command.

- -vhdl_port_pin_net_type

  A switch that returns the current setting of the -vhdl_port_pin_net_type of the set_insertion_options command.

- -open_input_pins

  A switch that returns the current setting of the -open_input_pins option of the set_insertion_options command.

- -remove_inactive_sections_in_unrolled_generate_loops

  A switch that returns the current setting of the -remove_inactive_sections_in_unrolled_generate_loops option of the set_insertion_options command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -trace_across_module_boundaries_to_find_functional_drivers

  A switch that returns the current setting of the -trace_across_module_boundaries_to_find_functional_drivers option of the set_insertion_options command.

- -undriven_created_input_message_severity

  A switch that returns the current setting of the -undriven_created_input_message_severity option of the set_insertion_options command.

- -create_unique_net_names_in_module

  A switch that returns the current setting of the -create_unique_net_names_in_module option of the set_insertion_options command.

- -gen_loop_net_size_threshold

  A switch that returns the current setting of the -gen_loop_net_size_threshold option of the set_insertion_options command.

- -edited_file_prefix

  A switch that returns the current setting of the -edited_file_prefix option of the set_insertion_options command.

Examples

The following example introspects the current setting of the -edited_module_prefix option of the set_insertion_options command.

get_insertion_option -edited_module_prefix

```
MyCore_
```

Related Topics

set_insertion_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_insert_test_logic_option

Context: dft -scan, dft -test_points

Mode: setup, analysis, insertion

Retrieves a prefix/infix value that is used when logic is created by the insert_test_logic command.

Usage

get_insert_test_logic_option [-inserted_object_naming] | [-inserted_object_prefix] |
[-persistent_cell_prefix] | [-persistent_clock_cell_prefix] |
[-logic_type [generic | clock_gater_fix | clock_control_fix | dedicated_input_wrapper | dedicated_input_holding_wrapper | dedicated_output_wrapper | lockup | pipeline | ram_fix | testpoint_control | testpoint_observation | tri_state_fix | x_bounding]] |
[-reuse_modules_when_possible] [-generate_third_party_script]

Description

The get_insert_test_logic_option command retrieves a prefix/infix value that is used when logic is created by the insert_test_logic command.

Arguments

- -inserted_object_naming

  An optional switch that returns the default naming style of the instances and associated nets created during “insert_test_logic”.

- -inserted_object_prefix

  An optional switch that returns the default prefix of HDL objects created during “insert_test_logic”.

- -persistent_cell_prefix

  An optional switch that returns the default prefix of persistent cell instances created during “insert_test_logic”.

- -persistent_clock_cell_prefix

  An optional switch that returns the default prefix of persistent clock cell instances created during “insert_test_logic”.

- -logic_type

  An optional switch that returns the default infix of HDL objects created during “insert_test_logic” for a specified logic type. The possible logic types are as follows:

  generic — Used for generic logic.

  clock_gater_fix — Used for clock gater control logic.

  clock_control_fix — Used for clock control logic.

  dedicated_input_wrapper — Used for dedicated input wrapper logic.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dedicated_input_holding_wrapper — Used for dedicated input holding wrapper logic.

dedicated_output_wrapper — Used for dedicated output wrapper logic.

lockup — Used for lockup logic.

pipeline — Used for pipeline logic.

ram_fix — Used for ram control logic.

testpoint_control — Used for control testpoint logic.

testpoint_observation — Used for observation testpoint logic.

tri_state_fix — Used for tristate control logic.

x_bounding — Used for X-bounding logic

- -reuse_modules_when_possible

  An optional switch that returns the default uniquification of new HDL modules for each instance created during “insert_test_logic”.

- -generate_third_party_script

  An optional switch that returns the setting of third-party script generation specified with the set_insert_test_logic command.

Examples

The following example demonstrates the use of the get_insert_test_logic_option command.

```
SETUP> get_insert_test_logic_option -logic_type  dedicated_input_wrapper

diw_

SETUP> set_insert_test_logic_options -logic_type  dedicated_input_wrapper \
            -logic_infix inreg_
SETUP> get_insert_test_logic_option -logic_type  dedicated_input_wrapper

inreg

SETUP> report_insert_test_logic_options
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  'insert_test_logic' Command Option           Current Value
//  -------------------------------------------  ------------------------
//  -inserted_object_naming                      standard
//  -inserted_object_prefix                      ts_
//  -logic_type clock_control_fix                clkfix_
//  -logic_type clock_gater_fix                  cgfix_
//  -logic_type dedicated_input_holding_wrapper  dihw_
//  -logic_type dedicated_input_wrapper          inreg_
//  -logic_type dedicated_output_wrapper         dow_
//  -logic_type generic                          logic_
//  -logic_type lockup                           lockup_
//  -logic_type pipeline                         pipeline_
//  -logic_type ram_fix                          ramfix_
//  -logic_type testpoint_control                cp_
//  -logic_type testpoint_observation            op_
//  -logic_type tri_state_fix                    trifix_
//  -logic_type x_bounding                       xb_
//  -persistent_cell_prefix                      tessent_persistent_cell_
//  -persistent_clock_cell_prefix                tessent_persistent_cell_
//  -reuse_modules_when_possible                 off
```

Related Topics

report_insert_test_logic_options

set_insert_test_logic_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_instance_polygons

Context: patterns -scan_diagnosis

Mode: analysis

Returns Tcl list of the x, y coordinate location of the specified instance in the LDB.

Usage

get_instance_polygons ***instance_path*** [-ldb_units]

Description

This command returns the x and y coordinate location corresponding to a specified instance path in the LDB. The tool returns chip-level coordinates.

By default, the command returns coordinates expressed in microns units. You can override the default by specifying the -ldb_units switch.

Arguments

- instance_path

  Required string that specifies the path to a polygon instance within the LDB.

- -ldb_units

  An optional switch that returns the x, y coordinates in LDB units rather than the default micron units.

Related Topics

get_xy_instance

report_instance_polygons

report_net_polygons

report_xy_instance

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_instances

Context: all contexts

Mode: all modes

Returns a collection of all instances instantiated relative to the current design that match the specified *name_patterns* list, subject to filtering imposed by the specified options.

Usage

get_instances [*name_patterns*]
[-of_modules *module_objects*]
[-parent_of_instances *instance_objects*
| -below_instances *instance_objects*
| -of_pins *pin_objects*
| -of_nets *net_objects*
| -of_gate_pins *gate_pin_objects*
| -of_icl_instances *icl_instance_objects*]
[-of_type *types*]
[-hierarchical]
[-match_rtl_reg | -use_path_matching_options]
[-filter *attribute_equation*]
[-regexp]
[-nocase]
[-silent]

Description

Returns a collection of all instances instantiated relative to the current design that match the specified *name_patterns* list, subject to filtering imposed by the specified options.

When the *name_patterns* list is omitted, all instances are returned subject to filtering imposed by the specified options.

By default, the tool reports an error if the get_instances command returns no result unless you have set the -silent switch.

You can change no result Tcl errors to warnings by issuing the following command:

set_tcl_shell_options -change_no_result_warnings_to_errors off

Note

This approach introduces a risk if your scripts cannot deal with empty collections correctly. As this may result in incorrect/unintended behavior, it is not recommended.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of instances. The string is a Tcl list of patterns separated by spaces and enclosed in braces ({}). If no *name_patterns* are specified, the tool searches all instances.

  After quick synthesis, you can specify the exact RTL paths here, and the tool finds the equivalent instance objects. This enables you to load a DftSpecification with RTL instance paths if the current views are quick synthesized.

  If you specify one of the optional switches (-parent_of_instances, -of_pins, -of_nets, -of_gate_pins or -hierarchical), the name patterns are matched against the *leaf_name* attribute value of the instances in which case no hierarchy separators are permitted in the pattern list.

  If you specify the -below_instances option, the name patterns are hierarchical name patterns relative to the specified instance objects otherwise the name patterns are relative to the current design.

  See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a complete description of the pattern syntax when the -regexp option is used and not used.

  Tip

  If your *name_patterns* string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- -of_modules *module_objects*

  An optional switch and value pair that constrains the command to return only instances of modules specified by *module_objects*. The *module_objects* is a single module name, a Tcl list of module names, or a collection of module objects as returned by the get_modules command.

- -parent_of_instances *instance_objects*

  An optional switch and value pair that constrains the command to return a collection of instance objects that are the parent instances of the specified *instance_objects*. The returned collection is filtered to only return a given parent instance once. When this option is specified, the optional *name_patterns* option is a string that is matched against the leaf instance name of every parent instance of the collection. An error is generated if the specified *name_patterns* contain more than one level of hierarchy when the -parent_of_instances option is used.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for instances below the specified *instance_objects*. The tool searches the specified *name_patterns* relative to each instance found in the *instance_objects* list. If the *instance_objects* is a null collection or a null string, the switch is ignored and instances are searched relative to the current design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

An exception is made for this switch to treat an empty collection as meaning the current design such that it could accept the return value of "get_instances -parent_of_instance XXX" even if XXX is directly instantiated into the current design. Very often, you want to create an instance or find objects in the same parent instance of another instance. If the -below_instances switch did not accept a null collection as meaning the current design, you could not use this simple form:

```
get_instances -below_instance [get_instances -parent_of_instance XXX]
```

You would always have to make an exception for instances found directly below the current design as follows:

```
set parent_instance [get_instances -parent_of_instance XXX]
if {[sizeof_collection $parent_instance] > 0} {
  get_instances
} else {
  get_instances -below_instances $parent_instance
}
```

Because an empty collection is interpreted as meaning the current design when passed to the -below_instances switch, you do not need the above exception code handling to handle instances directly instantiated in the current design differently from those instantiated at least two levels below the current design.

- -of_pins *pin_objects*

  An optional switch and value pair that constrains the command to return the instance on which the pins belong to. The returned collection is filtered to only return a given instance once even if it belongs to many pins in the *pin_objects* list. When this option is specified, the optional *name_patterns* is a string that is matched against the leaf instance name of the instances. An error is generated if the specified *name_patterns* contain more than one level of hierarchy when the -of_pins option is used.

- -of_nets *net_objects*

  An optional switch and value pair that constrains the command to return the instance in which the net objects are located. The returned collection is filtered to only return a given instance once even if it is the parent instance of many nets in the *net_objects* list. When this option is specified, the optional *name_patterns* is a string that is matched against the leaf instance name of the instances. An error is generated if the specified *name_patterns* contain more than one level of hierarchy when the -of_nets option is used.

- -of_gate_pins *gate_ pin_objects*

  An optional switch and value pair that constrains the command to return the collection of instances to which the gates associated with the gate pins belong. The returned collection is filtered to only return a given instance once even if it belongs to many gate pins in the *gate_pin_objects* list. When this option is specified, the optional *name_patterns* is a string that is matched against the leaf instance name of the instances. An error is generated if the specified *name_patterns* contain more than one level of hierarchy when the -of_gate_pins option is used.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the specified gate_pin exists under one or many levels of `celldefine module instances, the highest-level `celldefine module instance is returned. In addition, gate_pins directly under the root module return an empty (“”) string as the instance.

- -of_icl_instances *icl_instance_objects*

  An optional switch and value pair that constrains the command to return the instances corresponding to the specified *icl_instance_objects*. The *icl_instances_objects* can be retrieved using the get_icl_instances command. If the tool is successful, it returns a collection of these instances. If not, it reports an error unless the -silent switch is specified.

- -of_type *types*

  An optional switch and value pair that constrains the tool to return only instances whose type attribute is one of the following: design, cell, or primitive. The types value is either a single type string or a Tcl list of type strings such as {design cell}.

- -hierarchical

  An optional string that specifies to perform pattern matching relative to all instances found below the current design or below the specified *instance_objects* when the -below_of_instances option is specified. When this option is specified, the *name_patterns* argument cannot include slashes, which are hierarchy separator symbols.

- -match_rtl_reg

  An optional switch that performs register name-mapping using the rules you specified for RTL to post-synthesis/post-layout name mapping. Using this option enables you to find the register with the register mapping rules. See add_rtl_to_gates_mapping for complete information.

  Note

  When you use this switch, the *name_patterns* list is mandatory and cannot include wildcards. The -regexp and -nocase switches are ignored.

- -use_path_matching_options

  An optional switch that performs instance name-mapping using rules you specified for RTL to post-synthesis/post-layout name mapping. Using this option enables you to find an instance using the instance mapping rules. See add_rtl_to_gates_mapping for complete information.

- -filter *attribute_equation*

  An optional switch and string that specify to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the instance object type. See section “Attribute Filtering Equation Syntax” on page 3832 for more details on filtering attribute equation format.

- -regexp

  An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a real regular expression instead of as a simple wildcard pattern. You must properly

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

escape the bracket "[]" symbols so that they are interpreted as part of the regular expression and not interpreted by the Tcl shell. See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for instances matching the *name_patterns*.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

Note

You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

Examples

Example 1

The following example returns a collection of all instances of cell types that start with ABC found anywhere below the current design.

get_instances ABC* -hierarchical -of_type cell

```
{u2/ABCD u1/u5/ABCDEF}
```

Example 2

The following example starts with the collection of all instances and filters it based on an attribute_equation of user-defined attributes.

get_instances -filter {myType1 || myType2 == "ABC"}

```
{u2 u1/u5}
```

Example 3

The following example starts with the collection of all instances and filters it based on the expression specified by the attribute_equation string.

get_instances -filter {name=~"u4*/u1*"}

```
{u4/u1 u4/u5/u1}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

In the following example, the collection of all instances matching a list of hierarchical name patterns are additionally filtered to only return instances of type design. The names u1/* and u*/u* are searched relative to the current design.

get_instances {u1/* u*/u*} -of_type design

```
{u4/u5}
```

Example 5

In the following example, the collection of all instances matching a leaf name pattern is returned. The leaf instance name u3* searches for all instances found below the current design.

get_instances u3* -hierarchical

```
{u3 u4/u3 u4/u5/u3 u5/u3}
```

Example 6

In the following example, the collection of all instances matching a leaf name pattern is returned. The leaf instance name u3* is matched against instances found below instance u4.

get_instances u3* -hierarchical -below_instances u4

```
{u4/u3 u4/u5/u3}
```

Example 7

In the following example, the -parent_of_instances option is used to find the parent instance of each instance on which a collection of pins exist. Only the parent instances whose leaf instance names match ABC* are returned. As described in the description of the -parent_of_instances switch, the optional name_pattern argument, in this case ABC*, is a string that is matched against the leaf instance name of every parent instance of the collection when the -parent_of_instances option is used.

set pin [get_pins]
get_instances ABC* -parent_of_instances [get_instances -of_pins $pin]

```
{u4/ABC1 u4/u5/ABC2}
```

Alternatively, you could use the base_name attribute of the pin objects and the parent_instance attribute of the instances to achieve the same result. However, the method used in this example is much more efficient because it deals with objects instead of strings containing object names. This method also avoids issues with string concatenation in the presence of escaped identifiers.

Example 8

This example returns all design instances that correspond to all ICL instances. It uses the get_icl_instances command to retrieve the ICL instances.

get_instances -of_icl_instances [get_icl_instances]

```
//  { block1_I1/raw1_I1 block1_I1/raw1_I2 block1_I1/sib1_I1 block1_I1/sib1_I2 …}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 9

This example returns the equivalent instance object for the given RTL name. Generate blocks (generate for/if/case) are flattened after quick synthesis (the generate blocks are removed and the hierarchical names are added to the instance base names). For example, “B1[1].B2.u1” becomes “\B1[1].B2.u1 “, and “\B1[1] .B2.u1” also becomes “\B1[1].B2.u1 “.

get_instances B1[1].B2.u1

```
{\B1[1].B2.u1 }
```

get_instances {{B1[1].B2.u1 }}

```
{\B1[1].B2.u1 }
```

get_instances {{B1[1] .B2.u1}}

```
{\B1[1].B2.u1 }
```

get_instances {{B1[1].B2.u1 }}

```
{\B1[1].B2.u1 }
```

Related Topics

add_rtl_to_gates_mapping

create_instance

delete_instances

get_common_parent_instance

rename_instance

replace_instances

report_rtl_to_gates_mapping

uniquify_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_instrument_dictionary

Context: all contexts including the unspecified one

Mode: all modes

Returns the Tcl dictionary created by the instrument processed by process_dft_specification or process_patterns_specification.

Usage

get_instrument_dictionary [*dictionary_name* [{*key_name*}...]] [-list] [-silent | -exists]

Description

Returns the content of a Tcl dictionary entry or the list of available dictionaries.

Use the -list option to see the currently available dictionary. At startup, the only dictionary available is tshell_global. Once you have run the process_dft_specification command, other dictionaries become available based on the instrument types that were processed.

The content of the dictionary is useful when you are writing customization scripts as described in the "process_dft_specification.post_insertion Procedure" on page 1452. For example, to know the extension of the RTL files that were created and stored in the instruments directory, use:

```
get_instrument_dictionary DftSpecification rtl_extension
```

For Tessent MissionMode in-system test, this command returns a dictionary named "mentor::in_system_test::PatternsSpecification" when you run the process_patterns_specification command to generate in-system test patterns. You can use the dictionary to help you program the CPU as described in "Guidelines for Programming the CPU for In-System Test" in the *Tessent MissionMode User's Manual*.

See the "Instrument Dictionary" section for the description of the dictionary created by the boundary scan instrument.

Arguments

- *dictionary_name* [{*key_name*}...]

  An optional dictionary name followed by any number of keys from that dictionary. If no keys are specified, then the entire dictionary is returned.

- ***-list***

  An optional Boolean switch that request the command to return the list of available instrument dictionaries. At startup, the only dictionary available is tshell_global. Once you have run the process_dft_specification command, other dictionaries become available based on the instrument types that were processed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ***-silent***

  An optional Boolean switch that suppresses the error that is normally produced when a dictionary or key is requested and the dictionary or key does not exist. With this switch, no errors is generated and the command returns a null value.

- ***-exists***

  An optional Boolean switch that request the command to return 1 if the specified dictionary and key combination exist and returns 0 otherwise.

Examples

Example 1

Suppose you want to know the extension of the interface module created in the dft_inserted_designs directory when the current design is a Verilog module read in with format sv2005. You can query the dictionary using this command:

get_instrument_dictionary tshell_global Format2ExtensionMapping verilog sv2005

The command returns “sv05” and the interface module has the extension sv05_interface. To see the entire mapping list, use the following command:

format_dictionary [get_instrument_dictionary tshell_global Format2ExtensionMapping]

which returns the complete sub-dictionary.

The following example uses the -list switch to see the available dictionaries. It then shows how to display the entire “tshell_global” dictionary and its format using the format_dictionary command. The example then returns the value of the “Format2ExtensionMapping verilog 1995” key. The effect of the -silent and -exists switches are them shown.

get_instrument_dictionary -list

```
tshell_global
```

format_dictionary [get_instrument_dictionary tshell_global]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Extension2FormatMapping {
  verilog {
    v95 1995
    v 2001
    sv31a sv31a
    sv05 sv2005
    sv09 sv2009
  }
  vhdl {
    vhd87 1987
    vhd 1993
    vhd02 2002
    vhd08 2008
  }
}
Format2ExtensionMapping {
  verilog {
    1995 v95
    2001 v
    sv31a sv31a
    sv2005 sv05
    sv2009 sv09
  }
  vhdl {
    1987 vhd87
    1993 vhd
    2002 vhd02
    2008 vhd08
  }
}
Extension2DesignFormatMapping {
  v95 verilog_1995
  v verilog_2001
  sv31a verilog_sv31a
  sv05 verilog_sv2005
  sv09 verilog_sv2009
  vhd87 vhdl_1987
  vhd vhdl_1993
  vhd02 vhdl_2002
  vhd08 vhdl_2008
}
allowed_design_formats {verilog_1995 verilog_2001 verilog_sv31a
verilog_sv2005 verilog_sv2009 vhdl_1987 vhdl_1993 vhdl_2002 vhdl_2008}
date_time {
  format_string {
    %a %b
    %d %H:%M:%S
    %Z %Y
  }
  timezone {
  }
}
```

get_instrument_dictionary tshell_global Format2ExtensionMapping verilog 1995

```
v95
```

get_instrument_dictionary tshell_global junk

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Error: The specified dictionary 'tshell_global' does not contain the
//  requested key 'junk'.
```

get_instrument_dictionary tshell_global junk -silent

get_instrument_dictionary tshell_global junk -exists

```
0
```

Example 2

This example shows how to use the command to introspect the dictionary created by the process_dft_specification command based on the wrapper types that were processed. The dictionary mentor::ijtag::DftSpecification describes several aspects of the IJTAG network that the tool just processed. You can use the dictionary to retrieve the precise instance for the created nodes so that you can do custom editing within the process_dft_specification.post_insertion callback procedure.

get_instrument_dictionary -list

```
DftControlSources DftSpecification EDT ExtractSdc::dft_signals
LibraryCells PersistentCells RtlCells mentor::edt::DftSpecification
mentor::ijtag::DftSpecification mentor::jtag_bscan::DftSpecification
mentor::memory_bisr mentor::memory_bist::DftSpecification
mentor::occ::DftSpecification tshell_global
```

format_dictionary [get_instrument_dictionary mentor::ijtag::DftSpecification]

```
tap_ports {}
active_high_compliance_enables {}
active_low_compliance_enables {}
host_scan_interfaces {
  sri {
    type dr
    functions {
      tck {
        name corea_rtl_tessent_sib_sti_inst/ijtag_tck
        type pin
        exists 1
      }
      reset {
        name corea_rtl_tessent_sib_sti_inst/ijtag_reset
        type pin
        exists 1
        polarity active_low
      }
      scan_in {
        name corea_rtl_tessent_sib_sti_inst/ijtag_so
        type pin
        exists 1
      }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
        scan_out {
          name corea_rtl_tessent_sib_sti_inst/ijtag_so
          type pin
          exists 1
        }
        shift_en {
          name corea_rtl_tessent_sib_sti_inst/ijtag_se
          type pin
          exists 1
        }
        capture_en {
          name corea_rtl_tessent_sib_sti_inst/ijtag_ce
          type pin
          exists 1
        }
        update_en {
          name corea_rtl_tessent_sib_sti_inst/ijtag_ue
          type pin
          exists 1
        }
        select {
          name corea_rtl_tessent_sib_sti_inst/ijtag_sel
          type pin
          exists 1
        }
      }
    }
  }
  ImplementationOptions {
    use_capture_shift_en 0
    use_update_clock 0
    describe_capture_shift_en_in_icl 1
    describe_update_clock_in_icl 1
  }
  sri_ctrl_tdr {
    node_id tdr0
    port_list {
      ltest_en memory_bypass_en
      mcp_bounding_en tck_occ_en
      int_ltest_en ext_ltest_en
      int_mode ext_mode
    }
  }
  node_types {
    sib {
      sri sib0
      edt sib1
      sri_ctrl sib2
    }
    tdr {
      sri_ctrl tdr0
    }
  }
  inst(sib0) corea_rtl2_tessent_sib_sri_inst
  inst(sib1) corea_rtl2_tessent_sib_edt_inst
  inst(sib2) corea_rtl2_tessent_sib_occ_inst
  inst(tdr0) corea_rtl2_tessent_tdr_sri_ctrl_inst
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set id [get_instrument_dictionary mentor::ijtag::DftSpecification node_types tdr sri_ctrl]

```
tdr0
```

set inst [get_instrument_dictionary mentor::ijtag::DftSpecification inst($id)]

```
corea_rtl2_tessent_tdr_sri_ctrl_inst
```

Related Topics

format_dictionary

process_dft_specification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_instrument_parent_icl_module_list

Context: dft and patterns

Mode: analysis, insertion, and setup

Returns the parent or grandparent module names for ICL instance module names that match the specified *instrument_module_name*.

Usage

get_instrument_parent_icl_module_list ***instrument_module_name***

Description

Returns the parent or grandparent module names for ICL instance module names that match the specified ***instrument_module_name***. This command is useful to find all the actual ICL module names when it has been uniquified.

The tool returns the grandparent module name only when the icl_module attribute tessent_instrument_subtype of the ICL instances is shared_bus_assembly.

```
foreach module_name [get_instrument_parent_icl_module_list <instrument_controller_name> {
  iProcsForModule $module_name
    iProc ...
  }
```

The tool automatically inserts the command in the generated PDL files of the instrument.

Arguments

- ***instrument_module_name***

  A required string that specifies the module name to use in the module name matching.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_iproc_argument_default

Context: all contexts

Mode: all modes

Returns the default value for the specified arg_name of the specified proc_name attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by module_name.

Usage

get_iproc_argument_default ***proc_name arg_name*** [-exists] [-of_module *module_name*] [-silent]

Description

Returns the default value for the specified arg_name of the specified proc_name attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by module_name.

This command returns a null string when the specified argument has no default value. If you specify the -exists option, this command returns true if the specified argument has a default value and returns false when it does not.

Arguments

- ***proc_name***

  A required string that specifies the name of the iProc.

- ***arg_name***

  A required string that specifies the name of the argument.

- -exists

  An optional switch that changes the returned value of the command. When present, the command returns true when the specified argument has a default value and returns false when it does not. When not specified, the command returns the default value when it exists; otherwise it returns null.

- -of_module *module_name*

  An optional switch and value pair that constrains the command to return only the default value for the specified arg_name of the specified proc_name attached to the module specified by module_name.

- -silent

  An optional switch that specifies to suppress error messages if the specified proc_name, arg_name or module_name do not exist.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Return Values

If the -exist option is specified, this command returns true if the specified argument has a default value and returns false when it does not. If the -exist option is not specified, the command returns the default value when it exists; otherwise it returns null.

Examples

The following example lists each argument attached to the P1 iProc on module mychip and lists its default value, if one exists; otherwise it reports the argument does not have a default value.

```
foreach arg [get_iproc_argument_list P1 -of_module mychip] {
    if {![get_iproc_argument_default P1 $arg -exist]} {
        puts “Argument $arg has no default value”
    } else {
        set def [get_iproc_argument_default P1 $arg]
        puts “Argument $arg has default value of $def”
    }
}
```

Related Topics

delete_iprocs

get_iproc_argument_list

get_iproc_list

iProc

iProcsForModule

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_iproc_argument_list

Context: all contexts

Mode: all modes

Returns the list of argument names for the specified iProc attached to the ICL module that was specified by the last iProcsForModule command, or to the ICL module specified by module_name.

Usage

get_iproc_argument_list ***iproc_name*** [-of_module *module_name*] [-silent]

Description

Returns the list of argument names for the specified iProc attached to the ICL module that was specified by the last iProcsForModule command, or to the ICL module specified by module_name.

Arguments

- ***iproc_name***

  A required string that specifies the name of the iProc for which the arguments are returned.

- -of_module *module_name*

  An optional switch and value pair that constrains the command to return only instances of iProcs attached to the module specified by module_name.

- -silent

  An optional switch that specifies to suppress error messages if the specified iProc or module do not exist.

Return Values

A Tcl list that contains the argument names of the specified iProc.

Examples

The following example assigns the arguments attached to the chiptest iProc on module mychip to the variable chiptest_args:

```
set chiptest_args [get_iproc_argument_list chiptest -of_module mychip]
```

Related Topics

delete_iprocs

get_iproc_list

iProc

iProcsForModule

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_iproc_body

Context: all contexts

Mode: all modes

Returns the body for the specified iProc attached to the ICL module that was specified by the last iProcsForModule command, or to the ICL module specified by module_name.

Usage

get_iproc_body ***iproc_name*** [-of_module *module_name*] [-silent]

Description

Returns the body for the specified iProc attached to the ICL module that was specified by the last iProcsForModule command, or to the ICL module specified by module_name.

Arguments

- ***iproc_name***

  A required string that specifies the name of the iProc.

- -of_module *module_name*

  An optional switch and value pair that constrains the command to return only the body of iProcs attached to the module specified by module_name.

- -silent

  An optional switch that specifies to suppress error messages if the specified iProc or module do not exist.

Return Values

None

Examples

The following example returns the body of the chiptest iProc attached to module mychip:

```
get_iproc_body chiptest -of_module mychip
```

Related Topics

delete_iprocs

get_iproc_argument_list

get_iproc_list

iProc

iProcsForModule

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_iproc_list

Context: all contexts

Mode: all modes

Returns a Tcl list of iProcs attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by module_name.

Usage

get_iproc_list [-of_module *module_name*] [-silent]

Description

Returns a Tcl list of iProcs attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by module_name.

Arguments

- -of_module *module_name*

  An optional switch and value pair that constrains the command to return only instances of iProcs attached to the module specified by module_name.

- -silent

  An optional switch that specifies to suppress error messages if the specified module does not exist.

Return Values

A Tcl list of the iProc names, which are registered against the ICL module specified by the last iProcsForModule command or the ICL module specified by module_name.

Examples

The following example assigns the list of iProcs attached to the mychip ICL module to the variable myprocs:

```
set myprocs [get_iproc_list -of_module mychip]
puts $myprocs
```

Related Topics

delete_iprocs

get_iproc_argument_list

iProc

iProcsForModule

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_iproc_option

Context: all contexts

Mode: all modes

Returns properties of an iProc or iTopProc specification.

Usage

get_iproc_option ***proc_name*** {**-is_top_iproc** | **-file_path_name** | **-line_number**}
[-of_module *module_name*] [-silent]

Description

Returns properties of an iProc or iTopProc specification. These properties are not accessible through the get_iproc_argument_default, get_iproc_argument_list, get_iproc_body, or get_iproc_list commands.

Arguments

- ***proc_name***

  A required string that specifies the name of the iProc or iTopProc. The tool returns an error if ***proc_name*** does not exist.

- **-is_top_iproc** | **-file_path_name** | **-line_number**

  A required switch that specifies the value to return. Specify only one of the following:

  - **is_top_iproc** — Returns whether the proc_name was an iProc or an iTopProc. The tool returns “0” for an iProc and “1” for an iTopProc.
  - **file_path_name** — Returns the complete path name of the file containing the iProc or iTopProc specification. This switch requires that the tool processes the iProc or iTopProc specification files using the Tessent Shell source command. The result is empty when the tool processes specification files using the dofile command, or when in interactive mode. The result is also empty when the tool cannot determine file path name information.
  - **line_number** — Returns the line number of the iProc or iTopProc specification. This switch requires that the tool processes the iProc or iTopProc specification files using the Tessent Shell source command. The result is “0” when the tool processes specification files using the dofile command, or when in interactive mode. The result is also “0” when the tool cannot determine line number information.

- -of_module *module_name*

  An optional switch and value pair that specifies the ICL module associated with proc_name. When this switch is omitted, the tool uses the current iProcsForModule specification or returns an error if it does not exist.

- -silent

  An optional switch that specifies to suppress error messages.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example sources an iProc file for an ICL network and performs introspection to return some of the miscellaneous properties of the specification.

set_context patterns -ijtag
read_icl {data/icl/*.icl}
source data/pdl/block1.iprocs
report_iprocs

```
Module  iProcs
======  ======
chip    tap
block1  run_testa
```

get_iproc_option run_testa -is_top_iproc -of_module block1

```
0
```

file tail [get_iproc_option run_testa -file_path_name -of_module block1]

```
block1.iprocs
```

get_iproc_option run_testa -line_number -of_module block1

```
2
```

get_iproc_option tap -is_top_iproc -of_module chip

```
1
```

file tail [get_iproc_option tap -file_path_name -of_module chip]

```
block1.iprocs
```

get_iproc_option tap -line_number -of_module chip

```
36
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_layout_core_instance

Context: patterns -scan_diagnosis

Mode: setup, analysis

Returns the current layout core instance that was set with the set_layout_core_instance command.

Usage

get_layout_core_instance

Description

Returns the current layout core instance that was set with the set_layout_core_instance command.

Arguments

None

Related Topics

report_scan_polygons

set_layout_core_instance

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_license_queue_timeout

Context: all contexts

Mode: all modes

Returns the value specified by the previous set_license_queue_timeout command.

Usage

get_license_queue_timeout

Description

Returns the value specified by the previous set_license_queue_timeout command. The command returns the default value of “unlimited” if you have not issued the set_license_queue_timeout command in the current tool session.

Arguments

None

Examples

The following example sets the license queue timeout value to 3 minutes, then displays the timeout value that was set:

```
SETUP> set_license_queue_timeout 3
SETUP> get_license_queue_timeout

3
```

Related Topics

set_license_queue_timeout

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_load_unload_timing_option

Context: all contexts

Mode: all modes

Returns the value for load/unload timing options set with the set_load_unload_timing_options command for an icl_module, a module, the current_design, or the current physical block.

Usage

SSN Usage

```
get_load_unload_timing_option
    [-icl_module icl_module_name
      | -module module_name
      | -current_design
      | -current_physical_block]
    [-usage {ssn}]
    {-shift_clock_period [-time_unit {s | ms | us | ns | ps | fs | as_is}]
      | -ssn_bus_clock_period [-time_unit {s | ms | us | ns | ps | fs | as_is}]
      | -scan_en_setup_extra_cycles
      | -scan_en_hold_extra_cycles
      | -edt_update_setup_extra_cycles
      | -edt_update_hold_extra_cycles
      | -generated_timeplate_name
      | -ssn_bus_inputs_setup_relative_to_active_edge_percentage
      | -ssn_bus_outputs_measure_after_full_period_percentage
    }
    [-is_specified]
```

LogicBIST Usage

```
get_load_unload_timing_option
    [-icl_module icl_module_name
      | -module module_name
      | -current_design
      | -current_physical_block]
    [-usage {lbist}]
    { -scan_en_setup_extra_cycles
      | -scan_en_hold_extra_cycles
      | -edt_update_setup_extra_cycles
      | -edt_update_hold_extra_cycles
      | -prpg_en_setup_extra_cycles
      | -prpg_en_hold_extra_cycles
      | -misr_en_setup_extra_cycles
      | -misr_en_hold_extra_cycles
    }
    [-is_specified]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Description

Returns the value for load/unload timing options set with the set_load_unload_timing_options command for an icl_module, a module, the current_design, or the current physical block.

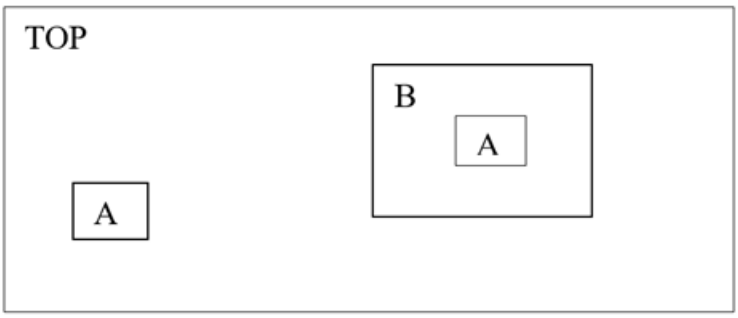
If you specified a module with the set_current_physical_block command and the load/unload options for the module, the tool returns those values. If you did not specify a module, the tool traverses the design hierarchy until it finds a parent that was specified as a physical block. Otherwise, the tool uses the module inherited from the current design, and returns the default values if no options were set.

If the module has multiple instantiations and all its instances inherit from the same parent, then the tool returns the values of that parent. If the instances of module inherit from different parents (see Figure 5-5), then the tool issues the following error:

```
//  Error: The specified Module 'foo'  was not set as a physical block by
//  'set_current_physical_block ' command. It has multiple instances that
//  do not belong to the same physical block.
```

For example, Figure 5-5 shows that module A was never specified as a physical block, and module B was specified as physical block. Because module A has two instances, one of them inherits from block B and the other inherits from current design TOP. This scenario will result in an error.

Figure 5-5. Design Inheritance and Module Matching



Arguments

- -icl_module *icl_module_name*

  A switch and string pair that specifies the name of a valid ICL module from which to return the load/unload timing option.

- -module *module_name*

  A switch and string pair that specifies the name of a module from which to return the load/unload timing option.

- -current_design

  A switch that returns the load/unload timing option for the current design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -current_physical_block

  A switch that returns the load/unload timing option for the current physical block.

- -usage {ssn | lbist}

  A switch and literal pair that returns the SSN or LBIST usage for the literal you specify.

- -shift_clock_period [-time_unit {s | ms | us | ns | ps | fs | as_is}]

  A switch, and optional switch and literal pair that returns the value as defined by the "set_load_unload_timing_options -shift_clock_period" command. This switch can only be used with -usage ssn.

  The -time_unit switch-value pair is an optional Boolean option that can be used when the command is returning a Value time property value; it is ignored otherwise. When specified or defaulted to as_is, the time value is returned unmodified with the unit string specified if it existed in the value of the configuration element. When specified to a value other than as_is, the time value is converted into a real number formatted to the specified units. The unit string is also dropped. The default is nanoseconds (ns).

- -ssn_bus_clock_period [-time_unit {s | ms | us | ns | ps | fs | as_is}]

  A switch, and optional switch and literal that returns the value as defined by the "set_load_unload_timing_options -ssn_bus_clock_period" command. This switch can only be used with -usage ssn.

  The -time_unit switch-value pair is an optional Boolean option that can be used when the command is returning a Value time property value; it is ignored otherwise. When specified or defaulted to as_is, the time value is returned unmodified with the unit string specified if it existed in the value of the configuration element. When specified to a value other than as_is, the time value is converted into a real number formatted to the specified units. The unit string is also dropped.

- -scan_en_setup_extra_cycles

  A switch that returns the value as defined by the "set_load_unload_timing_options -scan_en_setup_extra_cycles" command.

- -scan_en_hold_extra_cycles

  A switch that returns the value as defined by the "set_load_unload_timing_options -scan_en_hold_extra_cycles" command.

- -edt_update_setup_extra_cycles

  A switch that returns the value as defined by the "set_load_unload_timing_options -edt_update_setup_extra_cycles" command.

- -edt_update_hold_extra_cycles

  A switch that returns the value as defined by the "set_load_unload_timing_options -edt_update_hold_extra_cycles" command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -generated_timeplate_name

  A switch that returns the value as defined by the “set_load_unload_timing_options -generated_timeplate_name” command.

- -prpg_en_setup_extra_cycles

  A switch that returns the value as defined by the “set_load_unload_timing_options -prpg_en_setup_extra_cycles” command. This switch can only be used with -usage lbist.

- -prpg_en_hold_extra_cycles

  A switch that returns the value as defined by the “set_load_unload_timing_options -prpg_en_hold_extra_cycles” command. This switch can only be used with -usage lbist.

- -misr_en_setup_extra_cycles

  A switch that returns the value as defined by the “set_load_unload_timing_options -misr_en_setup_extra_cycles” command. This switch can only be used with -usage lbist.

- -misr_en_hold_extra_cycles

  A switch that returns the value as defined by the “set_load_unload_timing_options -misr_en_hold_extra_cycles” command. This switch can only be used with -usage lbist.

- -ssn_bus_inputs_setup_relative_to_active_edge_percentage

  A switch that returns the value as defined by the “set_load_unload_timing_options -ssn_bus_inputs_setup_relative_to_active_edge_percentage” command. This switch can only be used with -usage ssn.

- -ssn_bus_outputs_measure_after_full_period_percentage

  A switch that returns the value as defined by the “set_load_unload_timing_options -ssn_bus_outputs_measure_after_full_period_percentage” command. This switch can only be used with -usage ssn.

- -is_specified

  A switch that returns true if the specified timing option switch was set by a previous set_load_unload_timing_options command.

Examples

Example 1

The following example returns the shift clock period of the current_design.

```
get_load_unload_timing_option -usage ssn -current_design -shift_clock_period
```

Example 2

The following commands return the setup and hold extra cycle values defined by the set_load_unload_timing_options command. The PDL use these values to load the TDRs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
get_load_unload_timing_option -usage lbist -current_physical_block \
    -scan_en_setup_extra_cycles
get_load_unload_timing_option -usage lbist -current_physical_block \
    -scan_en_hold_extra_cycles
get_load_unload_timing_option -usage lbist -current_physical_block \
    -edt_update_setup_extra_cycles
get_load_unload_timing_option -usage lbist -current_physical_block \
    -edt_update_hold_extra_cycles
get_load_unload_timing_option -usage lbist -current_physical_block \
    -prpg_en_setup_extra_cycles
get_load_unload_timing_option -usage lbist- current_physical_block \
    -prpg_en_hold_extra_cycles
get_load_unload_timing_option -usage lbist -current_physical_block \
    -misr_en_setup_extra_cycles
get_load_unload_timing_option -usage lbist -current_physical_block \
    -misr_en_hold_extra_cycles
```

Related Topics

report_load_unload_timing_options

reset_load_unload_timing_options

set_load_unload_timing_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_loadboard_loopback_option

Context: all contexts

Modes: setup, analysis

Prerequisites: Design must be elaborated prior to using this command.

Returns the settings of options specified with the add_loadboard_loopback pairs command.

Usage

get_loadboard_loopback_option {-inputs | -outputs | -ac_delay_to_z}

Description

The get_loadboard_loopback_option returns the setting of a single option specified with the add_loadboard_loopback_pairs command

Arguments

- -inputs

  An optional switch that returns a collection of all input ports that are part of a loopback.

- -outputs

  An optional switch that returns a collection of all output ports that are part of a loopback.

- -ac_delay_to_z

  An optional switch that returns the ac_delay.

Examples

This example shows how you use this command to get each option set in the add_loadboard_loopback_pair:

**add_loadboard_loopback_pairs -inputs {din[4] ue} -outputs {dout[5] dout[0]} \
-ac_delay_to_z 10ns**

get_loadboard_loopback_option -inputs

```
{din[4] ue}
```

get_loadboard_loopback_option -outputs

```
{dout[5] dout[0]}
```

get_loadboard_loopback_option -ac_delay_to_z

```
10ns 10ns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_logfile

Context: unspecified, all contexts

Mode: all modes

Returns the name of and pathway to the current log file.

Usage

get_logfile

Description

Returns the name of and pathway to the current logfile as set by the tool invocation's -logfile *logfile_name* switch and value, or the value you set using the set_logfile_handling command.

Arguments

None.

Related Topics

set_logfile_handling

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_logical_library_list

Context: all contexts

Mode: all modes

Prerequisites: You must be in the -rtl context.

Returns a Tcl list with all logical design library names defined with the set_logical_design_libraries command.

Usage

get_logical_library_list [-default_library]

Description

Returns a Tcl list with all logical design library names defined with the set_logical_design_libraries command.

This command and the concept of logical libraries is only available in the RTL mode.

This command is useful for writing scripts that operate on all libraries such as in Example3 of the get_modules command reference.

Arguments

- -default_library

  An optional switch that specifies that the returned list only contains the default library.

Return Values

A Tcl list containing all logical design library names defined with the set_logical_design_libraries command.

Examples

In the following example, the Tcl variable libs is defined by the result returned by the get_logical_library_list command. You can then use a foreach command to iterate on the list elements.

```
set libs [get_logical_library_list]
```

Related Topics

set_logical_design_libraries

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_lvx_option

Context: patterns -scan, patterns -scan_retargeting

Mode: setup, analysis

Returns the current setting of an option specified with the set_lvx_options command.

Usage

get_lvx_option **-sequence** | **-repetitions** | **-payload_loop_count** | **-ssh_icl_instances** | **-sync_pulse_port** | **-sync_pulse_cycles**

Description

The get_lvx_option command returns information about the settings for creating LVX patterns.

Arguments

The following arguments are mutually exclusive; choosing one is required:

- **-sequence**

  A required switch that returns the LVX loop sequence to use for the pattern set.

- **-repetitions**

  A required switch that returns the number of times to repeat the sequence in the LVX payload.

- **-payload_loop_count**

  A required switch that returns the number of times to loop through the payload. If no count has been specified, this returns “off”.

- **-ssh_icl_instances**

  A required switch that returns a list of the SSH ICL instances targeted with the LVX pattern.

- **-sync_pulse_port**

  A required switch that returns the port pulsed at the start of each lvx_loop pattern set for LVP, if any.

- **-sync_pulse_cycles**

  A required switch that returns the number of shift cycles to pulse a port high at the start of each lvx_loop pattern set for LVP, if any.

Related Topics

set_lvx_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_master_module_name

Context: all contexts

Mode: all modes

Returns a string with the master_name attribute value for a design or icl module.

Usage

get_master_module_name [*obj_spec*] [-icl]

Description

When get_master_module_name is invoked without any arguments, a string is returned containing the master_name attribute value of the current design module. If the design is not elaborated, an empty string is returned.

The *obj_spec* string needs to match exactly one design or icl_module, or a collection consisting of one design module or icl_module. Specifying a design module returns the master_name attribute value for that design module. When specifying an icl_module, the -icl switch is also used to return the master_name attribute value for that icl_module. When *obj_spec* is a collection containing one design module or icl_module, the -icl switch is ignored, and the master_name attribute value of the specified design module or icl_module is returned.

Invoking get_master_module_name with only the -icl switch returns a string that contains the master_name attribute value of the current icl top module, or an empty string if there is no icl elaboration.

Arguments

- *obj_spec*

  An optional switch that specifies a single design or icl module, or a collection that contains one design module or icl module.

- -icl

  An optional switch that specifies a string of the master_name attribute value for the specified icl module is returned. An empty string is returned if the design is not icl elaborated.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_memory_instances

Context: dft

Mode: all modes

Prerequisite: The current design must be set with the set_current_design command.

Returns a collection of the instances for which there is a matched TCD Memory description.

Usage

get_memory_instances [*name_patterns*]
[-below_instances *instance_objects*]
[-filter *attribute_equation*] [-silent]
[-external_repair_ready]
[-repairable] [-non_repairable]
[-type *memory_type*]

Description

Returns a collection of instances for which there is a matched TCD Memory description. See "Tessent Core Description" on page 4793 for more information.

Memory instances within a lower level sub-block or physical block are excluded from the reporting. You can restrict the search to instances below a given collection of instances, and filter the collection based on an arbitrary filtering equation involving other attributes registered against or inherited by the instance object type.

If none of the options of this command is used, the command returns a collection that contains all of the memory instances in the design.

The -silent option suppresses the warning that is normally generated if no memory instances are returned.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of instances. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no *name_patterns* are specified, the tool searches all memory instances.

  If you specify the -below_instances option, the name patterns are hierarchical name patterns relative to the specified instance objects; otherwise, the name patterns are relative to the current design.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for instances below the specified *instance_objects*. The specified *name_patterns* are searched relative to each instance found in the *instance_objects* list.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the specified *instance_spec* is an empty collection, all instances below the current design are considered. See the description of the -below_instances switch of the get_instances command description for the justification of this behavior.

- -filter *attribute_equation*

  An optional switch and string pair that specifies to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the instance object type. See section "Attribute Filtering Equation Syntax" on page 3832 for more details on filtering using the attribute equation format.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

  Note

  You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

- -external_repair_ready

  An optional switch that returns a collection of memory instances that are compatible for external repair. External repair is implemented with the set_memory_instance_options -generate_external_repair_logic command and option.

- -repairable

  An optional switch that returns a collection of memory instances that contain spare elements within their memory TCD definitions. The collection does not contain memory instances that are within a MemoryCluster.

- -non_repairable

  An optional switch that returns a collection of memory instances the do not contain spare elements within their memory TCD definitions. The collection does not contain memory instances that are within a MemoryCluster.

- -type *memory_type*

  An optional switch and string pair that returns a collection of memory instances in which the specified *memory_type* matches the MemoryType property specified in the memory TCD (sram, rom, or dram), or cluster for memory instances that are within a MemoryCluster.

Examples

The following example uses the get_memory_instances command to return all instances of a module below a given instance for which there is a TCD Memory description.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_memory_instances -below_instance subblockA_i1

```
{subblockA_i1/mem1 subblockA_i1/mem2}
```

Related Topics

get_memory_instance_option

report_ijtag_instances

set_memory_instance_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_memory_instance_option

Context: all contexts

Mode: all modes

Prerequisite: The current design must be set with the set_current_design command.

Returns the value of an option on a specific memory instance, or returns the value of an option that applies to all memory instances.

Usage

get_memory_instance_option {***memory_instance*** [-bist_data_in_pipelining |
-physical_cluster_override | -power_domain_island | -test_clock_override |
-use_in_memory_bist_dft_specification | -use_in_memory_bisr_dft_specification] |
-generate_external_repair_logic |
-repair_sharing | -partitioning_group} |
-physical_cluster_size_ratio

Description

Returns the value of an option on a specific memory instance, or returns the value of an option that applies to all memory instances.

Arguments

- ***memory_instance***

  A required string that specifies the name of a memory instance or a collection containing a single memory instance. You can use the "foreach_in_collection inst [get_memory_instances]" command to iterate on all memory instances. You can use this option with any option other than -physical_cluster_size_ratio.

  Tip

  If your ***memory_instance*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- -bist_data_in_pipelining

  A Boolean option that specifies to return the value previously specified with the "set_memory_instance_options -bist_data_in_pipelining" command on the specified memory_instance.

- -physical_cluster_override

  A Boolean option that specifies to return the value previously specified with the "set_memory_instance_options -physical_cluster_override" command on the specified memory_instance.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -power_domain_island

  A Boolean option that specifies to return the computed power domain island for the specified memory_instance. This property is only defined when a UPF or CPF file was initially read in using the read_cpf or read_upf commands and you are in the analysis system mode. Two memories share a common power domain island name when they have the same power_domain_name and the two instances are not logically separated by instances having a different power_domain_name. For example, memories in u1 and memories in u1/u2/u3 sharing a common power_domain_name are in two power domain islands if u1/u2 is associated with a different power_domain_name.

- -test_clock_override

  A Boolean option that specifies to return the value previously specified with the "set_memory_instance_options -test_clock_override" command on the specified memory_instance.

- -use_in_memory_bist_dft_specification

  A Boolean option that specifies to return the value previously specified with the "set_memory_instance_options -use_in_memory_bist_dft_specification" command on the specified memory_instance. In analysis system mode, the value auto is replaced by on or off depending on whether the memory instance is already connected to a memory BIST interface module.

- -use_in_memory_bisr_dft_specification

  A Boolean option that specifies to return the value previously specified with the "set_memory_instance_options -use_in_memory_bisr_dft_specification" on the specified memory_instance. In analysis system mode, the value auto is replaced by on or off depending on whether the memory instance is already connected to a memory BISR register module.

- -generate_external_repair_logic

  A Boolean option that specifies to return the value previously specified with the "set_memory_instance_options -generate_external_repair_logic" command on the specified memory_instance.

- -repair_sharing

  A Boolean option that specifies to return the value previously specified with the "set_memory_instance_options -repair_sharing" command on the specified memory_instance.

- -partitioning_group

  A Boolean option that specifies to return the value previously specified with the "set_memory_instance_options -partitioning_group *group_label*" command on the specified memory instance.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -physical_cluster_size_ratio

  A Boolean option that specifies to return the value previously specified with the "set_memory_instance_options -physical_cluster_size_ratio" command. You cannot use this option together with the memory_instance option since it applies to all memory instances.

Examples

The following example performs a query using the -use_in_memory_bist_dft_specification option in both setup and analysis mode. Note how the value "auto" converts to "on" in the analysis mode. This means the memory instance is not already connected to a memory BIST interface module.

```
SETUP> get_memory_instance_option -use_in_memory_bist_dft_specification
auto

SETUP> check_design_rules
...
ANALYSIS> get_memory_instance_option -use_in_memory_bist_dft_spec
on
```

Related Topics

report_ijtag_instances

set_dft_specification_requirements

set_memory_instance_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_module_matching_options

Context: all contexts

Mode: all modes

Returns the settings specified with the set_module_matching_options and the "set_insertion_options -edited_module_prefix" commands for module matching.

Usage

Usage 1

get_module_matching_options {**-prefix_pattern_list** | **-suffix_pattern_list**} [-regexp | -nocase]

Usage 2

get_module_matching_options **-format** ***format_type*** **-excluded_modules_from_module_matching**

Usage 3

get_module_matching_options **-module_name_map_list**

Description

Returns the settings for module matching as specified by set_module_matching_options commands and the "set_insertion_options -edited_module_prefix" commands.

Depending on the arguments you specify, the tool returns the following:

- A list of prefixes.
- A list of suffixes.
- A list of 1s and 0s identifying if the corresponding element of the prefix or suffix pattern list is a regexp or a glob expression.
- A list of 1s and 0s identifying if the corresponding element of the prefix or suffix pattern list is to be matched irrespective of case or not.
- A collection of excluded modules that are not considered during module matching for the defined format specified with the "set_module_matching_options -format" command.

Arguments

- -prefix_pattern_list

  A switch that instructs the tool to return the list of prefixes as defined by set_module_matching_options command. This includes module prefixes you assign with the "set_insertion_options -edited_module_prefix" command and switch.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -suffix_pattern_list

  A switch that instructs the tool to return a list of suffixes as defined by set_module_matching_options command.

- -regexp

  An optional switch to the prefix_pattern_list or the suffix_pattern_list switch that instructs the tool to return a list of 1s and 0s identifying if the corresponding element of the prefix or suffix pattern list is to be matched irrespective of case or not.

- -nocase

  An optional switch to the prefix_pattern_list or the suffix_pattern_list switch that instructs the tool to return the set 'no case sensitive' values as 1 (on) or 0 (off) in the same sequence of the prefixes or suffixes as a list.

- **-format *format_type***

  A switch and literal pair that instructs the tool to return a collection of modules to be ignored during module matching for the format type you specified with the "set_module_matching_options -format ... -excluded_modules_from_module_matching" command. The -format switch is mandatory when you specify the -excluded_modules_from_module_matching switch.

  format_type must be one of the following:

  - icl
  - tcd_bscan
  - tcd_scan
  - tcd_fusebox
  - tcd_memory
  - tcd_memory_cluster

- -excluded_modules_from_module_matching

  A switch used in conjunction with the -format switch that instructs the tool to return a collection of modules to be ignored during module matching with the format type you specified with the "set_module_matching_options -format ... -excluded_modules_from_module_matching" command. The -format switch is mandatory when you specify the -excluded_modules_from_module_matching switch.

  If nothing was previously set for the specified format, an empty collection is returned.

- -module_name_map_list

  A switch that instructs the tool to return a list of patterns as defined by the "set_module_matching_options -module_name_map_list *list_forming_mapping_pairs*" command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following demonstrates various uses of the get_module_matching_options command:

set_module_matching_options -prefix_pattern_list "none_" -suffix_pattern_list ""

set_module_matching_options -prefix_pattern_list "www" -regexp -append

set_module_matching_options -prefix_pattern_list "pre" -append

set_module_matching_options -prefix_pattern_list "pre_" -append

set_module_matching_options -prefix_pattern_list "pre" -append

set_insertion_options -edited_module_prefix "pre"
report_module_matching_options

```
//  Option                        Value
//  ---------------------------   -----
//  case insensitivity override   off
//
//  affix type  regexp  pattern  by insertion*
//  ----------  ------  -------  -------------
//  prefix      no      "*ts_graybox_"       no
//  prefix      no      "none_"              no
//  prefix      no      "*ts_graybox_none_"  no
//  prefix      yes     "www"                no
//  prefix      yes     ".*ts_graybox_www"   no
//  prefix      no      "pre"                no
//  prefix      no      "*ts_graybox_pre"    no
//  prefix      no      "pre_"               no
//  prefix      no      "*ts_graybox_pre_"   no
//  prefix      no      "pre_"               yes
//  by insertion*: Additional prefix for ICL format defined by command
//  'set_insertion_options -edited_module_prefix'
```

get_module_matching_options -prefix_pattern_list

```
*ts_graybox_ none_ *ts_graybox_none_ www .*ts_graybox_www pre
*ts_graybox_pre pre_ *ts_graybox_pre_
```

get_module_matching_options -suffix_pattern_list

```
{}
```

get_module_matching_options -prefix_pattern_list -regexp

```
0 0 0 1 1 0 0 0 0
```

get_module_matching_options -suffix_pattern_list -regexp

```
0
```

get_module_matching_options -prefix_pattern_list -regexp

```
0 0 0 1 1 0 0 0 0
```

get_module_matching_options -suffix_pattern_list -regexp

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

0

get_module_matching_options -prefix_pattern_list -nocase

0 0 0 0 0 0 0 0 0

Related Topics

set_module_matching_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_module_parameter_list

Context: all contexts

Mode: all modes

Returns the list of parameters in a module.

Usage

get_module_parameter_list ***module_object*** [-overridden]

Description

Returns the list of parameters in a module.

There are restrictions you need to consider when you use this command on a module without a definition

- Only parameters that are specified on instantiation can be determined.
- The returned list of parameters is the same whether or not you use the -overridden switch.
- When parameters are not specified by name, but by order, the tool generates a name. The generated name is a sequential integer, starting at zero, indicating the position of the parameter on the instantiation.

This command is only available in -rtl mode (set_context dft -rtl). For more information, see the set_context command.

Arguments

- ***module_object***

  A required string that specifies the parameterized module objects whose parameters are to be returned.

- -overridden

  An optional switch that specifies to only list those parameters whose default values are overridden during an instantiation.

Return Values

List of parameters.

Examples

Example 1

In the following example, the get_module_parameter_list is used to obtain the list of parameters overridden in the specific instantiation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set inst u1/myinst_i1
set params [get_module_parameter_list [get_modules -of_instances $inst] -overridden]
```

Example 2

This example shows the names generated by the tool when parameters are not specified by name but by order.

```
set mods [get_module -of_instance [get_instances]]
foreach_in_collection mod $mods {
  puts "--- Module [get_single_name $mod] ---"
  foreach param [get_module_parameter_list $mod] {
    puts "parameter $param = [get_module_parameter_value $param -of_module $mod]"
  }
}
```

```
--- Module DW01_add@PV1 ---
parameter 0 = 8
--- Module DW01_add@PV2 ---
parameter 0 = 16
--- Module def_named@PV1 ---
parameter d_size = 3
parameter q_size = 4
--- Module def_ordered@PV1 ---
parameter d_size = 2
parameter q_size = 5
--- Module nodef_named@PV1 ---
parameter d_size = 3
parameter q_size = 4
--- Module nodef_ordered@PV1 ---
parameter 0 = 2
parameter 1 = 5
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_module_parameter_value

Context: all contexts

Mode: all modes

Returns the value of the parameter on the specified module object.

Usage

get_module_parameter_value ***parameter_name***
[-of_module *module_object*]
[-format {tcl | verilog | vhdl}]

Description

Returns the value of the parameter on the specified module object.

This command is only available in -rtl mode (set_context dft -rtl). For more information, see the set_context command.

Arguments

- ***parameter_name***

  A required string that specifies the name of the parameter whose value is to be returned.

- -of_module *module_object*

  An optional switch and string pair that specifies the design module whose parameter value is to be returned. When omitted, the parameters of the current design are introspected.

- -format {tcl | verilog | vhdl}

  An optional switch and literal pair to specify which format the tool uses to return binary and hexadecimal numbers.

  tcl — Use this switch if you want the tool to return binary or hexadecimal numbers in the Tcl format. This is the default.

  ```
  0b<num>
  0x<num>
  ```

  verilog — Use this switch if you want the tool to return binary or hexadecimal numbers in the Verilog format:

  ```
  'b<num>
  'h<num>
  ```

  vhdl — Use this switch if you want the tool to return binary or hexadecimal numbers in the VHDL format:

  ```
  B"<num>"
  H"<num>"
  ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Return Values

The value of the requested design module parameter.

Examples

In the following example, the values of all overridden parameters are echoed in the transcript.

```
set inst [get_instance u1/myinst_i1]
set mod [get_modules -of_instances $inst]
set params [get_module_parameter_list $mod -overridden]
foreach param $params {
        puts “parameter $param = [get_module_parameter_value $param \
            -of_mod $mod]”
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_modules

Context: all contexts

Mode: all modes

Returns a collection of all modules that match the specified *name_patterns* list.

Usage

get_modules [*name_patterns*] [-use_module_matching_options]
[-of_instances *instance_objects* | [-below_instances *instance_objects*] [-uniquified] |
-of_ports *port_objects* | -of_icl_modules *icl_module_objects* |
-parameterized_views_of_modules *module_objects* | -original_module_of *module*]
[-in_library *library*] [-vhdl_architecture *architecture*]
[-of_type *type*]
[-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Description

By default, this command returns all modules below the current design, and all modules present in memory, for example those modules you have read into the tool but are not yet instantiated. You can use the many switch options to constrain the search for your needs. If no modules exist in memory, the command returns an empty collection.

You can have any number of wildcards or regular expressions in the name patterns.

In the rtl context, you can have more than one logical library. Because VHDL is also supported in the rtl context, you can also have multiple architectures. Use the set_logical_design_libraries command to define the list of available logical libraries.

In the no_rtl context, only one library exists and is it named “work”.

The tool searches for the objects specified by the *name_patterns* argument in all libraries unless the -in_library option is specified. All other commands that have an option that refers to module names only search the default library. For example, you can use the following command to create an instance of module M1 from library L1:

create_instance instance_name -of_module [get_module M1 -in_library L1]

In another example, an instance of module M1 from the default library is created because only the get_modules command knows how to look into other libraries.

create_instance instance_name -of_module M1

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to match the returned list of modules against. If you do not specify any *name_patterns*, the tool searches all modules.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -use_module_matching_options

  An optional switch that instructs the command to return a collection of all the modules that can be found by combining the suffix and the prefix specifications from the set_module_matching_options command with the given *name_patterns*. This option is useful when looking for modules by names that were potentially uniquified during synthesis. When you use this switch, the *name_patterns* list is mandatory and cannot include wildcards. The -regexp and -nocase switches are ignored. The module name matching for parameterized modules is based on the module base names, such that the returned collection includes the base module and all its parameterized views.

  Note

  The tool uses the settings and functionality of the module matching process to retrieve the results, but does not take into account previous module matching results.

- -of_instances *instance_objects*

  An optional switch and value pair that constrains the command to return only modules of the specified *instance_objects*. You could use the module_name attribute of the instance objects to perform similar filtering, but this method is more direct and efficient and automatically deals with libraries and architectures.

  This switch is mutually exclusive with the -below_instances, -uniquified, -of_ports, -of_icl_modules, -parameterized_views_of_modules, and -original_module_of switches.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to return only modules corresponding to instances found below the specified *instance_objects*. The tool considers all instances below the current design if the specified *instance_spec* is an empty collection. See the description of the -below_instances switch of the get_instances command for the justification of this behavior.

  You can use this switch with the -uniquified switch, but it is mutually exclusive with the use of the -of_instances, -of_ports, -of_icl_modules, -parameterized_views_of_modules, and -original_module_of switches.

- -uniquified

  An optional switch that constrains the command to return only modules that were uniquified since entering the insertion system mode.

  You can use this switch with the -below_instances switch, but it is mutually exclusive with the use of the -of_instances, -of_ports, -of_icl_modules, -parameterized_views_of_modules, and -original_module_of switches.

- -of_ports *port_objects*

  An optional switch and value pair that constrains the command to return only modules of the specified *port_objects*. You could perform similar filtering operation using the module_name attribute of the port objects, but using this switch is more direct and efficient and automatically deals with libraries and architectures.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This switch is mutually exclusive with the use of the -of_instance, -below_instances, -uniquified, -of_icl_modules, -parameterized_views_of_modules, and -original_module_of switches.

- -of_icl_modules *icl_module_objects*

  An optional switch and value pair that constrains the command to return the modules corresponding to the specified *icl_module_objects*. You can retrieve the *icl_module_objects* using the get_icl_modules command. If the tool is successful, it returns a collection of these modules, and if not, it reports an error unless you specify the -silent switch.

  This switch is mutually exclusive with the use of the -of_instance, -below_instances, -uniquified, -of_ports, -parameterized_views_of_modules, and -original_module_of switches.

  Note

  The tool considers prefixes and suffixes you define with the set_module_matching_options command. If the base name of a design module matches an ICL module, the tool adds the related module to compute the results.

- -parameterized_views_of_modules *module_objects*

  An optional switch and value pair that constrains the command to return a collection of modules corresponding to all the parameterized views of the module objects specified. You can specify the *module_objects* as a Tcl list of module names or a collection of modules. Whether *module_objects* contains a parameterized view or the master copy of the module, this switch instructs the command to find all parameterized views and return them. Using this switch is equivalent to using the master_name attribute and requesting all modules having the same master_name value. However, using this switch is more efficient, and it also handles logical libraries. Use this switch to determine whether a module is uniquely instantiated or not and to find out if a module has multiple parameterized views.

  This switch is mutually exclusive with the use of the -of_instance, -below_instances, -uniquified, -of_ports, -of_icl_module, and -original_module_of switches.

- -original_module_of *module*

  An optional switch and value pair that returns the original module from which the module was copied. A module is copied from another module either by unification or with the copy_module command. The *module_object* parameter can be the name of a single module or a collection of one module. The module may be a parameterized view of the copied module. The tool returns the master copy of the original module. If the specified *module* is not a copied module, the tool reports an error unless you specify the -silent switch, in which case an empty collection is returned.

  This switch is mutually exclusive with the use of the -of_instance, -below_instances, -uniquified, -of_ports, -of_icl_module, and -parameterized_views_of_modules switches.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -in_library *library*

  An optional switch and value pair that specifies to restrict the search to a specified library. The library value must be a single string corresponding to one of the logical library names defined with the set_logical_design_libraries command. When you do not specify the -in_library option, all libraries are searched. This option is only permitted in the rtl context.

  Using -in_library XXX is equivalent to using -filter "library_name == XXX".

- -vhdl_architecture *architecture*

  An optional switch and value pair that specifies to restrict the search to a specific VHDL architecture name. This option is only useful when searching VHDL modules. When you do not specify this option, one module per available architecture is returned. This option is only permitted in the rtl context.

  Using -vhdl_architecture XXX is equivalent to using -filter "vhdl_architecture == XXX".

- -of_type *type*

  An optional switch and value pair that constrains the command to return only modules of the specified types. The type is a Tcl list of one or more strings matching one of the following values: design or cell.

- -filter *attribute_equation*

  An optional switch and string pair that specifies to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the module object type. See section "Attribute Filtering Equation Syntax" on page 3832 for more details on filtering attribute equation format. The =~ and !~ matching operator use simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option.

- -regexp

  An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a real regular expression instead of as a simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for modules matching the *name_patterns*.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The command does not return any module if you specify *name_patterns* that do not match used module names or specify a filter whose condition was not met by any module. The -silent option does not affect any other error reporting. When using the -silent switch, your code that follows should check for empty collections and deal with them correctly.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

Examples

Example 1

The following example looks for modules whose name matches a* or b*.

```
get_modules {a* b*}
```

```
{ab abc b bcd}
```

Example 2

The following example looks for modules having one or two characters.

```
get_modules {[:alnum:]{1,2}} -regexp
```

```
{ab b}
```

Example 3

The following example gets the module object associated to an instance.

```
get_modules -of_instances u3/u5
```

```
{myModule}
```

Example 4

The following example looks for modules in all libraries and displays their names and language value in a tabular format.

```
set format_string "   "
append format_string "module_name = %-10s "
append format_string "language = %-10s "

set libs [get_logical_library_list]foreach lib $libs {
   puts "Modules in library $lib are:"
   set mods [get_modules -in_library $lib]
   foreach_in_collection mod $mods {
      puts [format $format_string \
      [get_att_val $mod -name name]\
      [get_att_val $mod -name language]]
   }
}
```

```
Modules in library lib1 are:
   module_name = dut        language = verilog
Modules in library lib2 are:
   module_name = dut        language = verilog
Modules in library work are:
   module_name = top        language = verilog
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 5

This example shows how to report the names of the parameterized views for a module.

puts "[get_name_list [get_modules -parameterized_views_of_modules ff]]"

```
ff ff@PV1 ff@PV2
```

Example 6

The -use_module_matching_options switch is used to look for modules based upon the prefix and suffix values defined by the set_module_matching_options command.

//Setting the prefix and suffix module matching option

**set_module_matching_options -prefix_pattern_list {a*} \
 -suffix_pattern_list {_1 _2}**

// Getting the modules

**puts [join [get_name_list [get_modules {mymodule} \
 -use_module_matching_options]] "\n"]**

//The returned modules are:

```
a_mymodule
a_mymodule_1
a_mymodule_2
ab_mymodule
ab_mymodule_1
ab_mymodule_2
mymodule
mymodule_1
mymodule_2
```

Example 7

In this example, the -use_module_matching_options switch is used to look for parametrized modules based upon the suffix values defined here by the related defaults for the module matching process.

// Getting the current active suffixes for the module matching process
get_module_matching_options -suffix_pattern_list

```
{[_]+[0-9]+}
```

// Getting the modules
get_modules SYNC_1RW_GENERIC -use_module_matching_options

```
{SYNC_1RW_GENERIC_1 SYNC_1RW_GENERIC_1@PV1 SYNC_1RW_GENERIC_1@PV2}
```

Example 8

This example gets a collection of all read_cell_library models that contain a _ram, _rom, or _cram primitive at any level within that cell's hierarchy.

get_modules -of_type cell -filter has_ram_or_rom_primitives

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

create_module

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_multiprocessing_option

Context: unspecified, dft -edt, patterns -scan

Mode: setup, analysis, insertion

Returns the value of a single specified variable previously set with the set_multiprocessing_options command.

Usage

get_multiprocessing_option {-generic_delete | -generic_scheduler
| -license_timeout
| -lsf_heuristics | -lsf_learning | -lsf_options | -multithreading
| -processors_per_grid_request | -remote_shell | -result_time_limit
| -scheduler_timeout | -sge_options}

Description

Returns the value of a single specified variable previously set with the set_multiprocessing_options command.

The get_multiprocessing_option command returns the value of a single multiprocessing variable that affects the behavior of the tool's built-in multiprocessing manager. These are values that you can set with the set_multiprocessing_options command. To display the values of multiprocessing variables in a human-readable format, use the report_multiprocessing_options command.

For more information about setting up for multiprocessing, refer to "Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual* and "The Tessent Tcl Interface" in the *Tessent Shell User's Manual*.

Arguments

The arguments for the get_multiprocessing_option command are identical to those of the set_multiprocessing_options command, described on page 2743. For more information about the purpose of each switch, refer to the list of arguments there.

Examples

The following example sets and then returns the value of the remote_shell variable:

```
set_multiprocessing_options -remote_shell rsh
puts [get_multiprocessing_option -remote_shell]
```

```
rsh
```

Related Topics

add_processors

delete_processors

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_multiprocessing_options

report_processors

set_multiprocessing_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_name_list

Context: unspecified, all contexts

Mode: all modes

Returns the names of the specified objects, or the names of the name list with simplified escaped identifiers, if possible.

Usage

get_name_list {***object_spec*** | **-from_object_names** ***name_list*** [-language {verilog | vhdl}]} [-remove_escaping] [-silent]

Description

This command returns a list with the value of the name attribute for the objects specified by ***object_spec*** or the names of the ***name_list*** with simplified escape identifiers, if possible.

Using this command is equivalent to running the following command:

get_attribute_value_list ***object_spec*** **-name** ***name***

Using this command with the "-from_object_names *name_list*" option verifies the existence of the names in the design and removes simple escaped identifiers if they are not reserved words in the design language. You can use this to identify name strings with reserved design language words.

The -language argument defines which design language escaping should be removed while processing the -from_object_names name list.

The -remove_escaping argument removes the backslash ("\") and the trailing spaces in an escaped identifier. Many legacy commands such as add_clocks require the instance name to have the escape character stripped from its name. The argument also removes the escaping on reserved design language words. However, these legacy commands accept collections as inputs in cases where the name format is not important. See the Examples section in this topic for an illustration of this behavior.

If ***object_spec*** or ***name_list*** entries are not valid, an error message displays unless the -silent switch is specified.

Arguments

- ***object_spec***

  A required string that specifies a Tcl list of one or more object names, or a collection of one or more objects.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-from_object_names *name_list*** [-language {verilog | vhdl}]

  A required switch and string or Tcl list of object name strings pair, with an optional switch and literal pair. Simple escaping in parts of the names is removed from the defined language type if they are not language-specific reserved words.

  -language {verilog | vhdl}

  An optional switch and literal pair that specifies which language (Verilog or VHDL) should be considered during the removal of simple escaping. The default is Verilog when this option is not specified.

- -remove_escaping

  An optional switch that specifies to strip out the \ and the trailing spaces in escaped identifiers. This option is useful to interact with legacy commands and the collection is not directly passed as input to them. See Example 2 for an illustration of this usage.

- -silent

  An optional switch that specifies to suppress error messages if an object in ***object_spec*** does not exist.

Examples

Example 1

The following example retrieves the names for a collection of pins.

get_name_list [get_pins u1/i*]

```
{u1/i1 u1/i2 u1/i3}
```

Example 2

The following example illustrates the use of the -remove_escaping option. In this example, you can see how legacy commands natively support collections as input arguments but do not support names with escaped identifiers.

Note the use of the double set of braces when specifying a name pattern containing escape identifiers to get_pins. These braces are needed to prevent the Tcl shell from interpreting the backslash character ("\") as its own escape sequence. You could also escape the backslash using {\\xyz_i# /a} but using two sets of braces is easier when the pattern includes many slashes. See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a complete description of pattern syntax and suggestions to use when the pattern is targeted for names containing escape characters.

get_pins xyz*/*

```
{{\xyz_i# /a} {\xyz_i# /b} {\xyz_i# /c_in} {\xyz_i# /s} {\xyz_i# /c_out}}
```

add_clocks [get_pins xyz*/*]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Warning: Primary input '\xyz_i# /b_pport' is added at pin
'/\xyz_i# /b'
//  Warning: Primary input '\xyz_i# /c_in_pport' is added at pin
'/\xyz_i# /c_in'
//  Warning: Primary input '\xyz_i# /s_pport' is added at pin
'/\xyz_i# /s'
//  Warning: Primary input '\xyz_i# /c_out_pport' is added at pin
'/\xyz_i# /c_out'
```

add_clocks {\xyz_i# /a}

```
// Error: '
      yz_i# is not a primary input.
```

set names [get_name_list [get_pins xyz*/*] -remove_escaping]

```
{{xyz_i#/a} {xyz_i#/b} {xyz_i#/c_in} {xyz_i#/s} {xyz_i#/c_out}}
```

add_clocks $names

Example 3

The following example retrieves the names from a Tcl list of names of instances and instance pins with the -from_object_names option.

puts "from_object_names [get_name_list -from_object_names {{deep_input1/\input /\output .\e } {deep_input1/\input } {d/\a /\b /\c .\e }}]"
from_object_names {deep_input1/\input /\output .e} {deep_input1/\input } d/a/b/c.e

Due to name escaping, the names are defined in braces for the same reason as described in Example 2. Hierarchy separators are retained, so the "." is not changed to "/". The "input" and "output" name parts are Verilog keywords, so the escaping is not removed at those name sections. The last name does not contain keywords, so the escaping is removed.

Example 4

The following example retrieves the names from a Tcl list of names of instances and instance pins with the -from_object_names option.

puts "from_object_names [get_name_list -from_object_names {{deep_input1/\input /\output .\e } {deep_input1/\input } {d/\a /\b /\c .\e }} -remove_escaping]"
from_object_names deep_input1/input/output .e deep_input1/input d/a/b/c.e

Due to name escaping, the names are defined in braces for the same reason as described in Example 2. Hierarchy separators are retained, so the "." is not exchanged to "/". The "input" and "output" name parts are Verilog keywords but also removed due to the -remove_escaping switch. The last name does not contain keywords, so the escaping is removed.

Related Topics

get_attribute_list

get_attribute_option

get_attribute_value_list

get_common_parent_instance

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_ports

get_modules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_nets

Context: all contexts

Mode: all modes

Returns a collection of all hierarchical nets instantiated relative to the current design that match the specified name_patterns list, subject to filtering imposed by the specified options.

Usage

get_nets [*name_patterns*] [-below_instances *instance_objects*]
[-of_pins *pin_objects* | -of_ports *port_objects* | -of_gate_pins *gate_pin_objects*
|-of_pseudo_ports *pseudo_port_objects*] [-bundle]
[-hierarchical] [-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Description

Returns a collection of all hierarchical nets instantiated relative to the current design that match the specified *name_patterns* list, subject to filtering imposed by the specified options.

When the *name_patterns* list is omitted, all nets are returned subject to filtering imposed by the specified options.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of nets. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no *name_patterns* are specified, the tool searches all nets. The following limitations apply:

  - You can use wildcards in *name_patterns*, but you cannot use a wildcard as part of the left or right index of a range. For example: [* : <right_index>] is not supported.
  - A range (a dimension of the form [<left_index> : <right_index>]) can only be used in the last dimension of the last field of the *name_patterns*.

  See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a complete description of the pattern syntax when the -regexp option is used and not used.

  Tip

  If your *name_patterns* string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for nets below the specified *instance_objects*. Specified *name_patterns* are searched relative to the specified *instance_objects* when this option is used.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the specified *instance_spec* is an empty collection, all instances below the current design are considered. See the description of the -below_instances switch of the get_instances command description for the justification of this behavior.

- -of_pins *pin_objects*

  An optional switch and value pair that constrains the command to return the net objects associated to the *pin objects*. Those are the nets having the exact same name as the pins and residing inside the instances on which the pins are on. For pins of cells, the inside nets corresponding to the pins are not visible and thus not returned. You cannot use a *name_patterns* list when using this option.To get the net connected to pins from the outside, use get_fanin and get_fanout -stop_on net.

- -of_ports *port_objects*

  An optional switch and value pair that constrains the command to return the net objects associated to the port_objects. Those are the nets having the exact same name as the ports and reside inside the current design. You cannot use a *name_patterns* list when using this option.

- -of_gate_pins *gate_pin_objects*

  An optional switch and value pair that constrains the command to return the net objects associated to the gate_pin_objects. If a pin exists with the same name as the net, the net is returned using the command “get_nets -of_pins” and the “get_nets -of_gate_pins” command returns an empty collection. The command also returns an empty collection if the gate_pin is a port, in which case the command “get_nets -of_ports” returns the net you need. Refer to Example 7 in the get_pin command section for an example usage.

- -of_pseudo_ports *pseudo_port_objects*

  An optional switch and value pair that constrains the command to return the net objects associated to the pseudo_port_objects. If the pseudo port is defined on a pin, the pin is returned using the command get_pins -of_pseudo_ports while get_nets -of_pseudo_ports returns an empty collection. You can use this switch in combination with the -silent switch to determine if a pseudo_port was created on a net. Refer to Example 6 in the get_pin command section for an example usage.

- -bundle

  An optional switch that constrains the command to return net bundle objects. When specified, the tool traverses the bundle hierarchy of all root bundle objects declared in the selected view (current design or other views selected by -of_modules). If you do not specify name_patterns, the tool returns all root bundle objects. If the *name_patterns* is used (with -regexp or not), the results are all sub bundle objects or leaf objects match with the specified patterns. The “/”, “.”, and “[]” characters are the delimiters in the name pattern list. When the -regexp option is used, the “.” and “[]” characters, when used as delimiters, need to be escaped as those characters are meta characters in a regular expression.

  See “Bundle Object Introspection” in the *Tessent Shell User’s Manual* for complete information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -hierarchical

  An optional switch that specifies to perform pattern matching relative to all instances found below the current design. When this option is specified, the *name_patterns* argument cannot include any slash characters.

- -filter *attribute_equation*

  An optional switch and string that specify to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the net object type. See section "Attribute Filtering Equation Syntax" on page 3832 for more details on filtering attribute equation format. The =~ and !~ matching operators used simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option or not.

- -regexp

  An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a regular expression instead of as a simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for nets matching the *name_patterns*.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

  Note

  You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

Examples

Example 1

In the following example, the collection of all nets is filtered based on an attribute_equation of user-defined attributes.

```
get_nets -filter {MyType1 || myType2 == "AB"}
{u3_Y u4_O2}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

In the following example, the collection of all nets matching a list of hierarchical name patterns are returned. The names u1/* and u*/u* are only searched relative to the current design. See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for an example of pattern matching with regular expressions.

get_nets {u1/* u*/u*}

```
{u4/u4_Y u4/u1_Y u4/u2_Q u5/u4_Y u5/u1_Y u5/u2_Q}
```

Example 3

In the following example, the collection of all nets matching a leaf name pattern is returned. The leaf net name u4* searches for all nets found below the current design.

get_nets u4* -hierarchical

```
{u4_O1 u4_O2 u4_O3 u4/u4_Y u5/u4_Y}
```

Example 4

The following example returns the net bundle objects defined by the following:

```
typedef struct packed {
    logic f;
} struct_t;

typedef struct_t [1:0][1:1] array_t;
input array_t data_in [2:0]
```

get_nets

```
{data_in[2][1][1].f data_in[2][0][1].f data_in[1][1][1].f
data_in[1][0][1].f data_in[0][1][1].f data_in[0][0][1].f}
```

get_nets -bundle

```
{data_in}
```

get_nets data_in[2][*][1]

```
{data_in[2][1][1].f data_in[2][0][1].f}
```

get_nets data_in[2][*][1] -bundle

```
{data_in[2][1][1] data_in[2][0][1]}
```

get_nets data_in[2][*] -bundle

```
{data_in[2][1] data_in[2][0]}
```

get_nets data_in[*][*] -bundle

```
{data_in[2][1] data_in[2][0] data_in[1][1] data_in[1][0] data_in[0][1]
data_in[0][0]}
```

get_nets data_in[*] -bundle

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{data_in[2] data_in[1] data_in[0]}
```

get_nets data_in[2][*][1].* -bundle

```
{data_in[2][1][1].f data_in[2][0][1].f}
```

get_nets data_in[2][*][*].* -bundle

```
{data_in[2][1][1].f data_in[2][0][1].f}
```

get_bundle_objects -of_objects [get_nets data_in[2] -bundle]

```
{data_in[2][1] data_in[2][0]}
```

get_bundle_objects -of_objects [get_nets data_in[2] -bundle] -leaf

```
{data_in[2][1][1].f data_in[2][0][1].f}
```

get_bundle_objects -of_objects [get_nets data_in[2][*] -bundle] -leaf

```
{data_in[2][1][1].f data_in[2][0][1].f}
```

get_bundle_objects -of_objects [get_nets data_in[2][*] -bundle] -children

```
{data_in[2][1][1] data_in[2][0][1]}
```

get_bundle_objects -of_objects [get_nets data_in[2][*] -bundle] -parent

```
{data_in[2] data_in[2]}
```

get_bundle_objects -of_objects [get_nets data_in[2][*] -bundle] -root

```
{data_in data_in}
```

Related Topics

create_net

delete_nets

delete_pins

get_icl_ports

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_net_name_list

Context: unspecified, all contexts

Mode: all modes

Returns the design names of the associated port nets for a collection of ICL pins or ports.

Usage

get_net_name_list
-of_icl_object {***icl_port*** | ***icl_pin***}

Description

Use this command to get the design names of the associated port nets for a collection of ICL pins or ports. The result of this command depends on the tessent_design_port_nets attribute:

- **populated** — The get_net_name_list command returns the proper net name of the port net (or net names if the design port corresponding to the ICL port is a bus). The tessent_design_port_nets ICL port attribute is only populated when the name of the internal port net corresponding to the design port of the ICL port is different from the name of the design port.
- **unpopulated** — The get_net_name_list command returns the same value as the get_pin_name_list and get_port_name_list commands.

Arguments

- **-of_icl_object** ***icl_port*** | ***icl_pin***

  A required switch and value pair that specifies a collection of ICL pins or ports for which you require the names of the associated port nets. You can intermix ICL pins with ICL ports. The value the tool returns depends on the tessent_design_port_nets attribute.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_open_pattern_set

Context: patterns -ijtag

Mode: analysis

Returns the name of the currently open pattern set.

Usage

get_open_pattern_set

Description

Returns the name of the currently open pattern set.

This command returns an empty string if no pattern set is open.

Arguments

None

Return Values

The name of the currently open pattern set.

Examples

The following example prints the name of the currently open pattern set:

```
puts “The currently opened pattern set is called [get_open_pattern_set]”
```

Related Topics

close_pattern_set

get_pattern_set_list

open_pattern_set

report_pattern_sets

reset_open_pattern_set

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_pattern_cycle_count

Context: dft -edt, patterns -scan, patterns -scan_diagnosis, patterns -failure_mapping

Mode: analysis

Reports the total number of cycles in the currently loaded WGL or STIL test pattern set.

Usage

get_pattern_cycle_count

Description

Reports the total number of cycles in the currently loaded WGL or STIL test pattern set.

The get_pattern_cycle_count command reports the total number of cycles in the currently loaded test pattern set. The test pattern set must be STIL or WGL format.

Arguments

None

Examples

Example 1

The following example sets the pattern source to an external source, the *pat.wgl* file and displays the total number of cycles in the *pat.wgl* test pattern file:

```
read_patterns pat.wgl -wgl
get_pattern_cycle_count
```

Example 2

The following example sets the pattern source to an external pattern database source, the *pat1.wgl* file, appends the pat2.wgl test pattern file to it, and displays the total number of cycles contained in both test pattern files:

```
read_patterns pat1.wgl
read_patterns pat2.wgl -append
get_pattern_cycle_count   //reports total cycles for pat1 and pat2
```

Related Topics

read_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_pattern_set_data

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: analysis

Returns requested details of the internal representation of the specified pattern set.

Usage

get_pattern_set_data ***pattern_set_name*** [-unrolled] {-port_list | -event_list | -event_data *event_id*}

Description

Returns requested details of the internal representation of the specified pattern set.

The get_pattern_set_data command provides introspection into the internal representation of retargeted IJTAG patterns. Among other uses, this information can be used to generate pattern files for non-IJTAG compliant access mechanisms (for example, IEEE 1149.7), as long as the access hardware can produce the retargeted IJTAG patterns; that is, as long as the access mechanism does not put any restrictions on IJTAG retargeting.

Arguments

- ***pattern_set_name***

  A required string that specifies the pattern set from which to introspect the requested information.

  In the contexts “dft”, “patterns -scan”, “patterns -scan_diagnosis” and “patterns -scan_retargeting”, the pattern_set_name value can be “test_setup” and “test_end”, which represent the results of the retargeting of the corresponding test procedure iCalls.

  In the context “patterns -ijtag”, the pattern_set_name value can be any name used by open_pattern_set once it has been closed.

  The pattern_set_name value can be “ssn_setup” or “ssn_end” in the contexts “patterns -scan”, “patterns -scan_diagnosis”, and “patterns -scan_retargeting” when SSN is present in the design. Use these values after write_patterns to enable introspection into the SSN procedures.

- -unrolled

  An optional switch that specifies to restrict the information returned when the -event_list or -event_data switches are used. The -unrolled switch has no effect on the -port_list switch. For more information on using -unrolled with -event_list and -event_data, refer to the descriptions of those switches below.

- -port_list

  An optional switch that specifies to return the list of ports to which values are written or from which values are read in the specified pattern set.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -event_list

  An optional switch that specifies to return the list of all event identifiers of the pattern set.

  When the -event_list switch is used without the -unrolled switch, the returned list contains identifiers of events of type vector, scan, reset, loop, goto_state or nop.

  When the -event_list switch is used with the -unrolled switch, the returned list only contains identifiers of events of type vector, loop and nop. Events of type scan, reset and goto_state are unrolled or broken into the appropriate vector events.

  See "Event Types" below for details of all the event types.

- -event_data *event_id*

  An optional switch and string pair that specifies to return a list of key/value pairs of event data. *event_id* must be a string from the list returned by "get_pattern_set_data -event_list". You must first request the list of event identifiers from the pattern set, then you can use the event identifiers to get the pattern set data event by event.

  Note

  It is your responsibility to ensure that the -unrolled switch is used consistently. The event identifiers obtained by means of "get_pattern_set_data -event_list" only make sense in "get_pattern_set_data -event_data *event_id*", and the event identifiers obtained by means of "get_pattern_set_data -unrolled -event_list" only make sense in "get_pattern_set_data -unrolled -event_data *event_id*". As the *event_ids* are not guaranteed to be mutually exclusive, you must be careful to use -unrolled consistently.

  The return value of get_pattern_set_data pattern_set_name -event_data *event_id* is always a list with an even number of elements, in which the first, third, fifth ... elements represent keys and the second, fourth, sixth ... elements represent values. This convention enables the conversion into an associative Tcl array by means of the following command:

```
array set <variable name> <even-sized list>
```

  Tcl enables you to iterate on a list using several variables. This is useful to iterate on the results of get_pattern_set_data. For example, you can do the following:

```
foreach {key value} [get_pattern_set_data test -event_data] { … }
```

  Some of the key/value pairs are guaranteed to appear in the result, others appear only if the related values exist in the retargeted patterns. The entries in the tables are flagged as "mandatory" or "optional" to reflect this.

Event Types

- A vector event represents a single parallel vector, the stimulus data of the primary inputs (or bidirectional ports) of the design and the response data of primary outputs (or bidirectional ports).

  A scan event represents one scan load/unload.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

A reset event represents a global reset (such as would be happen, for example, by applying a ‘0’ to the TRST port of a TAP controller).

A loop event represents a certain idle time. The vector_count in the event data refers to tester periods, potentially rounded towards the next integer.

A goto_state event represents the transition from one TAP state to another one. The paths for those transitions are the same as the default paths for TAP state transitions in SVF.

A nop event (“no operation”) represents the absence of any action. It is only required to hold the information about notes (iNotes) that have been added after the last actual action.

The type of the event can be derived from the identifier names. An event of type vector has an identifier named vector number, where number is an increasing index. An event of type scan has an identifier named scan number and so on.

The following is an example of an event list:

```
{reset0 vector0 scan0 scan1 loop0 scan2 loop1 scan3 vector1}
```

Table 5-6. Events of Type “vector”

Key	Value
port_values (mandatory)	String consisting of the characters “U”, “D”, “0”, “1”, “X”, “Z”, “M”, “P” and “-”. The order is the same as in the port list (returned by get_pattern_set_data -port_list). The meaning of the characters is as follows: D expect 0 (hint: “down”) U expect 1 (hint: “up”) M expect Z (hint: “middle” or “median”) X do not expect any specific value L drive 0 (hint: “low”) H drive 1 (hint: “high”) Z drive Z P pulse (only appears in combination with TCKPorts or capture_shift_clocks) - used for ScanInPorts, which are not involved in a scan load D, U, X, L, H and Z are the same as in SVF. IJTAG adds P, M and - to the list of possible states.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-6. Events of Type "vector" (cont.)

Key	Value
write_var.port_list (optional)	Names of the primary input ports (or bidirectional ports), which the write_var.name_list and write_var.inversion_list specifications refer to. Each port can occur only once. The length of the write_var.port_list, write_var.name_list, and write_var.inversion_list lists is always the same. Use a for loop where the index goes from 0 to length-1 and lindex to extract the matching values from the three lists as shown in the example below. Refer to PDL Retargeting With Symbolic Variables in the *Tessent IJTAG User's Manual* for more information on the concept of iWriteVars.
write_var.name_list (optional)	Names of iWriteVars associated with the ports. One element per port, same order as in the write_var.port_list.
write_var.inversion_list (optional)	Specifies the inversion of the iWriteVars. The elements of the list can be 0 (no inversion) or 1 (inversion). The list must contain the same number of elements as the write_var.port_list.
read_var.port_list (optional)	Names of the primary output ports (or bidirectional ports), which the read_var.name_list and read_var.inversion_list specifications refer to. Each port can occur several times, because an output port can be used to observe several targets at the same time. The length of the read_var.port_list, read_var.name_list and read_var.inversion_list lists is always the same. Use a for loop where the index goes from 0 to *length*-1 and lindex to extract the matching values from the three lists as shown in the example later in this topic. There is one such entry per variable bit. A read_var is inferred for each iRead whether or not there was an iNote to name the iReadVar. When there is no specified tessent_pragma iReadVar to provide a name, the name of the ICL object that was read is used as the variable name. Refer to PDL Retargeting With Symbolic Variables in the *Tessent IJTAG User's Manual* for more information on the concept of iWriteVars.
read_var.name_list (optional)	Names of iReadVars associated with the ports. One element per port, same order as in the read_var.port_list.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-6. Events of Type "vector" (cont.)

Key	Value
read_var.inversion_list (optional)	Specifies the inversion of the iReadVars. The elements of the list can be 0 (no inversion) or 1 (inversion). The list must contain the same number of elements as the read_var.port list.
notes.text_list (optional)	List of notes preceding this event.
notes.type_list (optional)	List of the types of the notes. The list has the same number of elements as notes.text_list. The elements can be one of "user", "comment", "status", or "process". "user", "status" and "comment" notes are those that are generated as a result of the iNote command (with corresponding options). "process" notes are notes generated by the tool.

Table 5-7. Events of Type "scan"

Key	Value
port_values	All values except "-" have the same meaning as in "vector". In events of type "scan", the value "-" is used for all ScanInPorts. The actual scan load data can be found at <chain_id>.si_values. If the port_values contain expected values other than "X", this indicates that the comparison must be done in each cycle of the scan load.
write_var.port_list write_var.name_list write_var.inversion_list notes.text_list notes.type_list	Same as in "vector".
scan_interface (mandatory)	Name of the scan interface (as specified in ICL).
scan_type (mandatory)	"DR" or "IR". Always "DR" for designs without TAP.
start_state (mandatory)	Start state of the TAP controller. Can be one of IDLE, DRPAUSE, or IRPAUSE. For non-TAP designs, it can be IDLE or DRPAUSE. In this case, "DRPAUSE" means that there is no capture in this scan load, while "IDLE" means that there is a capture.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-7. Events of Type "scan" (cont.)

Key	Value
end_state (mandatory)	End state of the TAP controller. Can be one of IDLE, DRPAUSE, or IRPAUSE. For non-TAP designs, it can be IDLE or DRPAUSE. In this case, "DRPAUSE" means that there is no update in this scan load, while "IDLE" means that there is an update.
chain_list (mandatory)	List of identifiers of active chains. If the active ScanInterface does not contain Chain statements, the list consists of only one single element, which is called "default".
<chain_id>.length (mandatory)	Length of the scan chain.
<chain_id>.offset (mandatory)	The number of overshift cycles: that is, the difference between the number of shift cycles and the length of the chain.
<chain_id>.inversion (mandatory)	The chain inversion between ScanInPort and ScanOutPort.
<chain_id>.si_values (mandatory)	Scan-in value. A string consisting of "0" and "1" as wide as length+offset.
<chain_id>.so_values (mandatory)	Scan-out value. A string consisting of "0", "1" and "X" as wide as length+offset.
<chain_id>.write_var.cycle_list (optional)	List of cycles that the <chain_id>.write_var.name_list and <chain_id>.write_var.inversion_list refer to. The cycle corresponds to the location of the bit within the scan chain where cycle 0 is the closest to the scan out. The length of the <chain_id>.write_var.cycle_list, <chain_id>.write_var.name_list and <chain_id>.write_var.inversion_list lists is always the same. Use a for loop where the index goes from 0 to length-1 and lindex to extract the matching values from the three lists. There is one such entry per variable bit. A write_var is inferred for each iWrite when there was an iNote to name the iWriteVar. Refer to PDL Retargeting With Symbolic Variables in the *Tessent IJTAG User's Manual* for more information on the concept of iWriteVars.
<chain_id>.write_var.name_list (optional)	List of the names of the write_var where each bit of the variable is listed separately.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-7. Events of Type "scan" (cont.)

Key	Value
<chain_id>.write_var.inversion_list (optional)	List of inversions, consistent with cycle_list, analogously defined as write_var.inversion_list. **Note:** The inversions take the scan-in inversions into account, such that what has to be applied to the scan-in port can be derived directly. The length of the <chain_id>.read_var.cycle_list, <chain_id>.read_var.name_list, and <chain_id>.read_var.inversion_list lists is always the same. Use a for loop where the index goes from 0 to *length*-1 and lindex to extract the matching values from the three lists.
<chain_id>.read_var.cycle_list (optional)	List of cycles that the <chain_id>.read_var.name_list and <chain_id>.read_var.inversion_list refer to. The cycles correspond to the location of the bits within the scan chain where cycle 0 is the closest to the scan out. The length of the <chain_id>.read_var.cycle_list, <chain_id>.read_var.name_list, and <chain_id>.read_var.inversion_list lists is always the same. Use a for loop where the index goes from 0 to *length*-1 and lindex to extract the matching values from the three lists. There is one such entry per variable bit. A read_var is inferred for each iRead whether or not there was an iNote to name the iReadVar. When there is no specified tessent_pragma iReadVar to provide a name, the name of the ICL object that was read is used as the variable name. Refer to PDL Retargeting With Symbolic Variables in the *Tessent IJTAG User's Manual* for more information on the concept of iWriteVars.
<chain_id>.read_var.explicit_list (optional)	A string consisting of "0" and "1". A "1" indicates that an explicit iRead command uses the current tdo bit.
<chain_id>.read_var.name_list (optional)	List of the name of the read_var where each bit of the variable is listed separately.
<chain_id>.read_var.inversion_list (optional)	List of inversions, consistent with cycle_list, analogously defined as read_var.inversion_list. **Note:** The inversions take the scan-out inversions into account, such that what has to be expected at the scan-out port can be derived directly.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-7. Events of Type "scan" (cont.)

Key	Value
<chain_id>.register.name_list	The names of the ICL ScanRegisters in the correct order from scan-in to scan-out.
<chain_id>.register.length_list	The list of register lengths in the order from scan-in to scan-out. The elements in this list can be mapped uniquely to the elements in the list that is referenced by <chain_id>.register.name_list.

Table 5-8. Events of Type "reset"

Key	Value
notes.text_list notes.type_list	Same as in "vector".
sync (mandatory)	"On" or "Off". Specifies whether a synchronous reset shall be enforced.
type (mandatory)	"user" or "process". "user" indicates the usage of iReset, whereas "process" indicates a reset generated by the tool, for example on behalf of the open_pattern_set command (if used without no_initial_ireset).

Table 5-9. Events of Type "loop"

Key	Value
port_values write_var.port_list write_var.name_list write_var.inversion_list notes.text_list notes.type_list	Same as in "vector". If the port_values contain expected values other than "X", this indicates that the comparison must be done in each cycle of the loop.
type (mandatory)	"tck", "sck", or "time". Indicates the iRunLoop command type. For example, which switch (-tck, -sck, or -time) was used.
vector_count (mandatory)	Duration in terms of tester periods, rounded towards the next integer.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-10. Events of Type "goto_state"

Key	Value
port_values write_var.port_list write_var.name_list write_var.inversion_list notes.text_list notes.type_list	Same as in "vector". If the port_values contain expected values other than "X", this indicates that the comparison must be done in each cycle of the transition.
start_state (mandatory)	IDLE, DRPAUSE or IRPAUSE.
end_state (mandatory)	IDLE, DRPAUSE or IRPAUSE.

Table 5-11. Events of Type "nop"

Key	Value
notes.text_list notes.type_list	Same as in "vector".

Examples

The following example illustrates the usage of the get_pattern_set_data command.

```
SETUP> set_context patterns -ijtag -no_rtl
SETUP> read_icl data/icl/*.icl
SETUP> set_current_design chip
SETUP> add_clocks ClkA -period 10ns
SETUP> set_system_mode analysis
ANALYSIS> source data/pdl/raw1.iprocs
ANALYSIS> open_pattern_set test1 -tester_period 100ns
ANALYSIS> iCall block1_I1.raw1_I1.run_testa
ANALYSIS> close_pattern_set
ANALYSIS> get_pattern_set_data test1 -port_list

tck tdi tms trst tdo

ANALYSIS> get_pattern_set_data test1 -event_list

reset0 scan0 scan1 scan2 scan3 scan4 scan5 scan6 loop0 scan7

ANALYSIS> array set pd [get_pattern_set_data test1 -event_data scan5]
ANALYSIS> puts $pd(chain_list)

default

ANALYSIS> puts $pd(default.length)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
16
```

ANALYSIS> puts $pd(default.read_var.name_list)

```
block1_I1.raw1_I1.go block1_I1.raw1_I1.done
```

ANALYSIS> puts $pd(default.read_var.cycle_list)

```
7 8
```

ANALYSIS> puts $pd(default.so_values)

```
XXXXXXX00XXXXXXX
```

ANALYSIS> exit -f

Related Topics

close_pattern_set

open_pattern_set

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_pattern_set_list

Context: patterns -ijtag

Mode: analysis

Returns a Tcl list of all existing and closed pattern sets in the order of their creation.

Usage

get_pattern_set_list

Description

Returns a Tcl list of all existing and closed pattern sets in the order of their creation.

The currently open pattern set is not part of the list.

Arguments

None

Return Values

A Tcl list of all existing and closed pattern sets in the order of their creation.

Examples

The following example sets the patlist variable to the set of all existing pattern sets.

```
set patlist [get_pattern_set_list]
```

Related Topics

close_pattern_set

open_pattern_set

report_pattern_sets

reset_open_pattern_set

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_pattern_set_option

Context: patterns -ijtag

Mode: analysis

Returns the value of the requested option for the specified pattern_set_name.

Usage

get_pattern_set_option [*pattern_set_name*] {**-timeplate_name**
| **-tester_period** [-time_unit {s | ms | us | ns | ps | fs | as}]
| **-cycles** | **-tck_ratio** | **-initial_ireset** | **-active_scan_interfaces** | **-network_end_state** |
-tap_start_state | **-tap_end_state** | **-saved**}

Description

Returns the value of the requested option for the specified pattern_set_name.

Arguments

- *pattern_set_name*

  Specifies an optional pattern set name from which to introspect the specified option. By default, the tool introspects the currently open pattern set.

- **-timeplate_name**

  Returns the name of the optional timeplate that was used when opening the pattern set. The tool returns null if you did not specify the "open_pattern_set -timeplate" option. In this case, the tool uses the default timeplate. See open_pattern_set for a description of the default timeplate.

- **-tester_period** [-time_unit {s | ms | us | ns | ps | fs | as}]

  Returns the tester period with time units used by the pattern set.

  ```
  get_pattern_set_option my_patset -tester_period
  100ns
  ```

  The tool returns the tester period value when you specify the optional -time_unit switch and the value's time units.

  ```
  get_pattern_set_option my_patset -tester_period -time_unit ns
  100.0
  ```

- **-cycles**

  Returns the complete number of tester cycles required to run the pattern set on a tester.

- **-tck_ratio**

  Returns the tck_ratio value of the pattern set.

  The value of tck_ratio might depend on the maximum TCK clock frequency that you can apply to the circuit. See the "IJTAG Network Performance Optimization" section in the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tessent IJTAG User's Manual showing how to maximize the frequency of the IJTAG network test clock.

- **-initial_ireset**

  Returns 1 when the pattern set was opened without the -no_initial_ireset option; returns 0 when the option was used.

- **-active_scan_interfaces**

  Returns the names of the active scan interfaces.

- **-network_end_state**

  Returns keep, initial, or reset based on the value used with the "close_pattern_set -network_end_state" option.

- **-tap_start_state**

  Returns the expected TAP start state. The possible values are: "IDLE", "DRPAUSE", "IRPAUSE" or "any".

- **-tap_end_state**

  Returns the established TAP end state. The possible values are: "IDLE", "DRPAUSE", or "IRPAUSE". The tool reports an error message if you try to get the TAP end state of an open pattern set.

- **-saved**

  Returns 1 when the pattern set was written out using at least one write_patterns command. Otherwise, the tool returns 0.

Examples

In the following example, the total number of cycles for pattern set P1 is stored in the Tcl variable V1.

```
set V1 [get_pattern_set_option P1 -cycles]
```

Related Topics

close_pattern_set

open_pattern_set

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_pin_name_list

Context: all contexts

Mode: all modes

Returns the names of the associated design pins for a given collection of ICL pins.

Usage

get_pin_name_list **-of_icl_pins *icl_pins***

Description

Use this command to obtain the names of the associated design pins for a given collection of ICL pins. When the full view of the design is in memory, you can use get_pins -of_icl_pins to get the actual pin objects. However, if you only have a partial view of the design in memory, there may not be a pin object to return, but you may still be interested in finding the names of the design pins to feed to your SDC or simulation testbenches. If the design object is complex, such as a Verilog struct or system Interface port, the name differs between the RTL and synthesized view. The set_context -rtl / -no_rtl switch you used determines if the RTL or gate level name is returned. For example, the Tessent Shell tool uses this command to get the design name for ICL pins of type ClockPort to monitor those clock pins during simulation.

Arguments

- **-of_icl_pins *icl_pins***

  A required switch and value pair that specifies a collection of ICL pins for which the design names are required.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_pins

Context: all contexts

Mode: all modes

Returns a collection of all hierarchical pins instantiated relative to the current design that match the specified name_patterns list, subject to filtering by other options.

Usage

get_pins [*name_patterns*]
[-direction input | output | inout]
[-of_instances *instance_objects* | -below_instances *instance_objects*]
[-of_ports *port_objects*
| -of_nets *net_objects*
| -of_gate_pin *gate_pin_objects*
| -of_pseudo_ports *pseudo_port_objects*
| -of_icl_pins *icl_pin_objects*
| -of_icl_ports *icl_port_objects*]
[-bundle]
[-hierarchical]
[-use_path_matching_options]
[-filter *attribute_equation*]
[-regexp]
[-nocase]
[-silent]

Description

Returns a collection of all hierarchical pins instantiated relative to the current design. The pins can be filtered by different properties.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of pins. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no *name_patterns* are specified, the tool searches all pins.

  You can use the exact RTL instance path and port names here to find the equivalent pin objects on quick synthesis views. This enables you to load a DftSpecification with RTL instance paths in analysis mode if the current views are quick synthesized. Use only the full names of the leaf objects. Do not use the bundle objects unless you load the DftSpecification with RTL views.

  This argument cannot be specified with either the -of_nets or -of_gate_pin arguments.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following limitations apply:

- You can use wildcards in *name_patterns*, but you cannot use a wildcard as part of the left or right index of a range. For example: [* : <right_index>] is not supported.
- A range (a dimension of the form [<left_index> : <right_index>]) can only be used in the last dimension of the last field of the *name_patterns*.

See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a complete description of the pattern syntax when the -regexp option is used and not used.

Tip

If your *name_patterns* string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- -direction input | output | inout

  An optional switch and literal pair that specifies to filter results based on the value of the direction attribute on the pin object. When omitted, pins with any direction are considered.

- -of_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for pins on the specified *instance_objects*. When this option is specified, the optional *name_patterns* must be the name pattern of a leaf that is checked against the leaf name of the pins. An error is generated if the specified *name_patterns* contain more than one level of hierarchy when the -of_instances option is used. This method is more direct and efficient than traversing the *module_name* attribute because it automatically handles libraries and architectures.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for pins below the specified *instance_objects*. Specified *name_patterns* are searched relative to the specified *instance_objects* when this option is used.

  If the specified *instance_objects* is an empty collection, all instances below the current design are considered. See the description of the "get_instances -below_instances" command description for more information.

- -of_ports *port_objects*

  An optional switch and value pair that constrains the command to return a collection of pins associated to a collection of ports. The port objects must be associated to sub-modules instantiated at any level below the current design. Ports of the current design do not have associated pins. This switch finds all instances of the associated sub-modules, then finds the pins on those instances with a *leaf_name* matching the supplied port names.

  The value of this switch cannot be a Tcl list of port names because those names would only be searched on the current design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Use the "get_ports -of_module" command to create a collection of port objects. Pass the collection to the "get_pins -of_ports" command to map the ports to pins. Use a command as shown below to map a set of ports on a sub-module to their corresponding pin objects:

```
get_pins -of_ports [get_ports {p1 p2} -of_mod ABC]
```

Using a construct such as "get_pins -of_ports {p1 p2}" does not work because there are no pins associated to ports of the current design.

For an example of using this switch, see the description of the "set_insertion_options" command.

- -of_nets *net_objects*

  An optional switch and value pair that constrains the command to return the pin objects associated to *net_objects*. You cannot use a *name_patterns* list when using this option.

  More than one pin can be returned for a specified net:

```
module subA (
  .A({a2, a1, a2}),
  .B({b3, b2, b1, b0})
);
  input  a1;
  input  a2;
  output b3, b2, b1, b0;
endmodule

module top  (A, B);
  input [2:0] A;
  output [3:0] B;
  subA u_A (.A(A), .B(B));
endmodule
```

  In this example, the net u_A/a2 is associated with pins u_A/A[2] and u_A/A[0]:

```
SETUP> get_pins -of_nets {u_A/a2 u_A/b2}
{u_A/A[0] u_A/A[2] u_A/B[2]}
```

- -of_gate_pin *gate_pin_objects*

  An optional switch and value pair that constrains the command to return the pin objects associated with the specified *gate_pin_objects*. Given a collection or list of gate_pin objects, the gate_pin objects are mapped to the corresponding hierarchical pin objects. Any gate_pin object not having an associated hierarchical pin object is discarded. You cannot use a *name_patterns* list when using this option.

- -of_pseudo_ports *pseudo_port_objects*

  An optional switch and value pair that constrains the command to return the design pins associated with the specified pseudo-ports (user-added primary inputs or outputs).

- -of_icl_pins *icl_pin_objects*

  An optional switch and value pair that constrains the command to return the pin objects associated to the specified *icl_pin_objects*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the ICL attributes *tessent_design_gate_ports* and *tessent_design_rtl_ports* are defined for the ICL port of the given ICL pin, they are used to find the equivalent design pin. The value of the attributes *tessent_design_gate_ports* and *tessent_design_rtl_ports* are empty if they are the same as the ICL port name.

If you use this switch in -no_rtl mode after real logic synthesis, and the design footprint has changed (for example, due to bit-blasting), the value of the attribute *tessent_design_gate_ports* does not match the design objects in the current design.

The gate name mapping function finds (as with the -use_path_matching_option) the new equivalent port and updates the tessent_design_gate_ports attribute with the new value. See "Post-Synthesis Update" in the *Tessent Shell User's Manual* for a description of this function. This attribute assignment persists, and the design pin can be successfully found with subsequent calls to get_pins -of_icl_pins with the same ICL pin. See the add_rtl_to_gates_mapping command to configure the name mapping.

In the general scenario (-no_rtl mode), the attribute tessent_design_gate_ports is updated during set_current_design.

- -of_icl_ports *icl_port_objects*

  An optional switch and value pair that constrains the command to return the pin objects associated with the specified *icl_port_objects*. If you know the design instance associated with the specified pin objects, use the -of_instances or -below_instances switch. The ICL port objects' association to the design pins is determined by finding an instance of the design module matching the ICL module of the ICL port.

  When using the -of_instances or -below_instances options, the specified ICL ports must all exist in a single ICL module. Be certain that the ICL module of the specified ICL ports is associated with the modules of the specified instances. For computational efficiency, this association is assumed and is not verified by the tool.

  The searching behavior described for the -of_icl_pins option is applied here to find the equivalent design port.

- -bundle

  An optional switch that constrains the command to return pin bundle objects. When specified, the tool traverses the bundle hierarchy of all root bundle objects declared in the selected view (current design or other views selected by -of_modules). If you do not specify name_patterns, the tool returns all root bundle objects. If the *name_patterns* is used (with -regexp or not), the results are all sub bundle objects or leaf objects match with the specified patterns. The "/", ".", and "[]" characters are the delimiters in the name pattern list. When the -regexp option is used, the "." and "[]" characters, when used as delimiters, need to be escaped as those characters are meta characters in a regular expression.

  See "Bundle Object Introspection" in the *Tessent Shell User's Manual* for complete information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -hierarchical

  An optional switch that specifies to perform pattern matching relative to all instances found below the current design. When this option is specified, the *name_patterns* argument cannot include more than one slash.

  Using "get_pins *name_pattern* -hierarchical" is a very expensive operation. Avoid using this operation on large designs. See Example 8 for a better way to achieve the same result.

- -use_path_matching_options

  An optional switch that performs pin name-mapping using rules you specified for RTL to post-synthesis/post-layout name mapping. Using this option enables you to find a pin using the instance and port mapping rules.

  The following procedure is used to map the pin name: First, an attempt is made to look up the name as-is. If this is unsuccessful, an empty collection is created if the design is a quick synthesis view and "set_context *context* -rtl_mode" is specified or inferred. Otherwise, if in -no_rtl mode (after real logic synthesis), all default matching rules (instance path and port name) in the list of matching rules are applied in turn until a match is found or the rules are exhausted. User-specified name pairs are checked first, followed by the default mapping strategy, followed by any user-specified mapping expressions.

  Note

  User-specified rules are checked only for instance paths, not for port names.

  This switch is used to find the equivalent pins after logic synthesis using the original RTL names or initial post-synthesis names. The name_patterns parameter cannot contain wildcards if this switch is used. If a slice is specified in no RTL mode, the returned value is a Tcl collection of multiple pin objects.

  See add_rtl_to_gates_mapping for complete information.

- -filter *attribute_equation*

  An optional switch and string pair that specifies to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the pin object type. See section "Attribute Filtering Equation Syntax" on page 3832 for more details on filtering attribute equation format. The =~ and !~ matching operator use simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option.

- -regexp

  An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a regular expression instead of as a simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for pin names matching *name_patterns*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

  Note

  You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the set_tcl_shell_options -change_no_result_warnings_to_errors switch description for detailed usage.

Examples

Example 1

In the following example, the collection of all pins is filtered based on an attribute_equation of user-defined attributes.

get_pins -filter {myType1 || myType2 == "ABC"}

```
{u1/Y u4/u4/A0 u7/D}
```

Example 2

In the following example, the collection of all pins matching a list of hierarchical name patterns is returned. The names u1/Y and u*/Q are only searched relative to the current design. See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for examples of pattern matching with regular expressions.

get_pins {u1/Y u*/Q}

```
{u1/Y u2/Q u7/Q}
```

Example 3

In the following example, the pins with leaf_name matching Q* and found on instances of module DFF are returned

```
get_pins Q* -of_instances [get_instances -of_modules DFF]
{u2/reg1/Q u2/reg1/QB u3/reg5/Q}
```

Example 4

In the following example. the collection of all pins matching a leaf instance_name/leaf pin name pattern is returned. The tool searches for the leaf instance name u* in all instances found below the current design, and then searches for the leaf pin names matching Y* on those instances.

get_pins u*/Y* -hierarchical

```
{u1/Y u3/Y u4/u1/Y u4/u4/Y u4/u5/u1/Y u5/u1/Y u5/u3/Y u5/u4/Y u6/Y}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 5

This example return the pin names of the specified pseudo_port objects:

```
get_pins -of_pseudo_ports {user_pi1 user_pi2}
```

```
{u1/Y u3/Y u4/Y}
```

Example 6

This example shows a small proc used to get the net or pin on which the pseudo_port was created:

```
proc get_objects_from_pseudo_ports {pseudo_port} {
  set objects {}
  append_to_collection objects [get_pins -of_pseudo_ports $pseudo_port
    -silent]
  append_to_collection objects [get_nets -of_pseudo_ports $pseudo_port
    -silent]
return $objects
}
```

Example 7

This example shows a small proc used to get the net, pin, or port associated to a gate_pin:

```
proc get_objects_from_gate_pins {gate_pin} {
  set objects {}
  append_to_collection objects [get_pins  -of_gate_pin $gate_pin -silent]
  append_to_collection objects [get_ports -of_gate_pin $gate_pin -of_type
      all -silent]
  append_to_collection objects [get_nets  -of_gate_pin $gate_pin -silent]
  return $objects
}
```

Example 8

This example shows how to find all pins on instances of cell modules matching a given name list. One could have used the following:

```
get_pins -hierarchical {VSS VDD}
```

but this operation would be very slow on a large design given that every pin of every instance needs to be visited. Instead, use the following example:

```
foreach_in_collection cell_mod [get_modules -of_type cell] {
  set ports [get_ports {VSS VDD} -of_mod $cell_mod -silent]
  if {[sizeof_collection $ports] > 0} {
    set inst [get_instances -of_modules $cell_mod -silent]
    append_to_collection pins [get_pins {VSS VDD} -of_inst $inst -silent]
   }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 9

This example returns the pin bundle objects defined by the following:

```
typedef struct packed {
    logic f;
} struct_t;

typedef struct_t [1:0][1:1] array_t;
input array_t data_in [2:0]
```

get_pins

```
{u1/data_in[2][1][1].f u1/data_in[2][0][1].f u1/data_in[1][1][1].f u1/
data_in[1][0][1].f u1/data_in[0][1][1].f u1/data_in[0][0][1].f}
```

get_pins -bundle

```
{u1/data_in}
```

get_pins u1/data_in[2][*][1]

```
{u1/data_in[2][1][1].f u1/data_in[2][0][1].f}
```

get_pins u1/data_in[2][*][1] -bundle

```
{u1/data_in[2][1][1] u1/data_in[2][0][1]}
```

get_pins u1/data_in[2][*] -bundle

```
{u1/data_in[2][1] u1/data_in[2][0]}
```

get_pins u1/data_in[*][*] -bundle

```
{u1/data_in[2][1] u1/data_in[2][0] u1/data_in[1][1] u1/data_in[1][0] u1/
data_in[0][1] u1/data_in[0][0]}
```

get_pins u1/data_in[*] -bundle

```
{u1/data_in[2] u1/data_in[1] u1/data_in[0]}
```

get_pins u1/data_in[2][*][1].* -bundle

```
{u1/data_in[2][1][1].f u1/data_in[2][0][1].f}
```

get_pins u1/data_in[2][*][*].* -bundle

```
{u1/data_in[2][1][1].f u1/data_in[2][0][1].f}
```

get_bundle_objects -of_objects [get_pins u1/data_in[2] -bundle]

```
{u1/data_in[2][1] u1/data_in[2][0]}
```

get_bundle_objects -of_objects [get_pins u1/data_in[2] -bundle] -leaf

```
{u1/data_in[2][1][1].f u1/data_in[2][0][1].f}
```

get_bundle_objects -of_objects [get_pins u1/data_in[2][*] -bundle] -leaf

```
{u1/data_in[2][1][1].f u1/data_in[2][0][1].f}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_bundle_objects -of_objects [get_pins u1/data_in[2][*] -bundle] -children

```
{u1/data_in[2][1][1] u1/data_in[2][0][1]}
```

get_bundle_objects -of_objects [get_pins u1/data_in[2][*] -bundle] -parent

```
{u1/data_in[2] u1/data_in[2]}
```

get_bundle_objects -of_objects [get_pins u1/data_in[2][*] -bundle] -root

```
{data_in data_in}
```

Example 10

Given the following definition:

```
typedef struct packed {
      logic [3:0] in;
      logic [1:0] out;
} EDT_t;
Input EDT_t edt_channel;
```

To load the DftSpecification before quick synthesis, use the following:

```
EdtChannelsIn(8:5) {
  port_pin_name : edt_channel.in;
}
// In Setup before quick synthesis you can
read_config_data ../edt.config
process_dft_spec EDT_setup
```

To load the DftSpecification after quick synthesis, you must write the configuration as follows:

```
EdtChannelsIn(8:5) {
  port_pin_name : edt_channel.in[3], edt_channel.in[2], edt_channel.in[1],
edt_channel.in[0];
}
```

After quick synthesis, you can use the RTL leaf objects' full names to return the equivalent pin or port objects:

get_pins B1[1].B2.u1/data_in[5].ask[4].rsq

```
B1[1].B2.u1/data_in[31]
```

get_pins B1[1].B2.u1/data_in[31]

```
B1[1].B2.u1/data_in[31]
```

get_ports edt_channel.in

```
// (error)
```

get_ports edt_channel.in[2]

```
edt_channel[4]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 11

Examples of calling get_pins -of_icl_ports:

Table 5-12. Calling get_pins -of_icl_ports in RTL Mode

In RTL mode (RTL view):
`SETUP> get_pins -of_icl_ports \`
`  data_in_full[35] \`
`  -of_instances u1/u2`
`  {{u1/u2/\data_in&full [2][1].b}}`
	From original RTL (extract_icl in RTL mode):
	`Module corea {`
	`  // Created by ICL extraction`
	`  DataInPort data_in_full[35] {`
	`  Attribute tessent_design_gate_ports = \`
	`  "{\data_in&full [35]}";`
	`  Attribute tessent_design_rtl_ports = \`
	`  "{\data_in&full [2][1].b}";`
	`  }`
	`  ...`
	`}`

In RTL mode (quick synthesis view):
`SETUP> get_pins -of_icl_ports \`
`  data_in_full[35] \`
`  -of_instances u1/u2`
`  {{u1/u2/\data_in&full [35]}}`	

Table 5-13. Calling get_pins -of_icl_ports in No RTL Mode After Synthesis

`SETUP> get_pins -of_icl_ports \`
`  data_in_full[35] \`
`  -of_instances u1/u2`
`  {u1/u2/data_in_full_35_}`
	After logic synthesis:
	`Module corea {`
	`  // Created by ICL extraction`
	`  DataInPort data_in_full[35] {`
	`  Attribute tessent_design_gate_ports = \`
	`  "{\data_in_full_35_}";`
	`  Attribute tessent_design_rtl_ports = \`
	`  "{\data_in&full [2][1].b}";`
	`  }`
	`  ...`
	`}`

Example 12

Examples of calling get_pins -use_path_matching_options:

Table 5-14. Calling get_pins -use_path_matching_options in RTL Mode

In RTL mode (RTL view)
`SETUP> get_pins {{u1/loop[1].u2/\data_in&full [2][1].b}} \`
`  -use_path_matching_options`
`{{ u1/loop[1].u2/\data_in&full [2][1].b}}`

In RTL mode (quick synthesis view)
`SETUP> get_pins {{u1/\loop[1].u2 /\data_in&full [35]}}`
`-use_path_matching_options`
`{{ u1/\loop[1].u2 /\data_in&full [35]}}`

Table 5-15. Calling get_pins -use_path_matching_options in No RTL Mode After Synthesis

```
SETUP> get_pins {{u1/loop[1].u2/\data_in&full [35]}} \
    -use_path_matching_options
{u1/loop_1__u2/data_in_full_35_}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The RTL name cannot be used for name_patterns in No RTL mode if it includes flattened complex ports. See the following example:

Table 5-16. Calling get_pins -use_path_matching_options in No RTL Mode After Synthesis (Flattened Complex Ports)

```
SETUP> get_pins {{u1/loop[1].u2/\data_in&full [2][1].b}} \
    -use_path_matching_options
[no results]
```

Table 5-17. Calling get_pins -use_path_matching_options in No RTL Mode After Synthesis With Bit-Blasting

```
SETUP> get_pins {{u1/loop[1].u2/\data_in&full [2][1].b}} \
    -use_path_matching_options
{{u1/\loop[1].u2 /\data_in&full[2][1].b }}
```

Table 5-18. Calling get_pins -use_path_matching_options in No RTL Mode After Synthesis with Bit-Blasting (Special Characters Replaced by Underscores)

```
SETUP> get_pins {{\data_in&full [2][1].b}} -use_path_matching_options \
{u1/loop_1__u2/data_in_full_2_1__b }}
```

Related Topics

add_rtl_to_gates_mapping

create_pin

delete_pins

get_common_parent_instance

get_icl_ports

get_modules

get_nets

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_port_name_list

Context: all contexts

Mode: all modes

Returns the names of the associated design ports for a collection of ICL ports.

Usage

get_port_name_list **-of_icl_ports *icl_ports*** [-use_active_name]

Description

Use this command to obtain the names of the associated design ports for a given collection of ICL ports. When the full view of the design is in memory, you can use get_ports -of_icl_ports to get the actual port objects. However, if you only have a partial view of the design in memory, there may not be a port object to return, but you may still be interested in finding the names of the design ports to feed to your SDC or simulation testbenches. If the design object is complex, such as a Verilog struct or system Interface port, the name differs between the RTL and synthesized view. The set_context -rtl / -no_rtl switch you used determines if the RTL or gate level name is returned. For example, the Tessent Shell tool uses this command to get the design name for ICL pins of type ClockPort to monitor those clock pins during simulation.

Arguments

- **-of_icl_ports *icl_ports***

  A required switch and value pair that specifies a collection of ICL ports for which the design names are required.

- -use_active_name

  An optional switch that returns the active names. In RTL View, it returns the tessent_design_rtl_ports value. In quick synthesis view, it returns the tessent_design_gate_ports value. Use this switch only in RTL mode.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_ports

Context: all contexts

Mode: all modes

Returns a collection of all hierarchical ports on the current design or a given module that match the specified filters.

Usage

get_ports [*name_patterns*]
[-of_modules *module_objects*]
[-of_nets *net_objects*
| -of_pins *pin_objects*
| -of_gate_pins *gate_pin_objects*
| -of_icl_ports *icl_port_objects*
| -of_wrapper_scan_elements *scan_element_spec*]
[-bundle]
[-direction input | output | inout]
[-of_type real | pseudo | all]
[-use_path_matching_options]
[-filter *attribute_equation*]
[-regexp]
[-nocase]
[-silent]

Description

Returns a collection of all hierarchical ports on the current design or a given module that match the specified *name_patterns* list (default: all ports) and other options.

Note

Ports are not properly returned when the module has a parameter that specifies the size of the port, but the module is not instantiated in the design.

You can have any number of regular expressions in the name patterns list.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies the names of the ports to search and return. If no *name_patterns* are specified, the tool searches all ports in the current design.

  You can use the exact RTL port names here to find the equivalent port objects on quick synthesis views. This enables you to load a DftSpecification with RTL port names in analysis mode if the current views are quick synthesized. Use only the leaf objects' full names. Do not use the bundle objects unless you load the DftSpecification with RTL views.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following limitations apply:

- You can use wildcards in *name_patterns*, but you cannot use a wildcard as part of the left or right index of a range. For example: [* : <right_index>] is not supported.
- A range (a dimension of the form [<left_index> : <right_index>]) can only be used in the last dimension of the last field of the *name_patterns*.

See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a complete description of the pattern syntax when the -regexp option is used and not used.

- -of_modules *module_objects*

  An optional switch and value pair that constrains the command to return only the ports on the modules specified by *module_objects*.

- -of_nets *net_objects*

  An optional switch and value pair that constrains the command to return the port objects associated to net_objects. You cannot use a *name_patterns* list when using this option.

  More than one port can be returned for a specified net:

```
module top (
  .A({a2, a1, a2}),
  .B({b3, b2, b1, b0})
);
  input  a1;
  input  a2;
  output b3, b2, b1, b0;
endmodule
```

  In this example, the net a2 is associated with ports A[2] and A[0]:

```
SETUP> get_ports -of_nets {a2 b2}
{A[0] A[2] B[2]}
```

- -of_pins *pin_objects*

  An optional switch and value pair that constrains the command to return the ports defined on the modules that correspond to the pins specified by *pin_objects*.

  You can further constrain the command to return only pseudo_ports or only real Ports using "-of_type pseudo | real"; if you do not specify the -of_type switch, by default, the tool returns all real ports (-of_type real). If you use this option in conjunction with "-of_type pseudo", the tool maps the pin that was added as part of the pseudo_port (user-added PI/PO [primary in put or output) to the name of the associated pseudo_port. See "Example 5" on page 1199. If you added a user-specified primary input or output without specifying the pin name using "add_primary_inputs -pseudo_port_name," the name of the pseudo_port may not always match the name of the pin specified; in this case, you can use the -of_pins switch to find the name of the associated pseudo_port.

  You cannot use this switch with the "-of_type all" option. You also cannot use this switch with a *name_patterns* list.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -of_gate_pins *gate_pin_objects*

  An optional switch and value pair that constrains the command to return the ports associated to the gate_pin objects. Any gate_pin object not associated to a port object on the current design is ignored. You cannot use a *name_patterns* list when using this option.

- -of_icl_ports *icl_port_objects*

  An optional switch and value pair that constrains the command to return the ports defined on a design module associated with the ICL modules of the specified *icl_port_objects*. If you know the design module associated with the specified ICL module, use the -of_modules switch.

  When specifying the -of_modules switch with the -of_icl_ports switch, the specified ICL ports must all be on a single ICL module. When using these options, be certain that the ICL module of the specified ICL ports is associated with the specified design modules. This association is assumed and is not verified by the tool.

  If the ICL attributes *tessent_design_gate_ports* and *tessent_design_rtl_ports* are defined for the ICL port of the given ICL pin, they are used to find the equivalent design port. The values of the attributes *tessent_design_gate_ports* and *tessent_design_rtl_ports* are empty if they are the same as the ICL port name.

  If this switch is used in no RTL mode after logic synthesis and the design footprint has changed (for example, due to bit-blasting), the value of the attribute *tessent_design_gate_ports* does not match the design objects in the current design.

  The gate name mapping function is used to find the new equivalent port and update the tessent_design_gate_ports attribute with the new value. See "Post-Synthesis Update" in the *Tessent Shell User's Manual* for a description of this function. Subsequent calls to the "get_ports -of_icl_ports" command with the same ICL port objects find the design ports because this attribute assignment persists.

  In the typical scenario (no RTL mode), the attribute *tessent_design_gate_ports* is updated during set_current_design.

  When using the -of_icl_ports option without the -of_modules option, the search algorithm is applied on the current design module. If the -of_modules option is used with -of_icl_ports, only one module should be specified.

- -of_wrapper_scan_elements *scan_element_spec*

  An optional switch and value pair that returns a collection of port objects isolated by the supplied scan_element objects. You can use this switch for dedicated and shared wrapper cells.

  Note

  This switch is only available in dft -scan context in analysis mode.

- -bundle

  An optional switch that constrains the command to return port bundle objects. When specified, the tool traverses the bundle hierarchy of all root bundle objects declared in the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

selected view (current design or other views selected by -of_modules). If you do not specify name_patterns, the tool returns all root bundle objects. If the *name_patterns* is used (with -regexp or not), the results are all sub bundle objects or leaf objects match with the specified patterns. The "/", ".", and "[]" characters are the delimiters in the name pattern list. When the -regexp option is used, the "." and "[]" characters, when used as delimiters, need to be escaped as those characters are meta characters in a regular expression.

See "Bundle Object Introspection" in the *Tessent Shell User's Manual* for complete information.

- -direction input | output | inout

  An optional switch and literal pair that specifies to filter results based on the value of the direction attribute on the port object. When omitted, ports with any direction are considered.

- -of_type real| pseudo | all

  An optional switch and literal pair that specifies to filter results based on the specified type of the port object. When omitted, only real design ports are considered. The option types have the following meaning:

  real — Only real design ports (Port object type) are included in the returned collection.

  pseudo — Only user-added pseudo ports (pseudo_port object type) are included in the returned collection.

  all — Both real (design) and pseudo (user-added) ports are included in the returned collection.

- -use_path_matching_options

  An optional switch that uses a list of name mapping rules for post-synthesis and post-layout name mapping. Using this switch enables you to find a port using the port mapping rules.

  The following method is used to map the port name: First, an attempt is made to look up the name as-is. If this is unsuccessful, an empty collection is created if the design is a quick synthesis view and "set_context *context* -rtl_mode" is specified or inferred. Otherwise, if in -no_rtl mode (after logic synthesis), all default port name matching rules are applied until a match is found or the rules are exhausted.

  Note

  User-specified rules are verified only for instance paths and not for port names.

  This switch is used to find the equivalent design ports after logic synthesis using the original RTL names or the post-synthesis_name attribute in RTL mode. For complex ports, the RTL name is always a multidimensional structure. After logic synthesis, the ports are transformed to one-dimensional bus or scalar data types.

  Ports in the SystemVerilog interface are blasted into several ports. The name_patterns argument cannot contain wildcards if this switch is used. If a slice is specified in no RTL mode, the result is a Tcl collection of multiple port objects.

  See add_rtl_to_gates_mapping for complete information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -filter *attribute_equation*

  An optional switch and string that specifies to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the port object type. See "Attribute Filtering Equation Syntax" on page 3832 for more details on filtering attribute equation format. The =~ and !~ matching operators use simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option.

- -regexp

  An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a real regular expression instead of as a simple wildcard pattern. See "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for ports matching the *name_patterns*.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the *name_patterns* disappeared due to a change in a previous editing step. This -silent option has no effect on any other error reporting.

  Note

  You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the "set_tcl_shell_options -" switch description for detailed usage.

Examples

Example 1

This example looks for ports on the current design whose name match I* or O*

get_ports {I* O*}

```
{I1 I2 I3 I4 I5 I6 I7 I8 O1 O2 O3}
```

Example 2

This example looks for scalar ports having one or two characters on module ModA.

get_ports {[[:alnum:]]{1,2}} -regexp -of_modules ModA

```
{ab a b}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

This example return ports based on the option specified for the -of_type switch:

add_primary_input -internal inst1/Y inst1/Z -pseudo_port_name user_pi1
add_primary_input -internal inst2/Y -pseudo_port_name user_pi2
get_ports

```
{port1 port2 port3}
```

get_ports -of_type real

```
{port1 port2 port3}
```

get_ports -of_type pseudo

```
{user_pi1 user_pi2}
```

get_ports -of_type all

```
{port1 port2 port3 user_pi1 user_pi2}
```

get_pins -of_pseudo_ports user_pi1

```
{inst1/Y inst1/Z}
```

get_pins -of_pseudo_ports user_pi2

```
{inst2/Y}
```

get_pins -of_pseudo_ports {user_pi1 user_pi2}

```
{inst1/Y inst1/Z inst2/Y}
```

Example 4

This example returns the pseudo_ports associated with the user-specified primary inputs and primary outputs based on the option specified for the -of_type switch:

add_primary_input -internal inst1/Y inst1/Z -pseudo_port_name user_pi1
add_primary_input -internal inst2/Y

get_ports -of_pins inst1/Y

```
{Y}
```

get_ports -of_pins {inst1/Y inst1/Z} -of_type real

```
{Y Z}
```

get_ports -of_pins inst1/Y -of_type pseudo

```
{user_pi1}
```

get_ports -of_pins inst2/Y -of_type real

```
{Y}
```

get_ports -of_pins inst2/Y -of_type pseudo

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{inst2/Y}
```

get_ports -of_pins {inst1/Y inst1/Z inst2/Y} -of_type pseudo

```
{user_pi1 user_pi1 inst2/Y}
```

Example 5

This example returns the port bundle objects defined by the following:

```
typedef struct packed {
    logic f;
} struct_t;

typedef struct_t [1:0][1:1] array_t;
input array_t data_in [2:0]
```

get_ports

```
{data_in[2][1][1].f data_in[2][0][1].f data_in[1][1][1].f
data_in[1][0][1].f data_in[0][1][1].f data_in[0][0][1].f}
```

get_ports -bundle

```
{data_in}
```

get_ports data_in[2][*][1]

```
{data_in[2][1][1].f data_in[2][0][1].f}
```

get_ports data_in[2][*][1] -bundle

```
{data_in[2][1][1] data_in[2][0][1]}
```

get_ports data_in[2][*] -bundle

```
{data_in[2][1] data_in[2][0]}
```

get_ports data_in[*][*] -bundle

```
{data_in[2][1] data_in[2][0] data_in[1][1] data_in[1][0] data_in[0][1]
data_in[0][0]}
```

get_ports data_in[*] -bundle

```
{data_in[2] data_in[1] data_in[0]}
```

get_ports data_in[2][*][1].* -bundle

```
{data_in[2][1][1].f data_in[2][0][1].f}
```

get_ports data_in[2][*][*].* -bundle

```
{data_in[2][1][1].f data_in[2][0][1].f}
```

get_bundle_objects -of_objects [get_ports data_in[2] -bundle]

```
{data_in[2][1] data_in[2][0]}
```

get_bundle_objects -of_objects [get_ports data_in[2] -bundle] -leaf

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{data_in[2][1][1].f data_in[2][0][1].f}
```

get_bundle_objects -of_objects [get_ports data_in[2][*] -bundle] -leaf

```
{data_in[2][1][1].f data_in[2][0][1].f}
```

get_bundle_objects -of_objects [get_ports data_in[2][*] -bundle] -children

```
{data_in[2][1][1] data_in[2][0][1]}
```

get_bundle_objects -of_objects [get_ports data_in[2][*] -bundle] -parent

```
{data_in[2] data_in[2]}
```

get_bundle_objects -of_objects [get_ports data_in[2][*] -bundle] -root

```
{data_in data_in}
```

Example 6

Given the following definition:

```
typedef struct packed {
      logic [3:0] in;
      logic [1:0] out;
} EDT_t;
Input EDT_t edt_channel;
```

To load the DftSpecification before quick synthesis, use the following:

```
EdtChannelsIn(8:5) {
  port_pin_name : edt_channel.in;
}
// In Setup before quick synthesis you can
read_config_data ../edt.config
process_dft_spec EDT_setup
```

To load the DftSpecification after quick synthesis, you must write the configuration as follows:

```
EdtChannelsIn(8:5) {
  port_pin_name : edt_channel.in[3], edt_channel.in[2], edt_channel.in[1],
edt_channel.in[0];
}
```

After quick synthesis, you can use the RTL leaf objects' full names to return the equivalent pin or port objects:

get_pins B1[1].B2.u1/data_in[5].ask[4].rsq

```
B1[1].B2.u1/data_in[31]
```

get_pins B1[1].B2.u1/data_in[31]

```
B1[1].B2.u1/data_in[31]
```

get_ports edt_channel.in

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// (error)
```

get_ports edt_channel.in[2]

```
edt_channel[4]
```

Example 7

Examples of calling get_ports -of_icl_ports:

Table 5-19. Calling get_ports -of_icl_ports in RTL Mode

In RTL mode (RTL view): `SETUP> get_ports -of_icl_ports \ data_in_full[35] \ {{\data_in&full [2][1].b}}`	From original RTL (extract_icl in RTL mode): `Module top { // Created by ICL extraction DataInPort data_in_full[35] { Attribute tessent_design_gate_ports = \ "{\data_in&full [35]}"; Attribute tessent_design_rtl_ports = \ "{\data_in&full [2][1].b}"; } ... }`
In RTL mode (quick synthesis view): `SETUP> get_ports -of_icl_ports \ data_in_full[35] \ {{\data_in&full [35]}}`	

Table 5-20. Calling get_ports -of_icl_ports in No RTL Mode After Synthesis

In No RTL mode: `SETUP> get_ports -of_icl_ports \ data_in_full[35] \ {data_in_full_35_}`	Updated after logic synthesis: `Module top { // Created by ICL extraction DataInPort data_in_full[35] { Attribute tessent_design_gate_ports = \ "{data_in_full_35_"; Attribute tessent_design_rtl_ports = \ "{\data_in&full [2][1].b}"; } ... }`

Example 8

Examples of calling get_ports -use_path_matching_options:

Table 5-21. Calling get_ports -use_path_matching_options in RTL Mode

In RTL mode (RTL view) `SETUP> get_ports {{\data_in&full [2][1].b}} -use_path_matching_options {{\data_in&full [2][1].b}}`
In RTL mode (quick synthesis view) `SETUP> get_ports {{\data_in&full [35]}} -use_path_matching_options {{\data_in&full [35]}}`

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-22. Calling get_ports -use_path_matching_options in No RTL Mode After Synthesis

`SETUP> get_ports {{\data_in&full [35]}} -use_path_matching_options {data_in_full_35_}`
`SETUP> get_ports {{\data_in&full [2][1].b}} -use_path_matching_options (no results)`

Table 5-23. Calling get_ports -use_path_matching_options in No RTL Mode After Synthesis with Bit-Blasting

`SETUP> get_ports {{\data_in&full [2][1].b}} -use_path_matching_options {{\data_in&full[2][1].b}}`

Table 5-24. Calling get_ports -use_path_matching_options in No RTL Mode After Synthesis with Bit-Blasting (Special Characters Replaced by Underscores)

`SETUP> get_ports {{\data_in&full [2][1].b}} -use_path_matching_options {data_in_full_2_1__b}`

Related Topics

create_port

delete_ports

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_procfile_name

Context: all contexts

Mode: all modes

Returns the name of the procfile that was defined using the set_profile_name command.

Usage

get_procfile_name

Description

Returns the name of the procfile that was defined using the set_profile_name command.

Arguments

None

Return Values

The name of the procfile as it was specified to the set_procfile_name command.

Examples

In the following example, the path to the specified procfile name is stored in a Tcl variable.

```
set proc_file [get_procfile_name]
```

Related Topics

set_procfile_name

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_read_verilog_option

Context: unspecified, all contexts

Mode: all modes

Returns the options specified with the set_read_verilog_options command.

Usage

get_read_verilog_option -allow_enum_relaxation | -vcs_compatibility | -unbalanced_celldefine_severity | -strict_default_nettype_checks | -max_for_loop_size

Description

Returns the options specified with the set_read_verilog_options command.

Arguments

- -allow_enum_relaxation

  An option that returns a 1 if “set_read_verilog_options -allow_enum_relaxation” is set to “on”. Otherwise, the option returns 0.

- -vcs_compatibility

  An option that returns a 1 if “set_read_verilog_options -vcs_compatibility” are set to “on”. Otherwise, the option returns 0.

- -unbalanced_celldefine_severity

  An option that returns “error” if “set_read_verilog_options -unbalanced_celldefine_severity” is set to error. The option returns “warning” if the switch is set to warning.

- -strict_default_nettype_checks

  An option that returns a 1 if “set_read_verilog_options -strict_default_nettype_checks” is set to “on”. Otherwise, the option returns 0.

- -max_for_loop_size

  An option that returns the maximum number of for loop iterations defined by the set_read_verilog_options command. The default size is 5000.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_resource

Context: unspecified, all contexts

Mode: all modes

Returns true when all resources in the specified resource_list are available.

Usage

get_resource ***resource_list***

Description

Returns true when all resources in the specified resource_list are available.

You use this command inside your Tcl procs to check for the availability of resources and take appropriate action when they are missing. For example, you can choose to issue the create_flat_model command if your command depends on it and it is not yet available.

Arguments

- ***resource_list***

  A Tcl list of strings that contains any of the following resource names:

  design_is_elaborated

  Used to check that the hierarchical data model has been created using the set_current_design command.

  design_modules_exist

  Used to check that at least one Verilog or VHDL module exists in memory and can be accessed using the get_modules command.

  flat_model_exists

  Used to check that the flat data model is available. The flat data model was created by either the create_flat_model command or by the “set_system_mode analysis” command.

  icl_is_elaborated

  Used to check that the ICL hierarchical data model exists. It exists after the set_current_design command if an ICL module corresponding to the current design existed prior to issuing the command. It also exists in the analysis mode if the top-level ICL module was successfully extracted with ICL extract.

  icl_modules_exist

  Used to check that at least one ICL module exists in memory and can be accessed using the get_icl_modules command.

  rtl_mode_enabled

  Used to check that the -rtl option switch was used when specifying the context and that read-in Verilog and VHDL files can contain RTL constructs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example checks that both design and ICL modules exist.

```
if {![get_resource {design_modules_exist icl_modules_exist}]} {
        return -code error "you must first read in design and icl modules \
            before running this command"
}
```

Related Topics

create_flat_model

read_icl

read_verilog

read_vhdl

set_context

set_current_design

set_system_mode

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_run_synthesis_options

Context: unspecified

Mode: all modes

Returns the values for options that can be set for the run_synthesis command.

Usage

get_run_synthesis_options {[-default_synthesis_tool | -synthesis_output_directory]
| [***synthesis_tool*** [-compilation_options | -pre_compilation_commands
-post_compilation_commands | -startup_file | -command_name]]}
[-get_dictionary]

Description

Returns the default values for options that can be set for run_synthesis command. The options defined with set_run_synthesis_options are overridden if the are explicitly defined with the run_synthesis command.

Arguments

- -default_synthesis_tool | -synthesis_output_directory

  -default_synthesis_tool

  Returns the default synthesis tool previously set by the set_run_synthesis_options command. The default is ‘dc_shell’.

  -synthesis_output_directory

  Returns the synthesis output directory path. The default directory is *synthesis_outdir* and is located in the Tessent Shell invocation directory.

- ***synthesis_tool***

  A required value that specifies the synthesis tool when requesting a single tool-specific option. Possible values for ***synthesis_tool*** are:

  dc_shell

  genus

  The tool-specific options that require this switch are:

  - -compilation_options
  - -pre_compilation_commands
  - -post_compilation_commands
  - -startup_file
  - -command_name

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -compilation_options

  Returns the option list that adds additional options to the synthesis compile command used in the *<synthesis_tool>.synthesis_tcl*.

- -pre_compilation_commands

  Returns the command list that is added before the compile command in the *<synthesis_tool>.synthesis_tcl*.

- -post_compilation_commands

  Returns the command list that is added after the compile command in the *<synthesis_tool>.synthesis_tcl*.

- -startup_file

  Returns file path that enables you to explicitly specify the startup file that is used by synthesis.

- -command_name

  Defines the command name to invoke the synthesis tool. The default is dc_shell.

- -get_dictionary

  Returns all set synthesis options as a Tcl ‘dict’.

Examples

The following are examples of the get_run_synthesis_options using various options.

Example 1

This example returns the default pre_compilation_commands:

```
INSERTION> get_run_synthesis_options dc_shell -pre_compilation_commands \
           optimize_registers
```

Example 2

This example returns the default compilation options:

```
INSERTION> get_run_synthesis_options dc_shell -compilation_options \
           -map_effort high
```

Related Topics

check_synthesis

report_run_synthesis_options

run_synthesis

set_run_synthesis_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_scan_chain_families

Context: dft -scan

Mode: analysis

Gets scan chain families with certain characteristics.

Usage

get_scan_chain_families [*name_patterns*]
[-regexp]
[-nocase]

Description

Use this command to get scan chain families with certain characteristics.

Arguments

- *name_patterns*

  An optional list that enables returning scan chain families by their name or name pattern.

- -regexp

  An optional switch that enables returning scan chain families by regular expression matching.

- -nocase

  An optional switch that ignores character casing during matching.

Related Topics

create_scan_chain_family

delete_scan_chain_families

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_scan_chain_option

Context: pattern -scan

Mode: setup, when LBIST is on

Returns the setting of an option specified in the set_scan_chain_options command.

Usage

get_scan_chain_option {**-chain_name** ***chain_name***... | **-INStance** ***instance_name*** **-Block_chain_index_list** ***block_chain_index_list*** ...}
[-decompressor_to_scan_in_inversion | -scan_out_to_compactor_inversion]

Description

Returns the setting of an option specified in the set_scan_chain_options command.

Arguments

- **-chain_name*chain_name*...**

  Required switch and repeatable string that specifies the name of a scan chain in the current design.

- **-INStance** ***instance_name*** **-Block_chain_index_list** ***block_chain_index_list***...

  Required set of switches and values that specifies a core instance and the list of chain indices starting with “1” on the EDT controller. When this is used, the masking applied to the specified chain is OX. Use this option in conjunction with the -USed_chains switch.

- -decompressor_to_scan_in_inversion

  An optional Boolean switch that returns 1 when the option was explicitly set with the set_scan_chain_options command; otherwise a 0 is returned.

- -scan_out_to_compactor_inversion

  An optional Boolean switch that returns 1 when the option was explicitly set with the set_scan_chain_options command; otherwise a 0 is returned.

Examples

Suppose you specified the following command:

set_scan_chain_options -chain_name chain3 -decompressor_to_scan_in_inversion on -scan_out_to_compactor_inversion on

The get_scan_chain_option would return the following values:

get_scan_chain_option -chain_name chain3 -decompressor_to_scan_in_inversion

```
1
```

get_scan_chain_option -chain_name chain3 -scan_out_to_compactor_inversion

```
1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_scan_elements

Context: dft (with no sub-context), dft -scan, dft -test_points

Mode: setup, analysis, insertion

Returns a collection of scan elements that the tool has identified in the design.

Usage

get_scan_elements [*name_patterns*]
[-regexp]
[-nocase]
[-silent]
[-of_modules *obj_spec*]
[-of_instances *obj_spec* | -below_instances *obj_spec*]
[-of_child_scan_modes *child_mode_names*]
[-of_scan_modes *obj_spec*]
[-of_pins *of_pin_objects*]
[-of_chain_families *of_chain_families*]
[-of_gate_pins *of_gate_pins*]
[-of_test_points *test_point_spec*]
[-of_wrapped_ports *port_spec*]
[{-below_scan_elements *elements* [-flat] | -above_scan_elements *elements*} [-hierarchical]]
[-class {core | standard | wrapper}]
[-states {ignored | unusable | unusable_child_mode | unresolved_child_mode | usable | usable_indirectly | any_state}...]
[-types { chain | inferred | leaf_cell | segment | shift_register | transparent_leaf_cell | combinational_leaf_cell | any_type}...]
[-filter *attribute_equation*]

Description

Returns a collection of scan elements that the tool has identified in the design. Scan element can be a scan chain, a scan segment, a shift register, or leaf node. By default, the command only returns segment and leaf_cell elements. For more information, see Scan Element Object Type.

See the following sections in the *Tessent Scan and ATPG User's Manual* for usage examples:

- Scan Insertion Flow Steps
- Chain Port Sharing
- Scan Mode

Arguments

- *name_patterns*

  An optional list that enables returning scan elements by their name or name pattern.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -regexp

  An optional switch that enables returning scan elements by regular expression matching.

- -nocase

  An optional switch that ignores character casing during matching.

- -of_modules *obj_spec*

  An optional switch and value pair that gets the scan elements that are on and below all the instances of the given module or modules.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a possible mistake when specifying patterns. This -silent option has no effect on any other error reporting.

  Note

  You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the "set_tcl_shell_options -" switch description for detailed usage.

- -of_instances *obj_spec*

  An optional switch and value pair that gets the scan elements that are on the given instance or instances.

- -below_instances *obj_spec*

  An optional switch and value pair that gets the scan elements that are below the given instance or instances.

  If the specified *obj_spec* is an empty collection, all instances below the current design are considered. See the description of the of the "get_instances -below_instances" command description for the justification of this behavior.

- -below_scan_elements *elements*

  An optional switch and value pair that gets the scan elements that are inside the given hierarchical scan element or elements.

- -above_scan_elements *elements*

  An optional switch and value pair that gets the parent scan elements of the scan element or elements.

- -of_child_scan_modes *child_mode_names*

  An optional switch and value pair that gets the existing segments declared with the given mode or modes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -of_scan_modes *obj_spec*

  An optional switch and value pair that gets the scan elements that are included in the given mode or modes, previously added with the command add_scan_mode. This switch is only meaningful after issuing the analyze_scan_chains command.

- -of_pins *of_pin_objects*

  An optional switch and value pair that gets the scan_elements associated with the specified instance pins. Specified pins can be pins on scan element leaves or existing segment connection pins.

- -of_chain_families *of_chain_families*

  An optional switch and value pair that gets the scan elements that are included in the given chain families, previously added with the command create_scan_chain_family.

- -of_gate_pins *of_gate_pins*

  An optional switch and value pair that gets the scan elements that contain the specified flat model gate pin or pins.

- -of_test_points *test_point_spec*

  An optional switch and value pair that returns a collection of scan elements associated with each of the specified test points. These scan elements are similar to the scan elements the tool creates when it adds a dedicated wrapper cell. Each is a new scan element, not associated with an existing non-scan flop within the design.

  test_point_spec — A collection of test points. You can use the get_test_points command to get the collection using various options.

- -of_wrapped_ports *port_spec*

  An optional switch and value pair that gets a collection of wrapper scan_elements that isolate the supplied ports. You can use this switch for dedicated and shared wrapper cells.

  Note

  This switch is only available in dft -scan context in analysis mode.

- -flat

  An optional switch that recursively traverses the local hierarchies and returns only the leaf scan elements below the parent or parents.

- -hierarchical

  An optional switch that recursively traverses the local hierarchies and returns all encountered scan elements below the parent or parents.

- -class {core | standard | wrapper}

  An optional switch that returns only the scan elements of the specified class.

  The possible classes are:

  standard — Cell class of all elements when no wrapper analysis has been performed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

core — Cell class of all internal elements once the wrapper analysis has been performed.

wrapper — Cell class of all external elements once the wrapper analysis has been performed.

- -states {ignored | unusable | unusable_child_mode | unresolved_child_mode | usable | usable_indirectly | any_state}...

  An optional switch and literal pair that returns only the scan elements of the specified state. By default, only scan elements that have a state "usable" are returned. To introspect scan elements that have other states, use the -state option. See the example below for usage of "-state usable_indirectly" and "-state ignored".

  The possible states are:

  ignored — Scan elements ignored for scan insertion purposes.

  Tip

  This option is useful to report the list of non-scan cells.

  unusable — Scan elements that cannot be specified in a population for distribution purposes.

  unusable_child_mode — Scan elements that cannot be specified in a population for distribution purposes because the necessary "active_child_scan_mode" attribute is not set appropriately.

  unresolved_child_mode — Scan elements that cannot be specified in a population for distribution purposes because the "active_child_scan_mode" was not set and cannot be inferred.

  usable — Scan elements that can be specified in a population for distribution purposes.

  usable_indirectly — Usable scan elements that cannot be specified directly in a population for distribution purposes because they are part of a usable hierarchical container, which must be used instead.

  any_state — Scan elements of all states.

- -types { chain | inferred | leaf_cell | segment | shift_register | transparent_leaf_cell | combinational_leaf_cell | any_type}...

  An optional, repeatable switch that returns only the scan elements of the specified type or types.

  The possible types are:

  chain — Existing chains and new allocated chains that were created by distribution.

  inferred — Hierarchical scan elements allocated by the tool during distribution.

  leaf_cell — Scan elements from a cell library.

  segment — Existing segments or new virtual segment created by the user.

  shift_register — Shift-register scan arrangement identified by the tool.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

transparent_leaf_cell — Transparent cells (for example, lockup cell) traced inside existing chains or segments that have a scan length of zero.

combinational_leaf_cell — Combinational cells (for example, OR-gate) traced inside existing chains or segments.

any_type — Scan elements of all types.

- -filter *attribute_equation*

  An optional switch and value pair that returns only the scan elements that satisfy the provided attribute equation.

Examples

Example 1

This example illustrates how to get the traced scan elements inside a scan segment by using the "-state usable_indirectly" option. When you omit this option, the command does not return any of the child elements of the segment, because only their parent segment is "usable".

The example also shows how to identify scan elements that you have marked as non-scan using the "-state ignored" option.

```
> add_scan_segments subch1 -clock_pins clk1 -scan_enable_pin sen -on_module S \
   -so_connections so1 -si_connections si1> add_scan_segments subch2 -clock_pins clk2 -
scan_enable_pi sen -on_module S \
   -so_connections so2 -si_connections si2> add_nonscan_instances /uA1>
set_system_mode analysis
> report_scan_elements
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
==========================================================================================
name                  type       length  si_so_clocks  state           is_non_scannable_reason
-------------------  ---------  ------ ------------  --------------  -----------------
subch1/uA1/uS1/so1    segment    0      --           ignored          user_specified
subch1/uS1/so1        segment    7      +clk1,+clk1   usable           --
subch2/uA1/uS1/so2    segment    0      --           ignored          user_specified
subch2/uS1/so2        segment    2      +clk2,+clk2   usable           --
/psff7/Q              leaf_cell  1      +clk1        usable_indirectly  --
/psff6/Q              leaf_cell  1      +clk1        usable_indirectly  --
/psff5/Q              leaf_cell  1      +clk1        usable_indirectly  --
/psff4/Q              leaf_cell  1      +clk1        usable_indirectly  --
/psff3/Q              leaf_cell  1      +clk1        usable_indirectly  --
/psff2/Q              leaf_cell  1      +clk1        usable_indirectly  --
/psff1/Q              leaf_cell  1      +clk1        usable_indirectly  --
/psff9/Q              leaf_cell  1      +clk2        usable_indirectly  --
/psff8/Q              leaf_cell  1      +clk2        usable_indirectly  --
/ff1                  leaf_cell  1      -clk1n       usable           --
/ff2                  leaf_cell  1      +clk1        usable           --
/ff                   leaf_cell  1      -clk1        usable           --
/ff4                  leaf_cell  1      +clk1        usable           --
/ff5                  leaf_cell  1      +clk1        usable           --
/ff6                  leaf_cell  1      +clk2        usable           --
/ff7                  leaf_cell  1      -clk2n       usable           --
/ts2_mff1             leaf_cell  4      +clk1        usable           --
/ts4mff2              leaf_cell  4      +clk2        usable           --
/uA1/ff1              leaf_cell  1      +test_clk    ignored          user_specified
/uA1/ff2              leaf_cell  1      +test_clk    ignored          user_specified
/uA1/ff3              leaf_cell  1      +test_clk    ignored          user_specified
/uA1/ff4              leaf_cell  1      +test_clk    ignored          user_specified
/uA1/ff5              leaf_cell  1      +test_clk    ignored          user_specified
/uA1/uS1/sff1         leaf_cell  1      +test_clk    ignored          user_specified
/uA1/uS1/sff2         leaf_cell  1      +test_clk    ignored          user_specified
/uA1/uS1/sff3         leaf_cell  1      +test_clk    ignored          user_specified
/uA1/uS1/sff4         leaf_cell  1      +test_clk    ignored          user_specified
/uA1/uS1/sff5         leaf_cell  1      +test_clk    ignored          user_specified
/uA1/uS1/sff6         leaf_cell  1      +test_clk    ignored          user_specified
/uA1/uS1/sff7         leaf_cell  1      +test_clk    ignored          user_specified
/uA1/uS1/sff8         leaf_cell  1      +test_clk    ignored          user_specified
/uA1/uS1/\sff[9       leaf_cell  1      +test_clk    ignored          user_specified
/uS1/sff1/Q           leaf_cell  1      +clk1        usable_indirectly  --
/uS1/sff2/Q           leaf_cell  1      +clk1        usable_indirectly  --
/uS1/sff3/Q           leaf_cell  1      +clk1        usable_indirectly  --
/uS1/sff4/Q           leaf_cell  1      +clk1        usable_indirectly  --
/uS1/sff5/Q           leaf_cell  1      +clk1        usable_indirectly  --
/uS1/sff6/Q           leaf_cell  1      +clk1        usable_indirectly  --
/uS1/sff7/Q           leaf_cell  1      +clk1        usable_indirectly  --
/uS1/sff8/Q           leaf_cell  1      +clk2        usable_indirectly  --
/uS1/\sff[9] /Q       leaf_cell  1      +clk2        usable_indirectly  --
```

ANALYSIS> get_scan_elements -of_instances /uS1

```
{subch2/uS1/so2 subch1/uS1/so1}
```

ANALYSIS> get_scan_elements -below_scan_elements subch1/uS1/so1

```
//  Error: No scan elements.
```

ANALYSIS> get_scan_elements -below_scan_elements subch1/uS1/so1 \

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

-state usable_indirectly

```
{/uS1/sff1/Q /uS1/sff2/Q /uS1/sff3/Q /uS1/sff4/Q /uS1/sff5/Q /uS1/sff6/Q /uS1/sff7/Q}
```

ANALYSIS> get_scan_elements -of_instances /uA1/uS1 -state ignored

```
{subch2/uA1/uS1/so2 subch1/uA1/uS1/so1}
```

ANALYSIS> get_scan_elements -below_instances uA1 -state ignored

```
{/uA1/ff1 /uA1/ff2 /uA1/ff3 /uA1/ff4 /uA1/ff5 subch2/uA1/uS1/so2 subch1/uA1/uS1/so1 {/uA1/
uS1/\sff[9] } /uA1/uS1/sff1 /uA1/uS1/sff2 /uA1/uS1/sff3 /uA1/uS1/sff4 /uA1/uS1/sff5 /uA1/
uS1/sff6 /uA1/uS1/sff7 /uA1/uS1/sff8}
```

ANALYSIS> get_scan_elements -below_instances uA1 -state ignored -types leaf_cell

```
{/uA1/ff1 /uA1/ff2 /uA1/ff3 /uA1/ff4 /uA1/ff5 {/uA1/uS1/\sff[9] } /uA1/uS1/sff1
/uA1/uS1/sff2 /uA1/uS1/sff3 /uA1/uS1/sff4 /uA1/uS1/sff5 /uA1/
uS1/sff6 /uA1/uS1/sff7 /uA1/uS1/sff8}
```

ANALYSIS> get_scan_elements -below_instances uA1 -state ignored -types segment

```
{subch2/uA1/uS1/so2 subch1/uA1/uS1/so1}
```

Example 2

The following example shows two methods of using the -of_test_points switch:

get_scan_element -of_test_points {tp#u1/U39/Z tp#U75/Z tp#u1/U23/Z tp#U93/Z}

Note

You can create the list of test point names above using the report_test_points command.

get_scan_elements -of_test_points [get_test_points -below_instance u1]

Related Topics

add_scan_chains

add_scan_segments

delete_scan_elements

report_scan_elements

report_scan_segments

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_scan_modes

Contest: dft -scan

Mode: analysis, insertion

Returns the scan modes with certain characteristics.

Usage

get_scan_modes [*name_patterns*]

Description

Use this command to get scan modes of the current design with certain characteristics.

Arguments

- *name_patterns*

  An optional list that retrieves all scan modes by their name. If no name_patterns are specified, all scan modes are returned.

Related Topics

add_scan_mode

delete_scan_modes

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_scratch_directory

Context: unspecified, all contexts

Mode: all modes

Returns the absolute path to the *scratch* directory.

Usage

get_scratch_directory

Description

Returns the absolute path to the *scratch* directory.

During invocation, the tool creates the *scratch* directory in the temporary directory using the following syntax:

```
.tessent.tmp.username.hostname.process_id
```

When you exit, the tool deletes this *scratch* directory.

By default, it stores the scratch directory in the location identified by the TMPDIR environment variable. If you have not defined the TMPDIR environment variable, the tool creates the scratch directory in the */tmp* directory or in the current directory if the */tmp* directory is not available.

You can explicitly specify the location where the temporary *.tessent.tmp.username.hostname.process_id* directory is created by setting the TESSENT_TMP_LOCATION environment variable. The tool creates the temporary *.tessent.tmp.username.hostname.process_id* directory in the default location if the location specified by the TESSENT_TMP_LOCATION environment variable does not exist, is not a valid directory, or does not have write permission. The tool uses the TESSENT_TMP_LOCATION environment variable when both TMPDIR and TESSENT_TMP_LOCATION exist.

Arguments

None

Examples

Suppose TMPDIR points to */home/mydir*. The following example reports the location of the *scratch* directory:

get_scratch_directory

```
/home/mydir/.tessent.tmp.user1.myhostname.28140/scratch
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_silicon_insight_job_status

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Returns the status of diagnosis jobs that are running in the background.

Usage

get_silicon_insight_job_status [-job_handle *job_handle*] [-display {active | all | completed}] [-message_type {screen_only | info}]

Description

You can use this command when running diagnosis on an ATPG instrument and have specified off for the wait_for_diagnosis_result configuration debug option.

Arguments

- -job_handle *job_handle*

  An optional switch and string pair that specifies to return the “Running” or “Completed” status of the specified job. The specified job handle is the job ID that the tool uses when it submits a job.

- -display {active | all | completed}

  An optional switch and string pair that specifies the status type to display as a list. The choices are:

  all — displays all jobs.

  active — displays the actively running jobs.

  complete — displays the completed jobs.

- -message_type {screen_only | info}

  An optional switch and string pair that specifies to display the status only on-screen (the default) or both on-screen and in the transcript.

Examples

get_silicon_insight_job_status -display active

```
JOB: ‘all_cores_c1_2.3/job_script.sh’ completed.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_silicon_insight_option

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Returns current settings of options specified in the set_silicon_insight_option command.

Usage

get_silicon_insight_option [-cdp_verification {on | off}] [-interactive_ijtag] [-current_cdp] [-simdut_port] [-ignore_uncontacted_pin_data {off | on}] [-interactive_ijtag_pattern_type {svf | stil}] [-maximum_failing_cycles *number*]

Arguments

- -cdp_verification { on | off }

  An optional switch that specifies whether to verify that new test patterns added as part of a test were generated by Tessent tools, which means they contain a signature that links them to the design they were generated for. The default is “on.” You may want to turn CDP verification off if you are adding test patterns that were not generated by Tessent tools.

- -interactive_ijtag

  An optional switch that returns “on” or “off”.

- -current_cdp

  An optional switch that returns the current CDP.

- -simdut_port

  An optional switch that returns the name of the SimDUT port.

- -ignore_uncontacted_pin_data {off | on}

  An option switch and string pair that specifies whether to display or ignore uncontacted pin data.

- -interactive_ijtag_pattern_type {svf | stil}

  An optional switch and string pair that returns either the SVF or STIL pattern type.

- -maximum_failing_cycles *number*

  An optional string and integer that specifies to display the specified number of maximum failing cycles.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_simdut_data

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Returns information about specified design objects within designs simulated by SimDUT.

Usage

get_simdut_data **-value_type** { **value** | **type** | **regs** | **nets** | **modules** | **memories** } ***object_name***

Arguments

- **-value_type**

  A required switch and literal pair that specifies the information to return about the specified design object. The choices are:

  value — Value of the specified register, net, net array, or register array.

  type — Type of object, where the returned type value can be one of the following:

  Reg — Register

  Net — Net

  Module — Module

  Array Reg[*array_size*] — Register array

  Array Net[*array_size*] — Net array

  regs — List of registers for the specified module. You can only use this literal when you specify a module for the object name.

  nets — List of nets that the specified module contains. You can only use this literal when you specify a module for the object name.

  modules — List of modules that the specified module contains. You can only use this literal when you specify a module for the object name.

  memories — List of memory objects (2D+ registers) in the specified path.

- *object_name*

  A required literal that specifies a design object.

Description

Use this command to introspect nets, registers, and module instances within designs simulated by SimDUT. For more information, refer to "Simulating Desktop, ATE, and ATPG Behavior" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

Examples

The following example returns the registers associated with the SIMDUT_TB.DUT_inst.block4_I1 module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set result [get_simdut_data -value_type regs SIMDUT_TB.DUT_inst.block4_I1]
     puts $result
     sel_shiftreg[0:1] sel_nr enB enC enD enE enF selA selB art art2 art3[4:0][3] art4[20]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_simulation_context_list

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Returns the available simulation contexts in a Tcl list.

Usage

get_simulation_context_list [-predefined | -user_defined]

Description

Returns the available simulation contexts in a Tcl list.

The get_simulation_context_list command returns all simulation contexts or returns only predefined or user-defined. If you issue the command without arguments, the command returns all simulation contexts. Note that for predefined simulation contexts, this command returns only the simulation contexts for which simulation data currently exists. This requires that the tool first read a procedure file.

Arguments

- -predefined

  An optional switch that specifies predefined simulation contexts, which are listed in Table 8-6 on page 2434.

- -user_defined

  An optional switch that specifies user-defined simulation contexts.

Examples

The following example shows the Tcl list of values returned by the get_simulation_context_list command:

ANALYSIS> set_current_simulation_context stable_load_unload
ANALYSIS> add_simulation_context myContext1
ANALYSIS> add_simulation_context myContext2
ANALYSIS> get_simulation_context_list

```
{ stable_after_setup stable_load_unload stable_capture stable_shift
myContext1 myContext2 }
```

ANALYSIS > get_simulation_context_list -user_defined

```
{  myContext1 myContext2 }
```

ANALYSIS > get_simulation_context_list -predefined

```
{ stable_after_setup stable_load_unload stable_capture stable_shift }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

add_simulation_context

add_simulation_forces

copy_simulation_context

delete_simulation_contexts

delete_simulation_forces

get_current_simulation_context

report_simulation_contexts

report_simulation_forces

set_current_simulation_context

simulate_clock_pulses

simulate_forces

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_simulation_library_sources

Context: unspecified, all contexts

Mode: all modes

Returns the pathnames or file extensions previously specified with the set_simulation_library_sources command.

Usage

get_simulation_library_sources [-path_list | -extension_list | -sv_extension_list | -logical_library_map_list | -include_directory_list | -f_files]

Description

Returns the pathnames or file extensions previously specified with the set_simulation_library_sources command.

By default, the get_simulation_library_sources command returns a list of paired values that specify the pathnames to file and directory names. The first item in the pair is -v or -y to indicate whether the pathname is a file or a directory, respectively. This is useful when you want to add additional simulation library sources, because any invocation of the set_simulation_library_sources command overwrites previous invocation settings.

Arguments

- -path_list

  An optional switch that specifies to return the list of simulation library source paths with directory and filenames. Each name is associated with a -y and -v element to identify whether the path is a directory or filename. This switch is inferred when no other switch is specified.

  You can modify this returned list and feed it back to the set_simulation_library_sources command as shown below. The use of the {*} syntax is a Tcl 8.5 feature that converts the returned lists as string arguments.

  ```
  set_simulation_library_source {*}[get_design_source -path_list] \
                                {*}[get_design_source -ext_list]
  ```

- -extension_list

  A Boolean switch that specifies to return the list of extensions that was used to search the library directory for the specified format.

- -sv_extension_list

  A Boolean switch the specifies to return the list of SV extensions that was specified using the set_simulation_library_sources command.

- -logical_library_map_list

  A Boolean switch that specifies to return the list of library directory pairs that was specified using the set_simulation_library_sources command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -include_directory_list

  A Boolean switch that specifies to return the list of `include file directory pairs that were specified with the set_simulation_library_sources command.

- -f_files

  A Boolean switch that specifies to return a list of one or more ASCII files that contain pointers to design files that were specified using the set_simulation_library_sources command.

Examples

This example specifies information about simulation library sources, and then introspects the list of pathnames and the list of extensions specified:

SETUP> set_simulation_library_sources -v lib1.v lib2.v -y /u/joe/lib3 -extensions .v
SETUP> get_simulation_library_sources -path_list

```
-v lib1.v -v lib2.v -y /u/joe/lib3
```

SETUP> get_simulation_library_sources -extension_list

```
.v .v.gz
```

Related Topics

report_simulation_library_sources

set_simulation_library_sources

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_simulation_option

Context: all contexts

Mode: all modes

Returns the values for options that can be set for the set_simulation_options command.

Usage

```
get_simulation_option
    -set_reset_dominate_clock
    | -c6_mask_races
    | -multicycle_fault_simulation
    | -mux_select_x_sim_x
    | -feedback_buffer_x_init_value
    | -report_x_capturing_cells
    | -simulate_learned_sequential_tie_gates
    | -OBS_SCAN_PER_cycle_data_limit
    | -OBS_SCAN_LAST_shift_cycles
```

Description

Returns the values for options that can be set for the set_simulation_options command.

This command takes one argument at a time and returns a 0 or a 1.

Arguments

- -set_reset_dominate_clock

  A switch that instructs the tool to return the setting for the set_simulation_options command that controls whether the set or reset port dominates the clock assertion value for both flip-flops and latches when multiple ports are on simultaneously. Possible return values are 0 (off) or 1 (on).

- -c6_mask_races

  A switch that instructs the tool to return the set_simulation_options command setting of the pessimistic simulation on C6 violated state elements and mask the capture values only when the data port transitions at the same frame the clock port captures option. Possible return values are 0 (off) or a 1 (on).

- -multicycle_fault_simulation

  A switch that instructs the tool to return the set_simulation_options command setting of the multi-cycle path ATPG and fault simulation option. Possible return values are 0 (off) or a 1 (on).

- -mux_select_x_sim_x

  A switch that instructs the tool to return the set_simulation_options command setting of the model _mux primitives in the design as either consensus or non-consensus option. Possible return values are 0 (off) or a 1 (on).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -feedback_buffer_x_init_value

  A switch that instructs the tool to return the set_simulation_options command setting of the FB_BUF primitive to carry over its value from the previous cycle instead of being initialized to X option. Possible return values are 0 (off) or a 1 (on).

- -report_x_capturing_cells

  A switch that instructs the tool to return the set_simulation_options command setting of the option that reports scan cells that capture X during simulation. Possible return values are 0 (off) or a 1 (on).

- -simulate_learned_sequential_tie_gates

  A switch that instructs the tool to return the set_simulation_options command setting of the change the ATPG simulator behavior for sequential gates learned as tied during DRC option. Possible return values are 0 (off) or a 1 (on).

- -OBS_SCAN_PER_cycle_data_limit

  A switch that instructs the tool to return the set_simulation_options command setting of the default maximum number of patterns for which the tool stores per-cycle data (that the tool can monitor for debugging purposes).

- -OBS_SCAN_LAST_shift_cycles

  A switch that instructs the tool to return the set_simulation_options command setting of the number of shift cycles simulated in the parallel testbench.

Related Topics

set_simulation_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_simulation_value_list

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Returns a Tcl list of the simulation values on the specified gate_pin objects.

Usage

get_simulation_value_list ***obj_spec***

Description

Returns a Tcl list of the simulation values on the specified gate_pin objects.

The get_simulation_value_list command returns the simulation values on the specified gate_pin objects in the current simulation context. The simulation value of a single gate_pin is either X, 0, 1, or Z.

Arguments

- ***obj_spec***

  A required value that specifies one or more gate_pin objects or pins that map to gate_pin objects (on library cell boundaries or preserved design boundaries).

Examples

The following example shows that the get_simulation_value_list command is equivalent with querying the simulation_value attribute. However, note that get_attribute_value_list command requires a Tcl list of gate_pins in order to work correctly, as shown in the example:

```
set_current_simulation_context stable_after_setup
get_simulation_value_list {pi clk an_2/A0 an_2/Y}
```

```
1 X 1 X
```

```
get_attribute_value_list [get_gate_pins {pi clk an_2/A0 an_2/Y}] -name simulation_value
```

```
1 X 1 X
```

Related Topics

add_simulation_context

add_simulation_forces

copy_simulation_context

delete_simulation_contexts

delete_simulation_forces

get_simulation_context_list

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_simulation_contexts

report_simulation_forces

simulate_clock_pulses

simulate_forces

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_single_attribute_value

Context: unspecified, all contexts

Mode: all modes

Retrieves the value of an attribute on the specified design object.

Usage

get_single_attribute_value ***object_spec*** **-name** ***attribute_name*** [-is_specified]

Description

This command is the same as the get_attribute_value_list command except the ***object_spec*** must be a collection with exactly one element, otherwise the tool issues an error.

The return value is a single value.

Arguments

- ***object_spec***

  A required value that specifies a collection of exactly one element.

  It is also legal to supply a Tcl list of one object name, although this is not recommended for the following reasons:

  - Objects of different types can have the same name, which can lead to confusion.
  - Objects found may not support the requested attribute, which can generate an error.
  - For types that accept escaped identifiers, using such a name may not find the object at all. For example, using an escaped port name like {\nport } with "get_single_attribute_value [list {\nport }] ..." finds nothing. This is because escaping is not necessary for a port name that is valid without escaping. For that specific example, you should use "get_ports [list {\nport} -use_path_matching_options", and then pass the resulting collection object to the get_single_attribute_value command.

- **-name** ***attribute_name***

  A required switch and string pair that specify the name of the attribute whose value is to be returned.

  If the attribute ***attribute_name*** is not available for the *object_type* of ***object_spec***, that is, the attribute is not registered for this *object_type* and cannot be inherited, then the tool issues an error.

- -is_specified

  An optional Boolean switch that returns 1 when the attribute was explicitly set on the object otherwise a 0 is returned

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example:

```
get_single_attribute_value u1/DEF -name parent_instance
```

Is equivalent to the following get_attribute_value_list operation:

```
lindex [get_attribute_value_list u1/DEF -name parent_instance] 0
```

Example 2

The following example returns the value of *parent_instance* value for pins:

```
get_single_attribute_value u1/u5/u8/ABC -name parent_instance
```

```
u1/u5
```

Example 3

The following example returns pin names and their direction:

```
foreach_in_collection pjn  [get_pins u1/*] {
  puts "The pin [get_single_name $pin] has the direction [get_single_attribute_value \
        $pin -name direction]."
}
```

```
The pin u1/a has the direction input.
The pin u1/b has the direction input.
The pin u1/z has the direction output.
```

Example 4

When you use the get_single_attribute command to query an inout pin with the has_functional_source attribute, only the outer side of the instance is regarded:

```
get_single_attribute_value -name has_functional_source <pin>
```

Example 5

You can also use the get_single_attribute command to query the inner side of ports of modules (module side):

```
get_single_attribute_value -name has_functional_source [get_ports -of_pins <pin>]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_single_name

Context: unspecified, all contexts

Mode: all modes

Returns a string with the name of the element specified by the *object_spec* or the *name* arguments with simplified escaped identifiers, if possible.

Usage

get_single_name {***object_spec*** | **-from_object_name *name*** [-language {verilog | vhdl}]} [-remove_escaping] [-silent]

Description

This command returns the value of the name attribute for the object specified by ***object_spec*** or the name of the ***name*** with simplified escape identifiers, if possible.

Using this command with the "-from_object_names *name*" option verifies the existence of the name in the design and removes simple escaped identifiers if they are not reserved words in the design language. You can use this to identify name strings with reserved design language words.

The -language argument defines which design language escaping should be removed while processing the -from_object_names name list.

The -remove_escaping argument removes the backslash (\) and the trailing spaces in an escaped identifier. Many legacy commands such as add_clocks require the instance name to have the escape character stripped from its name. The argument also removes the escaping on reserved design language words. However, these legacy commands do accept collections as inputs in cases where the name format is not important. See the examples for an illustration of this behavior.

If ***object_spec*** or ***name*** entries are not valid, an error message displays unless the -silent switch is specified.

Arguments

- ***object_spec***

  A required string that specifies an object name or a single-element collection.

- **-from_object_name *name*** [-language {verilog | vhdl}]

  A required switch and string pair with an optional switch and literal pair.

  -language {verilog | vhdl}

  An optional switch and literal pair that specifies which language (Verilog or VHDL) should be considered during the removal of simple escaping. The default is Verilog when this option is not specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -remove_escaping

  An optional switch that specifies to strip out the backslash (\) and the trailing spaces in escaped identifiers. This option is useful to interact with legacy commands and the collection is not directly passed as input to them. See Example 3 for an illustration of this usage.

- -silent

  An optional switch that specifies to suppress error messages if the object in ***object_spec*** does not exist.

Examples

Example 1

The following example prints out all of the modules in the database:

```
foreach_in_collection elem [get_modules] {
   puts "Module: [get_single_name $elem]"
}

Module: modb
Module: modc
Module: modd
Module: top
Module: GND
...
```

Example 2

The following example generates an error because the collection of pins returned by the get_pins commands contains more than one element.

get_single_name [get_pins u1/i*]

```
//  Error: Collection contains more than one element.
```

Example 3

The following example uses the -remove_escaping argument to strip out the slash "\" from an escaped identifier.

get_single_name {\x[0] } -remove_escaping

```
x[0]
```

Example 4

The following example retrieves the a name of an instance with the -from_object_name option.

puts "from_object_name [get_single_name_list -from_object_name {deep_input1/\input /\output .\e }"
from_object_name {deep_input1/\input /\output .e}

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Hierarchy separators are retained, so the "." is not exchanged to "/". The "input" and "output" name parts are Verilog keywords, and the escaping is kept by this command, by default. To remove this escaping, also include the -remove_escaping switch.

Note

The braces used in the example to specify a name pattern containing escape identifiers are needed to prevent the Tcl shell from interpreting the \ character as its own escape sequence. See "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a complete description of pattern syntax and suggestions to use when the pattern is targeted for names containing escape characters.

Example 5

The following example retrieves the name of a port with the -from_object_name option.

```
puts "from_object_name [get_single_name -from_object_name {{ d/\a /\b /\c .\in }}]"
from_object_name d/a/b/c.in
```

The name is defined in braces due to the name escaping, as described in previous examples. Hierarchy separators are retained, so the "." is not exchanged to "/". The name does not contain language-specific keywords, so all escaping is removed.

Example 6

The following example retrieves the name of a hierarchical pin with the -from_object_name option.

```
set port_name {\d }
if {$port_name ne [get_single_name -from_object_name $port_name]} {
   puts "No design language reserved name in [get_single_name -from_object_name
{$port_name}]"
}
```

Due to the name escaping, the name is defined in braces. The name does not contain language-specific keywords, so the escaping is removed. Use this to check whether the port name contains reserved names in the design language.

Related Topics

get_attribute_list

get_attribute_option

get_attribute_value_list

get_common_parent_instance

get_icl_ports

get_modules

get_name_list

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_ssn_datapath_option

Returns settings specified using the set_ssn_datapath_options command.

Usage

get_ssn_datapath_option [-datapath *datapath_id*]
{**-alternate_input_id** | **-alternate_output_id** | **-input_bus_width** | **-output_bus_width**}

Description

The get_ssn_datapath_option command returns the requested datapath setting.

Arguments

- -datapath *datapath_id*

  An optional switch and value pair that specifies the name of the datapath to introspect the properties for. If you specify -alternate_input_id or -alternate_output_id, this argument is required. You can otherwise omit it if the datapath is unambiguous.

- **-alternate_input_id**

  A switch that returns the alternate input identifier of the datapath specified with the -datapath switch. You must use -datapath with this switch.

- **-alternate_output_id**

  A switch that returns the alternate output identifier of the datapath specified with the -datapath switch. You must use -datapath with this switch.

- **-input_bus_width**

  A switch that returns the input bus width of the datapath specified with the -datapath switch. You must use -datapath with this switch if the datapath is not unambiguous.

- **-output_bus_width**

  A switch that returns the output bus width of the datapath specified with the -datapath switch. You must use -datapath with this switch if the datapath is not unambiguous.

Examples

Example 1

The following sequence of commands returns the current value of the input bus width of the datapath “dp1”. Then it sets the width to a different value and introspects the new value.

get_ssn_datapath_option -datapath dp1 -input_bus_width

```
auto
```

set_ssn_datapath_options -datapath dp1 -input_bus_width 2
get_ssn_datapath_option -datapath dp1 -input_bus_width

```
2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following sequence of commands sets the alternate output ID for the datapath “dp1” to a different value and introspects the new value.

```
set_ssn_datapath_options -datapath dp1 -alternate_output_id out1
get_ssn_datapath_option -datapath dp1 -alternate_output_id
```

```
out1
```

Related Topics

set_ssn_datapath_options

add_icl_ssn_datapaths

set_icl_ssn_datapath_ports

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_ssn_option

Context: dft, patterns

Mode: setup, analysis

Returns the status of SSN in the current physical block or design specified with the set_ssn_options command.

Usage

get_ssn_option [-is_active]

Description

Returns the status of SSN in the current physical block or design set with the set_ssn_options command.

Arguments

- -is_active

  A switch that returns 1 if there are any active SSH instances in the current mode, and 0 otherwise. This switch is valid only in analysis mode.

Examples

This example demonstrates the use of the get_ssn_option and set_ssn_options commands:

ANALYSIS> get_ssn_option -is_active

```
1
```

ANALYSIS> set_ssn_options off

```
//  Error: SSN mode can only be changed in SETUP mode.
```

ANALYSIS> set_system_mode setup

SETUP> set_ssn_options off

SETUP> get_ssn_option

```
off
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_static_dft_signal_icall

Context: dft patterns

Mode: setup, analysis, insertion

Returns the iCall corresponding to the setting specified with the set_static_dft_signal_values command.

Usage

get_static_dft_signal_icall

Description

Returns the iCall corresponding to the setting specified with the set_static_dft_signal_values command.

When in dft -scan, dft -test_points, dft -edt and in patterns -scan, this command is automatically called when going from setup to analysis mode and the resulting value is inserted as if you had used it as the first "set_test_setup_icall [get_static_dft_signal_icall]" call. The command exists as a stand alone command for when you want to create a stand-alone pattern set in patterns context as shown in the example below.

This command is also called during process_patterns_specification to obtain the iCall corresponding to the DftControlSettings specified in the Patterns wrapper.

Arguments

None

Examples

The following example shows the use of the set_static_dft_signal_values followed by report_static_dft_signal_settings and the get_static_dft_signal_icall inside an open_pattern_set. See the description of the set_static_dft_signal_values command to understand where the "Set value" of the DFT signals showing a "Set source" value of "Inferred" came from.

As described above, this command is automatically invoked to configure test_setup when in a context that supports the set_test_setup_icall command.

```
set_context patterns -ijtag
read_icl ../sub_block_l3.icl
set_current_design sub_block_l3

set_system_mode analysis
open_pattern_set P1
report_static_dft_signal_settings
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// ICL Module      : sub_block_l3
// ---------------  -------------------  ---------  ----------
// DFT Signal Name  Usage                Set value  Set source
// ---------------  -------------------  ---------  ----------
// all_test         global_dft_control       -      -
// ltest_en         logic_test_control       -      -
// int_ltest_en     logic_test_control       -      -
// ext_ltest_en     logic_test_control       -      -
// int_edt          scan_mode(internal)      -      -
// ext_multi        scan_mode(external)      -      -
//
```

set_static_dft_signal_values int_edt 1
report_static_dft_signal_settings

```
// ICL Module      : sub_block_l3
// ---------------  -------------------  ---------  ----------
// DFT Signal Name  Usage                Set value  Set source
// ---------------  -------------------  ---------  ----------
// all_test         global_dft_control       1      Inferred
// ltest_en         logic_test_control       1      Inferred
// int_ltest_en     logic_test_control       1      Inferred
// ext_ltest_en     logic_test_control       -      -
// int_edt          scan_mode(internal)      1      Explicit
// ext_multi        scan_mode(external)      -      -
//
```

puts [get_static_dft_signal_icall]

```
dft_signal_iproc
  sub_block_l3_rtl2_tessent_tdr_sri_ctrl_inst.all_test 1
  sub_block_l3_rtl2_tessent_tdr_sri_ctrl_inst.int_edt 1
  sub_block_l3_rtl2_tessent_tdr_sri_ctrl_inst.int_ltest_en 1
  sub_block_l3_rtl2_tessent_tdr_sri_ctrl_inst.ltest_en 1
```

iCall {*}[get_static_dft_signal_icall]
close_pattern_set
write_patterns p1.v -verbose

Related Topics

add_dft_signals

reset_static_dft_signal_values

report_static_dft_signal_settings

set_static_dft_signal_values

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_synchronous_clock_groups

Context: dft, patterns

Mode: setup, analysis

Returns a Tcl list of collection handles. Each collection is a synchronous group of clock objects.

Usage

get_synchronous_clock_groups [-scope {capture | shift}]

Description

This command returns a Tcl list of collection handles. Each collection is a synchronous group of clock objects.

If no sync groups are defined, this command returns an empty list.

Arguments

- -scope {capture | shift}

  An optional argument and literal pair that specifies whether to return capture or shift clock groups. By default, capture clock groups are returned.

Examples

This example creates two synchronous capture clock groups and uses this command in Tcl code to return the clock groups.

```
SETUP> add_synchronous_clock_group {clk1 clk2}

SETUP> add_synchronous_clock_group {clk3 clk4}

foreach grp [get_synchronous_clock_groups] {
    puts [lsort [get_name_list $grp]]
}
```

```
clk1 clk2
clk3 clk4
```

Related Topics

add_synchronous_clock_group

delete_synchronous_clock_group

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_system_mode

Context: unspecified, all contexts

Mode: all modes

Returns the system mode.

Usage

get_system_mode [-previous | -is_in_transition]

Description

Returns the system mode.

Possible returned values are: setup, analysis, and insertion.

Arguments

- -previous

  An optional switch that specifies to return the previous system mode.

- -is_in_transition

  An optional switch that returns “1” when called within a callback that is running inside the system mode transition.

Examples

The following example returns the current system mode and the previous system mode. This enables you to know that you came to the insertion mode through the analysis mode.

get_system_mode

```
insertion
```

get_system_mode -previous

```
analysis
```

The following example shows how to use the -is_in_transition switch inside a callback to skip the operation when the feature is called within a system mode transition.

```
if {[get_system_mode -is_in_transition]}{
    return
}
```

Related Topics

set_system_mode

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_tcl_shell_option

Context: all contexts

Mode: setup

Introspects the Tcl shell options as specified with the set_tcl_shell_options command.

Usage

get_tcl_shell_option {-abort_dofile_on_error | -legacy_dofile_comments | -legacy_commands_with_spaces | -minimum_command_typing | -change_no_result_warnings_to_errors | -regex_style}

Description

Introspects the Tcl shell options as specified with the set_tcl_shell_options command. You can only specify one option at a time. The tool generates an error if you specify more than one option in a single command execution.

Arguments

- -abort_dofile_on_error

  An optional switch that returns the current setting of the abort_dofile_on_error option. Possible return values are off, on, and exit.

- -legacy_dofile_comments

  An optional switch that returns the current setting of the legacy_dofile_comments option. Possible return values are off and on.

- -legacy_commands_with_spaces

  An optional switch that returns the current setting of the legacy_commands_with_spaces option. Possible return values are off and on.

- -minimum_command_typing

  An optional switch that returns the current setting of the minimum_command_typing option. Possible return values are off and on.

- -change_no_result_warnings_to_errors

  An optional switch that returns the current setting of the change_no_result_warnings_to_errors option. Possible return values are off and on.

- -regex_style

  An optional switch that returns the current setting of the regex_style option. Possible return values are extended and perl.

Related Topics

report_tcl_shell_options

set_tcl_shell_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_test_end_icall_list

Context: dft, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Modes: setup, analysis

Returns the iCalls added to the test_end procedure by the set_test_end_icall command. Each iCall is a list containing the iProc and its arguments.

Usage

get_test_end_icall_list

Description

For each iCall added to the test_end procedure by the set_test_end_icall command, the get_test_end_icall_list command returns a list consisting of the three components that normally follow an iCall: the name of the iProc, preceded by the optional ICL instance path and separated by a dot, and one or more optional arguments to the iProc.

Arguments

None

Examples

The following example returns the iCalls added to the test_end procedure by the set_test_end_icall commands. Each iCall is a list consisting of the ICLinstance path, the name of the iProc, and the arguments to the iProc.

```
set_test_end_icall "m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on \
    edt_bypass off"

set_test_end_icall "m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on \
    edt_bypass off" -append

get_test_end_icall

{m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on edt_bypass off}
{m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on edt_bypass off}
```

Related Topics

iCall

set_test_end_icall

set_test_setup_icall

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_test_point_analysis_options

Context: dft -test_points

Mode: setup, analysis

Returns the current setting of an option specified with the set_test_point_analysis_options command. If you have not specified the option, the tool returns the default value of that option.

Usage

For VersaPoint test points when test point type is set to lbist_test_coverage:

```
get_test_point_analysis_options
    {-total_number |
     -control_points_number |
    -observe_points_number |
    -test_coverage_target |
    -pattern_count_target |
    -max_control_points_per_path |
    -exclude_cross_domain_paths |
    -observe_primary_outputs |
    -observe_blocked_faults |
    -exclude_clock_domains |
    -allow_in_xbounding_regions |
    -assume_clock_controllability |
    -shared_control_points_per_flop |
    -shared_observe_points_per_flop |
    -minimum_shift_length}
```

For VersaPoint test points when test point type is set to edt_pattern_count:

```
get_test_point_analysis_options
    {-total_number |
     -control_points_number |
    -observe_points_number |
    -max_control_points_per_path |
    -exclude_cross_domain_paths |
    -exclude_clock_domains |
    -observe_blocked_faults |
    -allow_in_xbounding_regions |
    -assume_clock_controllability |
    -shared_control_points_per_flop |
    -shared_observe_points_per_flop}
```

For VersaPoint test points when test point type is set to {edt_pattern_count lbist_test_coverage::

```
get_test_point_analysis_options
    {-total_number |
    -control_points_number |
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

-observe_points_number |
-test_coverage_target |
-pattern_count_target |
-max_control_points_per_path |
-exclude_cross_domain_paths |
-observe_primary_outputs |
-observe_blocked_faults |
-exclude_clock_domains |
-allow_in_xbounding_regions |
-assume_clock_controllability |
-shared_control_points_per_flop |
-shared_observe_points_per_flop |
-minimum_shift_length}

Description

Returns the current setting of an option specified with the set_test_point_analysis_options command. If you have not specified the option, the tool returns the default value of that option.

Arguments

- -total_number

  An optional switch that returns the maximum number of test points to be inserted in the design. The value can be an integer or a percentage of the total number of flops. In analysis mode, it always returns an integer value. However, in setup mode it returns percentage of total number of flops if you specified the total number as the percentage of the total number of flops.

- -control_points_number

  An optional switch that returns the maximum number of control points to be inserted in the design. The value can be an integer or a percentage of the total number of flops.

- -observe_points_number

  An optional switch that returns the maximum number of observe points to be inserted in the design. The value can be an integer or a percentage of the total number of flops.

- -test_coverage_target

  An optional switch that returns the target fault coverage to be achieved by the number of random patterns.

- -pattern_count_target

  An optional switch that returns the number of random patterns to be applied to the design.

- -max_control_points_per_path

  An optional switch that returns the maximum number of control points to be inserted on a single path.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -exclude_cross_domain_paths

  An optional switch that returns the current setting of the -exclude_cross_domain_paths option. Possible return values are ON or OFF.

- -observe_primary_outputs

  An optional switch that returns the current setting of the -observe_primary_outputs option. Possible return values are ON or OFF.

- -observe_blocked_faults

  An optional switch that returns the current setting of the -observe_blocked_faults option. Possible return values are ON or OFF.

- -exclude_clock_domains

  An optional switch that returns the list of clock domains excluded from test point insertion.

- -allow_in_xbounding_regions

  An optional switch that returns the current setting of the -allow_in_xbounding_regions option. Possible return values are ON or OFF.

- -assume_clock_controllability

  An optional switch that returns the current setting of the -assume_clock_controllability option. Possible return values are ideal or default.

- -shared_control_points_per_flop

  An optional switch that returns the maximum number of control points that can be driven by a single flop.

- -shared_observe_points_per_flop

  An optional switch that returns the maximum number of observe points that can be driven by a single flop.

- -minimum_shift_length

  An optional switch that returns the minimum observation scan shift length for fault simulation.

Examples

Example 1

The following example returns the value of the -total_number option.

ANALYSIS> get_test_point_analysis_options -total_number

```
1000
```

Example 2

The following example returns the value of the -allow_in_xbounding_regions option.

ANALYSIS> get_test_point_analysis_options -allow_in_xbounding_regions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
OFF
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_test_point_insertion_options

Context: dft -test_points

Mode: setup, analysis

Returns the current setting of an option specified with the set_test_point_insertion_options command. If you have not specified the option, the tool returns the default value of that option.

Usage

get_test_point_insertion_options
{-control_point_enable | -observe_point_enable | -test_point_clock
-prefer_scan_cells | -capture_per_cycle_en}

Description

Returns the current setting of an option specified with the set_test_point_insertion_options command. If you have not specified the option, the tool returns the default value of that option.

Arguments

- -control_point_enable

  An optional switch that returns the name of the enable signal for control points.

- -observe_point_enable

  An optional switch that returns the name of the enable signal for observe points.

- -test_point_clock

  An optional switch that returns the type of the clock to use to drive scan cells for test points. Possible return values are system_clocks, always_tclk or tclk_as_needed.

- -prefer_scan_cells

  An optional switch that returns the current setting of -prefer_scan_cells option. Possible return values are ON or OFF.

- -capture_per_cycle_en

  An optional switch that returns the name of the observation scan enable pin.

Examples

Example 1

The following example returns the value of the -test_point_clock option.

ANALYSIS> get_test_point_insertion_option -test_point_clock

```
always_tclk
```

Example 2

The following example returns the name of control point enable signal.

ANALYSIS> get_test_point_insertion_option -control_point_enable

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
cp_enable
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_test_point_sharing_restrictions

Context: dft -test_points

Mode: all

Report current test point sharing restrictions.

Usage

get_test_point_sharing_restriction
[{-instances | -modules | -cluster | -all}]

Description

Introspects the test point sharing restrictions as specified with the set_test_point_sharing_restrictions command.

Arguments

- -instances

  An optional switch that returns the list of all the instances for which the tool restricts test point sharing to only locations inside those instances. You cannot use this switch with any other switch for this command.

- -modules

  An optional switch that returns the list of all the modules for which the tool restricts test point sharing to only locations inside those modules. You cannot use this switch with any other switch for this command.

- -cluster

  An optional switch that returns the list of all the cluster_name attributes for which the tool restricts test point sharing. You cannot use this switch with any other switch for this command.

- -all

  An optional switch that returns all test point sharing restrictions. You cannot use this switch with any other switch for this command. This is the default.

Examples

The following examples all report on the following scenario:

```
set_test_point_sharing_restrictions on -instance {add_8 sbox_f_1 sbox_7 sbox_0 sbox_3}
-module {keysched1_9}
```

Example 1

The following example uses the get_test_point_sharing_restriction command to report all current test point sharing restrictions.

```
get_test_point_sharing_restriction -all
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Type      Name
//  --------  -----------
//  Instance  sbox_3
//  Instance  sbox_0
//  Instance  sbox_7
//  Instance  sbox_f_1
//  Instance  add_8
//  Module    keysched1_9
```

Example 2

The following example reports only the current test point sharing restrictions with respect to instances.

get_test_point_sharing_restriction -instance

```
//  Type      Name
//  --------  -----------
//  Instance  sbox_3
//  Instance  sbox_0
//  Instance  sbox_7
//  Instance  sbox_f_1
//  Instance  add_8
```

Example 3

The following example reports only the current test point sharing restrictions with respect to cluster_name attributes.

get_test_point_sharing_restriction –cluster

```
//  There is no restriction on cluster names.
```

Related Topics

set_test_point_sharing_restrictions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_test_point_type

Context: dft -test_points

Mode: setup, analysis

Reports the current test point type as specified by the set_test_point_types command.

Usage

get_test_point_type

Description

Reports the current test point type as specified by the set_test_point_types command.

The default test point type is edt_pattern_count.

Arguments

None.

Examples

The following example reports the default test point type, updates the test point type to lbist_test_coverage, and then reports the updated test point type for your design:

set_context dft -test_points -no_rtl
read_verilog *design_name*
read_cell_library *cell_library_name*
read_sdc *sdc_filename*
get_test_point_type

```
edt_pattern_count
```

set_test_point_type lbist_test_coverage
get_test_point_type

```
lbist_test_coverage
```

Related Topics

add_control_points

add_observe_points

set_test_point_types

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_test_points

Context: dft -test_points

Mode: all modes

Returns a collection of all test points, or a collection of test points matching an optional Tcl list of specified test point attributes.

Usage

get_test_points [*name_patterns*]
[-type { control | observe }] [-origin { design_analysis | user_specified }]
[-control_point_type { AND | OR }] [-control_point_target { data_path | clock_path }]
[-below_instances *below_instances*]
[-of_pins *of_pins*] [-filter *filter_expression*]
[-regexp] [-nocase] [-silent]

Description

This command returns a collection of all test points, or a collection of test points matching an optional Tcl list of test point name/location strings, with the option of filtering test points according to the following attributes:

- name or location (including regular expressions)
- type (control or observation)
- origin (user specified or tool generated by design analysis)
- control_point_type (AND or OR) (relevant only for control points)
- attribute values
- instance
- associated pins
- control point target (clock path or data path)

Note

Removing the “tp#” prefix from a test point name does not always produce the location of the test point. To convert a list of test point names to a list of locations, you must get the values of the location attribute for each test point. For example, you can use the following command:

```
get_attribute_value_list [get_test_points ...] -name location
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- name_patterns

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of test points according to their location (or name). The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no location_patterns are specified, the tool searches all test points. See section "Glob and Regular Expression Pattern Matching Syntax" for a complete description of the pattern syntax when the -regexp option is used or not used.

- -type { control | observe }

  An optional switch and literal pair that specifies to filter results based on the specified type of the test point. When omitted, all test points are considered. The option types have the following meaning:

  - control —Test points that are for the purpose of increasing controllability in design areas where, previously, the tool could not easily force certain state values. The returned collection includes only these test points.
  - observe —Test points that are for the purpose of increasing observability in design areas where, previously, the tool could not easily observe a certain state. The returned collection includes only these test points.

- -origin { design_analysis | user_specified }

  An optional switch and literal pair that specifies to filter results based on the specified origin of the test point. When omitted, all test points are considered. The origin options have the following meaning:

  - design_analysis — Test points that were generated by the tool. The returned collection includes only these test points.
  - user_specified— Test points that were manually specified. The returned collection includes only these test points.

- -control_point_type { AND | OR }

  An optional switch and literal pair that enable filtering results based on the type of gate used in the control type test point. Setting this flag filters out all test points that are not of the specified gate type (including all observe points).

- -control_point_target { data_path | clock_path }

  An optional switch and literal pair that specifies whether the control point is on the clock_path or the data_path

- -below_instances *below_instances*

  An optional switch and string pair that returns the test points that are below the specified instance.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -of_pins *of_pins*

  An optional switch and string pair that returns the test points that are on the specified pins.

- -filter *filter_expression*

  An optional switch and string pair that returns only the test points that satisfy the provided attribute equation. See "Test Point Data Model" for the list of Test point attributes that are valid attribute names.

- -regexp

  An optional switch that specifies to the tool to interpret the value of the name_patterns argument as a regular expression instead of as a simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" for a complete description of the regular expression syntax.

- -nocase

  An optional switch that specifies to the tool to perform case-insensitive pattern matching when looking for nets matching the name_patterns.

- -silent

  An optional switch that suppresses the error message normally generated when the command returns an empty collection. The error message is intended to indicate a possible mistake when specifying patterns. This -silent option has no effect on any other error reporting.

  Note

  You can use the -silent switch to manage the case in which an empty collection is returned. Before using the -silent switch for this purpose, refer to the "set_tcl_shell_options -" switch description for detailed usage.

Examples

Example 1

The following example returns all test points in the design:

```
get_test_points
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{tp#test_module2/neg1/Z tp#test_module2/neg2/Z
tp#1/test_module2/neg1/Z tp#1/test_module2/neg2/Z
tp#test_module1/neg1/Z tp#test_module1/neg2/Z
tp#1/test_module1/neg1/Z tp#1/test_module1/neg2/Z
tp#u1/U15/Z tp#u1/U14/Z tp#u1/U75/Z tp#U64/Z
tp#u1/U71/Z tp#U114/Z tp#u1/U45/Z tp#u1/U61/Z
tp#U111/Z tp#u1/U80/Z tp#u1/U42/Z tp#u1/U70/Z
tp#U119/Z tp#u1/U22/Z tp#U128/Z tp#u1/U43/Z
tp#u1/U39/Z tp#U75/Z tp#u1/U23/Z tp#U93/Z
tp#test_module2/nand2/Z tp#u1/U25/Z tp#U109/Z
tp#u1/U17/Z tp#U98/Z tp#U81/Z tp#U92/Z tp#U104/Z
tp#u1/my_module/nand2/Z tp#u1/U31/Z tp#u1/U32/Z
tp#U73/Z tp#U130/Z tp#U131/Z tp#u1/U67/Z
tp#u1/U49/Z tp#U126/Z tp#U65/Z tp#u1/U47/Z
tp#U100/Z tp#u1/U81/Z tp#u1/U53/Z ...}
```

Example 2

The following example returns all control points below the specified instances:

get_test_points -below_instances test_module* -type control

```
{tp#test_module2/neg1/Z
tp#test_module2/neg2/Z
tp#test_module1/neg1/Z
tp#test_module1/neg2/Z
tp#test_module1/neg2/A}
```

Example 3

The following example generates a report of the test points returned by the get_test_points command:

report_test_points [get_test_points -below_instances test_module* -type control]

```
==========================================================================================
name                    type     control_point_type  enable_connection_name  clock_domain
flop_location
----------------------  -------  ------------------  ----------------------  --------------
tp#test_module2/neg1/Z  control  AND                 /lbist_en               clk2
/test_module2/ts_default_PD_u12_cp_6sffp1_i

tp#test_module2/neg2/Z  control  OR                  /lbist_en               clk2
/test_module2/ts_default_PD_u12_cp_8sffp1_i

tp#test_module1/neg1/Z  control  AND                 /lbist_en               clk2
/test_module1/ts_default_PD_u11_cp_0sffp1_i

tp#test_module1/neg2/Z  control  OR                  /lbist_en               clk2
/test_module1/ts_default_PD_u11_cp_4sffp1_i

tp#test_module1/neg2/A  control  OR                  /lbist_en               clk1
/test_module1/ts_default_PD_u11_cp_2sffp1_i
```

Example 4

This example shows how to report only the control points inserted for improving the clock gaters controllability.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_test_points [get_test_points -control_point_target clock_path]

Related Topics

report_test_points

delete_test_points

insert_test_logic

report_test_logic

set_test_point_insertion_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_test_setup_icall_list

Context: dft, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Modes: setup, analysis

Returns the iCalls added to the test_setup procedure by the set_test_setup_icall command. Each iCall is a list containing the iProc and its arguments.

Usage

get_test_setup_icall_list

Description

For each iCall added to the test_setup procedure by the set_test_setup_icall command, the get_test_setup_icall_list command returns a list consisting of the three components that normally follow an iCall: the name of the iProc, preceded by the optional ICL instance path and separated by a dot, and one or more optional arguments to the iProc.

Arguments

None

Examples

The following example returns the iCalls added to the test_setup procedure by the set_test_setup_icall commands. Each iCall is a list consisting of the ICL instance path, the name of the iProc, and the arguments to the iProc.

set_test_setup_icall "m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on \ edt_bypass off"

set_test_setup_icall "m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on \ edt_bypass off" -append

get_test_setup_icall

```
{m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on edt_bypass off}
{m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on edt_bypass off}
```

Related Topics

get_test_end_icall_list

iCall

set_test_end_icall

set_test_setup_icall

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_testbench_simulation_options

context: all contexts

mode: all modes

Returns the value of a given option that may have been configured using the set_testbench_simulation_options command.

Usage

```
get_testbench_simulation_options
    {[-default_simulator |  -simulation_output_directory
     | -parallel_simulations ]
    | [-keep_simulation_data ]
    | [ -store_simulation_waveforms ]
    | [ -simulation_timeout ]
    | [ {questa | vcs} [ -simulator_options | -simulation_run_commands ]
        | -waveform_configuration_commands ] ]  }
    [ -get_dictionary ]
```

Description

Returns the value of a given option that may have been configured using the set_testbench_simulation_options command.

Arguments

- -default_simulator

  A switch that specifies to return the value of the run_testbench_simulations command option having the same name. The default value is configurable using the set_testbench_simulation_options command.

- -simulation_output_directory

  A switch that specifies to return the value of the run_testbench_simulations command option having the same name. This default value is also used by the check_testbench_simulations -simulation_output_directory_list option. The default value is configurable using the set_testbench_simulation_options command.

- -parallel_simulations

  A switch that specifies to return the value of the run_testbench_simulations command option having the same name. The default value is configurable using the set_testbench_simulation_options command.

- -keep_simulation_data

  A switch that specifies to return the value of the run_testbench_simulations command option having the same name. The default value is configurable using the set_testbench_simulation_options command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -store_simulation_waveforms

  A switch that specifies to return the value of the run_testbench_simulations command option having the same name. The default value is configurable using the set_testbench_simulation_options command.

- -simulation_timeout

  A switch that specifies to return the value of the run_testbench_simulations command option having the same name. The default value is configurable using the set_testbench_simulation_options command.

- questa | vcs

  A literal that specifies the simulator you want to use for simulation. The default value is "questa" from Siemens EDA.

- -simulator_options

  A switch that specifies to return the value of the run_testbench_simulations command option having the same name. The default value is configurable using the set_testbench_simulation_options command.

- -simulation_run_commands

  A switch that specifies to return the value of the run_testbench_simulations command option having the same name. The default value is configurable using the set_testbench_simulation_options command.

- -waveform_configuration_commands

  A switch that specifies to return the value of the run_testbench_simulations command option having the same name. The default value is configurable using the set_testbench_simulation_options command.

- -get_dictionary

  A switch that specifies to return the value of the run_testbench_simulations command option having the same name. The default value is configurable using the set_testbench_simulation_options command.

Examples

This example returns the default value used for the -simulation_output_directory option.

get_testbench_simulation_option -simulation_output_directory

```
./simulation_outdir
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_tester_option

Context: patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting, patterns -silicon_insight

Mode: All

Enables you to confirm the value of an individual tester-specific option that controls how the tool writes tester patterns when you use SSN.

Usage

get_tester_option {**-timeplate_constraints**
| **-use_timing_variables**
| **-timing_variables_group_name**}

Description

Returns the setting of an option specified with the set_tester_options command.

Note

These tester options apply only to designs with SSN.

Arguments

- {-timeplate_constraints | -use_timing_variables | -timing_variables_group_name}

  A required switch that specifies which tester option value to check. See the set_tester_options command for details of what may be returned.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_timeplate_list

Context: all contexts

Mode: all modes

Returns a list of the currently defined timeplates.

Usage

get_timeplate_list

Description

Returns a list of the currently defined timeplates. The data becomes available when you transition from setup mode to analysis mode. In analysis mode, you can change the data with subsequent read_procfile commands.

In addition to being available in analysis mode, the get_timeplate_list command is available in setup mode for cases when the transition to analysis mode fails after reading the procedure file, due to a DRC violation. Even though you are still in setup mode, the procedure file information is still in memory so that report and introspection commands can still be used on the information that was read.

Arguments

None

Return Values

A list of the currently defined timeplates.

Examples

get_timeplate_list

```
tp1 tp2 tp3
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_tool_info

Context: unspecified, all contexts

Mode: all modes

Returns information about the current tool and the current run as a Tcl result.

Usage

get_tool_info -name | -version | -cpu_time [ms | s | m | h | d] | -memory [b | kb | mb | gb] | -hostname | -interactive

Description

Returns information about the current tool and the current run as a Tcl result.

You must specify exactly one switch.

Arguments

- -name

  A switch that specifies to return the name of the current tool.

- -version

  A switch that specifies to return the version of the current tool.

- -cpu_time [ms | s | m | h | d]

  A switch that specifies to return the user time of the process. By default, this command returns the time in seconds, but you can optionally specify one of the other units. The possible units are:

  ms — Specifies the time unit is milliseconds.

  s — Specifies the time unit is seconds.

  m — Specifies the time unit is minutes.

  h — Specifies the time unit is hours.

  d — Specifies the time unit is days.

- -memory [b | kb | mb | gb]

  A switch that specifies to return the memory usage. By default, this command returns the usage in bytes, but you can optionally specify one of the other units.

  b — Specifies the memory usage unit is bytes.

  kb — Specifies the memory usage unit is kilobytes.

  mb — Specifies the memory usage unit is megabytes.

  gb — Specifies the memory usage unit is gigabytes.

- -hostname

  A switch that specifies to return the hostname.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -interactive

  A switch that specifies to return a “1” when the tool is in interactive mode or a “0” when running from a dofile in batch mode.

Examples

Example 1

The following examples return information about the current tool:

get_tool_info -name

```
tessent_shell
```

get_tool_info -version

```
2020.3
```

get_tool_info -cpu_time

```
2.773
```

get_tool_info -memory

```
228343808.0
```

get_tool_info -hostname

```
my_host
```

Example 2

The following shows how to format the returned double value for the -memory switch:

puts [format "Used memory %.2f GB." [get_tool_info -memory GB]]
//which prints the following:
...

```
Used memory 1.26 GB.
```

The -cpu_time switch also returns double values.

Related Topics

report_licenses

report_resources

set_tcl_shell_options

set_tool_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_tool_option

Context: all contexts

Mode: all modes

Returns certain Tessent Shell options from the set_tool_options command.

Usage

get_tool_option -reapply_settings_after_reelaboration | -allow_vhdl_2008

Description

Returns certain Tessent Shell options from the set_tool_options command.

Arguments

- -reapply_settings_after_reelaboration

  A switch that returns the re-elaboration settings for the loaded design.
- -allow_vhdl_2008

  A switch that returns the whether any blocks contain VHDL 2008 constructs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_trace_flat_model_option

Context: all contexts

Mode: setup, analysis

Introspects the default behavior of the trace_flat_model command.

Usage

get_trace_flat_model_option [-controllability] [-latch] [-tag_condition] [-stop_condition] [-stop_at_first_tag] [-max_levels] [-map_tag_to_design_module_boundary]

Description

Introspects the default behavior of the trace_flat_model command.

You can only specify one option at a time. The tool generates an error if more than one option is specified in a single command execution.

Arguments

- -controllability

  An optional switch that returns the current setting of the controllability option. Possible return values are:

  controlling— Gates are only traversed if the other inputs have simulated constant values making a single input directly controlling the output without an inversion.

  constant — Only pins with a constant value are traversed. This mode is useful to find the source of a constant value.

  unblocked — Gates are only traversed if the other inputs do not have simulated constant values making the input blocked from the output pin. Only unblocked combinational paths are traversed.

  connected—With this value, gates are always traversed when reached.

- -latch

  An optional switch that returns the current setting of the latch option. Possible return values are:

  normal — A latch is traced through only if the simulated value in the current simulation context makes the latch satisfy the specified controllability requirements. In case of "-controllability connected" all latch input pins - including the latch enable pin - are traced through.

  transparent — All latches are traced through the data input, set and reset pins regardless of the value at the latch enable pin. The latch enable pin is not traced through.

- -tag_condition

  An optional switch that returns the current setting of the tag condition option. The returned value is an attribute expression. For more details about filtering attribute equation formats, refer to "Attribute Filtering Equation Syntax" on page 3832.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -stop_condition

  An optional switch that returns the stop condition for the trace. The returned value is an attribute expression.

- -stop_at_first_tag

  An optional switch that returns the current setting of the -stop_at_first_tag option. This option specifies whether to interrupt tracing when the first tag object with the specified controllability is found. Possible return values are: {ON | OFF}

- max_levels

  An optional switch that returns the current setting of the max_levels option. This option specifies the maximum level of gates to trace. Possible return values are: unlimited or a number.

- -map_tag_to_design_module_boundary

  An optional switch that returns the current setting of the map_tag_to_design_module_boundary option, which specifies whether to map the tag points to gate_pins outside of library cells. Possible return values are: {ON | OFF}

Examples

Example 1

The following example returns the current setting for the -controllability option.

```
get_trace_flat_model_option -controllability
```

Example 2

The following example returns the current setting for the -tag_condition option.

```
get_trace_flat_model_option -tag_condition
```

Example 3

The following example returns the current setting for -stop_condition option.

```
get_trace_flat_model_option -stop_condition
```

Example 4

The following example returns the current setting for the -stop_at_first_tag option.

```
get_trace_flat_model_option -stop_at_first_tag
```

Example 5

The following example returns the current setting for the -max_levels option.

```
get_trace_flat_model_option -max_levels
```

Example 6

The following example returns the current setting for the -latch option.

```
get_trace_flat_model_option -latch
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

set_trace_flat_model_options

trace_flat_model

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_transcript_style

Context: unspecified, all contexts

Mode: all modes

Returns the keyword that specifies the style in which the tool transcribes commands.

Usage

get_transcript_style [-result_collection_limit | -argument_collection_limit | -one_collection_entry_per_line]

Description

Returns the keyword that specifies the style in which the tool transcribes commands.

The get_transcript_style command returns one of four keywords previously set with the set_transcript_style command. Possible return values are: off, full, tool_commands_only, and input_only.

Note that running the set_transcript_style command without an argument prints the style. Running get_transcript_style returns a value that you can assign to a variable for later use (see example below).

Arguments

- -result_collection_limit

  An optional switch that specifies to return the number of collection entries shown in the interactive mode.

- -argument_collection_limit

  An optional switch that specifies to return the number of collection entries shown in the listing when a collection is passed to a command as an argument.

- -one_collection_entry_per_line

  An optional switch that specifies to return the collections of arguments and results with one entry per line.

Examples

The following example stores the current transcript style in a variable, disables transcribing, then later restores the original transcript style:

```
set old_transcript_style [get_transcript_style]
set_transcript_style off

... # do something with transcribing disabled
set_transcript_style $old_transcript_style
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

echo

no_transcript

set_logfile_handling

set_transcript_style

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_tsdb_info

Context: dft, patterns

Mode: setup, analysis, insertion

An introspection command used to extract information stored in the *.tsdb_info* files found in the *dft_inserted_design* directories of the TSDB directories.

Usage

get_tsdb_info [*module_name*] [-design_identifier *id*] [-silent]
{-dictionary [-specified_module_only] | -exists | -level | -tsdb_info_file | -most_recent_design_identifier | -library_name | -ids_with_dates | -opened_tsdb_list | -child_blocks_dict | -version | -gate_extension }

Description

An introspection command used to extract information stored in the *.tsdb_info* files found in the dft_inserted_designs directories of the TSDB directory. Except for the -dictionary switch, all other switches apply to a specific *module_name* and *design_id* set. If the design is not elaborated in the session, you need to specify the *module_name* and potentially the *design_id* for which you want the requested information.

Arguments

- *module_name*

  A string value that specifies the name of a module, or a collection containing a single module that identifies the module for which the *tsdb_info* is requested. If unspecified, then the tool uses the *module_name* returned by the get_current_design command.

- -design_identifier *id*

  A switch and string pair that specifies the *design_id* associated with the design for which the *tsdb_info* is requested. You only need to specify this switch if the TSDB contains dft_inserted_designs directories associated with more then one *design_id*. The tool issues an error if there is more than one *design_id* available and this switch is not specified.

- -silent

  A switch that suppresses the error message generated when a *module_name* and *design_id* combination is specified, and there is no dft_inserted_designs directory found for the combination in any of the opened TSDB directories. In this case, the tool returns a null value.

- -dictionary

  A switch that returns the complete information set found in all *tsdb_info* files in the opened TSDBs. When this switch is specified, the tool ignores the specified *module_name* and

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

design_id values, and returns the information for all design and *design_id* pairs that exist. The format of the dictionary is as follows:

```
module_name {
 design_identifier {
   tsdb_info_file        <leaf name of .tsdb_info file>
   tsdb_info_version     integer
   creation_date         integer //epoch time
   level                 chip | physical_block | sub_block
   icl_extraction_needed On | Off
   full_path             <relative path of .tsdb_info
file from current directory>
   library_name          <logical library name
of module_name>
   gate_extension        <extension of concatenated netlist>
   opened_tsdb_list      <list of opened tsdb(s)when created>
   child_blocks_dict {
     module_name design_identifier
     ...
  }
}
```

- -specified_module_only

  An optional switch used in conjunction with the -dictionary switch that returns only the dictionary for the specified *design_name*/*design_id*.

- -exists

  A switch that returns a value that indicates whether a *design_name*/*design_id* set exists. Return values are as follows:

  1 —Indicates the *design_name*/*design_id* set exists.

  0 —Indicates the *design_name*/*design_id* set does not exist.

- -level

  A switch that returns the design level associated with the specified *design_name*/*design_id* set. The returned value reflects the set_design_level command's value used when the tool created the dft_inserted_design directory.

- -tsdb_info_file

  A switch that returns the full path name of the *.tsdb_info* file associated with the specified *design_name*/*design_id* set.

- -most_recent_design_identifier

  A switch that returns the most recent *design_id* for the specified *design_name*.

- -library_name

  A switch that returns the logical library name the specified *design_name* is associated to.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -ids_with_dates

  A switch that returns the list of *design_id*s associated with the specified *design_name*. The returned value is a list of list containing the following three elements: the *design_id* string; the date represented as an integer (epoch time); and the date represented in your local time zone.

- -opened_tsdb_list

  A switch that returns the list of TSDB directories that were open when the tool created the *.tsdb_info* file. The paths are reported relative to the current working directory in Tessent Shell, unlike those in the file itself, which are relative to the directory in which the tsdb info file resides. The first path in the list is always the relative path to the tsdb in which the tsdb info file resides.

- -child_blocks_dict

  A switch that returns a dictionary listing the child blocks found immediately below the specified *design_name*/*design_id* set. The keys are the child block names, and the values are the *design_id*s.

- -version

  A switch that returns the version of the *.tsdb_info* file. Return values are as follows:

  1 — Indicates the current version.

  0 — Indicates that there is no child block dictionary. If you specify the -child_block_dict switch, then the tool returns an empty dictionary.

- -gate_extension

  A switch that returns the extension used by the concatenated netlist for the specified *design_name*/*design_id* set.

Examples

Example 1

The following example illustrates the use of several switches of the get_tsdb_info command. Note how the format_dictionary command is used to format the returned dictionary when the -dictionary switch is used.

help get_tsdb_info

```
get_tsdb_info [ <module_spec> ] [ -design_identifier <id> ] [ -silent ]
 { -dictionary | -exists | -level | -tsdb_info_file |
   -most_recent_design_identifier |-library_name | -ids_with_dates |
   -opened_tsdb_list | -child_blocks_dict | -version | -gate_extension }
```

puts [format_dictionary [get_tsdb_info -dictionary]]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
chip {
   pass2 {
     tsdb_info_file chip.tsdb_info
     tsdb_info_version 1
     creation_date 1439219813
     level chip
     icl_extraction_needed On
     full_path simple_tsdb/dft_inserted_designs/chip_pass2.dft_inserted_design
     library_name work
     gate_extension vg
     opened_tsdb_list ./simple_tsdb
     child_blocks_dict {
       Pb1 pass2
       Sb1 pass2
     }
   }
   pass1 {
    tsdb_info_file chip.tsdb_info
    tsdb_info_version 1
    creation_date 1439219765
    level chip
    icl_extraction_needed On
    full_path simple_tsdb/dft_inserted_designs/
chip_pass1.dft_inserted_design
    library_name work
    gate_extension v_gate
    opened_tsdb_list ./simple_tsdb
    child_blocks_dict {
      Pb1 pass1
      Sb1 pass1
    }
  }
}
Pb1 {
   pass2 {
     tsdb_info_file Pb1.tsdb_info
     tsdb_info_version 1
     creation_date 1439219800
     level physical_block
     icl_extraction_needed On
     full_path simple_tsdb/dft_inserted_designs/
Pb1_pass2.dft_inserted_design
     library_name work
     gate_extension vg
     opened_tsdb_list ./simple_tsdb
     child_blocks_dict {}
   }
   pass1 {
     tsdb_info_file Pb1.tsdb_info
     tsdb_info_version 1
     creation_date 1439219759
     level physical_block
     icl_extraction_needed On
     full_path simple_tsdb/dft_inserted_designs/
Pb1_pass1.dft_inserted_design
     library_name work
     gate_extension v_gate
     opened_tsdb_list ./simple_tsdb
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
      child_blocks_dict {}
    }
  }
  Sb1 {
    pass2 {
      tsdb_info_file Sb1.tsdb_info
      tsdb_info_version 1
      creation_date 1439219783
      level sub_block
      icl_extraction_needed On
      full_path simple_tsdb/dft_inserted_designs/Pb1_pass2.dft_inserted_design
      library_name work
      gate_extension vg
      opened_tsdb_list ./simple_tsdb
      child_blocks_dict {}
    }
    pass1 {
      tsdb_info_file Sb1.tsdb_info
      tsdb_info_version 1
      creation_date 1439219754
      level sub_block
      icl_extraction_needed On
      full_path /simple_tsdb/dft_inserted_designs/Pb1_pass1.dft_inserted_design
      library_name work
      gate_extension v_gate
      opened_tsdb_list ./simple_tsdb
      child_blocks_dict {}
   }
  }
```

puts [get_tsdb_info Sb1 -most_recent_design_id]

```
pass2
```

puts [get_tsdb_info Sb1 -design_id pass2 -level]

```
sub_block
```

puts [get_tsdb_info Sb1 -design_id pass2 -library_name]

```
work
```

puts [get_tsdb_info Sb1 -design_id pass2 -open_tsdb_list]

```
./simple_tsdb
```

puts [get_tsdb_info Sb1 -design_id pass2 -gate_extension]

```
vg
```

puts [get_tsdb_info Sb1 -design_id pass2 -version]

```
1
```

puts [get_tsdb_info Sb1 -design_id pass2 -tsdb_info_file]

```
./simple_tsdb/dft_inserted_designs/Sb1_pass2.dft_inserted_design/
Sb1.tsdb_info
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

puts [get_tsdb_info Sb1 -design_id pass2 -exists]

```
1
```

Example 2

The following example shows how to use the result of the “get_tsdb_info -child_blocks_dict” command to define a variable.

SETUP> set_context dft -rtl
SETUP> open_tsdb [glob -type d -directory bottom_up tsdb*]
SETUP> read_design chip -view interface
SETUP> set_current_design chip

```
//  ---------------------------------------------------------------------
------
//  Begin ICL elaboration and checking.
//  -----------------------------------
//  ICL elaboration completed, CPU time=0.25 sec.
//  ---------------------------------------------------------------------
------
```

SETUP> set childBlocksDict [get_tsdb_info -child_blocks_dict]

```
Pb1 rtl Pb2 rtl Sb1 rtl
```

SETUP> format_dictionary -var child_blocks_dict $childBlocksDict

```
set child_blocks_dict {
  Pb1 rtl
  Pb2 rtl
  Sb1 rtl
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_tsdb_list

Context: all contexts

Mode: all modes

Returns a Tcl list of directory paths, relative to the current working directory, corresponding to all opened TSDB directories.

Usage

get_tsdb_list

Description

Returns a Tcl list of directory paths, relative to the current working directory, corresponding to all opened TSDB directories.

When the default or specified TSDB output directory is already created, it is returned in the list even if there was no explicit open_tsdb command for it.

Arguments

None

Examples

The following example shows the use of the get_tsdb_list command before and after some are closed.

get_tsdb_list

```
/home/projects/chipa/tsdbs/corea.tsdb /home/projects/chipa/tsdbs/coreb.t
sdb /home/projects/chipa/tsdbs/corec.tsdb
```

close_tsdb [list ${tsdb_home}/corea.tsdb ${tsdb_home}/coreb.tsdb]

get_tsdb_list

```
/home/projects/chipa/tsdbs/corec.tsdb
```

Related Topics

close_tsdb

set_tsdb_output_directory

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_tsdb_option

Context: all contexts

Mode: all modes

Retrieves the current setting of an option for referencing local and external Tessent Shell Database (TSDB) paths in design source dictionary files.

Usage

get_tsdb_option [-directory_path_map]

Description

This command returns the current value previously set with the set_tsdb_options command.

Arguments

- -directory_path_map

  An optional switch to return a Tcl list of strings representing the mapping that was set with the set_tsdb_options command.

Examples

The following examples shows how the get_tsdb_options command returns the list that was previously set with the set_tsdb_options command.

```
> set_tsdb_options -directory_path_map {/ny/designs /la/designs}
> get_tsdb_option -directory_path_map
/ny/designs /la/designs

> set_tsdb_options -directory_path_map {../../../designs /la/designs} \
                   -append
> get_tsdb_option -directory_path_map
/ny/designs /la/designs ../../../designs /la/designs
```

Related Topics

set_tsdb_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_tsdb_output_directory

Context: unspecified, dft, patterns

Mode: setup, analysis, insertion

Returns the directory used by the process_dft_specification command—the directory previously set by the set_tsdb_output_directory command.

Usage

get_tsdb_output_directory

Description

The get_tsdb_output_directory command returns the value that the process_dft_specification command uses. The TSDB output directory defaults to *./tsdb_outdir* until you run the set_tsdb_output_directory command. When you run either the get_tsdb_output_directory or set_tsdb_output_directory command, the tool verifies that the specified path is a writable directory if it already exists, or that the parent directory is a writable directory if it does not.

The process_dft_specification command is currently the only command that makes use of the TSDB directory.

Arguments

None

Examples

Example 1

The following example stores the TSDB output directory into a Tcl variable called tsdb_outdir.

set tsdb_outdir [get_tsdb_output_directory]

Example 2

The following example shows a possible error that can be generated when invoking the command.

set_tsdb_output_directory dir1
exec “rm -r dir1; touch dir1”
get_tsdb_output_directory

```
//  Error: The specified tsdb_output_directory 'dir1' exists but is not a
//  directory.
//  Error: Directory was either deleted or its write permission changed
//  between now and the invocation of set_tsdb_output_directory command.
```

Related Topics

set_tsdb_output_directory

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_validated_objects

Context: dft, patterns

Mode: setup, analysis, insertion

Validates the values supplied to arguments or switches of a command created with the register_tcl_command command.

Usage

get_validated_objects ***object_spec*** [-allowed_object_type_list *allowed_object_type_list*] | [-argument_name *argument_name* | -switch_name *switch_name*]

Description

This command accepts a list of object names or a collection of objects, and that verifies if their types are present in the permitted object type list. When this condition is met, the tool returns a collection with all object names converted to objects.

If you are validating the value of a switch, specify this value using the -switch_name switch. For example, if you are validating the value provided to a switch called -below_instance, specify "-switch_name below_instance". The error messages start with "Supplied -below_instance value contains..."

If you are validating a positional argument, use the -argument_name switch to specify the argument's name. For example, if you are validating the <port_spec> argument, then specify -argument_name port_spec. The error messages start with "Supplied <*port_spec*> contains..." If you do not specify either the -argument_name switch or the -switch_name switches, the error messages start with "Supplied <*node_spec*> contains...".

Arguments

- ***object_spec***

  A Tcl list with one or more object names or a collection containing one or more objects.

- -allowed_object_type_list *allowed_object_type_list*

  A switch-value pair that specifies which object types are permitted. The value is a Tcl list of one or more object names. The permitted object names are as follows:

  - port
  - pseudo_port
  - pin
  - net
  - module
  - instance

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

  - gate_pin
  - icl_port
  - icl_pin
  - icl_module
  - icl_instance

  When unspecified, the default permitted object type list is {port pin net pseudo_port gate_pin}.

- -argument_name *argument_name*

  A switch value pair that specifies the name of the argument to be validated. Specifying this option affects the error messages that are generated when the object_spec contains invalid objects. When both the -argument_name and the -switch_name switches are not specified, the error messages start with “Supplied <*node_spec*> contains...” When the -argument_name switch specifies a value xxx, then the error messages start with “Supplied <*xxx*> contains...”

  For example, if you are validating the <*port_spec*> argument, then specify -argument_name port_spec, and the error messages start with “Supplied <*port_spec*> contains...”

- -switch_name *switch_name*

  A switch value pair that specifies the name of a switch value to be validated. Specifying this option affects the error messages that are generated when the object_spec contains invalid objects. When both the -argument_name and the -switch_name switches are not specified, then the error messages start with “Supplied <*xxx*> contains...” When the -switch_name switch specifies a value xxx, then the error messages start with “Supplied -xxx value contains...”

  For example, if you are validating the value provided to a switch called -below_instance, specify -switch_name below_instance, and the error messages start with “Supplied -below_instance value contains...”

Examples

The following example uses the get_validated_objects command with the body of a registered Tcl command to validate the argument of a command having this syntax:

get_instance_leaf_name *instance_spec*

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The proc implementing this command uses the get_validated_objects as follow:

```
proc get_instance_leaf_name.body {args} {
  array set ARGS {instance_spec ""}
  array set ARGS $args

  set instance_objects [get_validated_objects $ARGS(instance_spec) \
                        -argument_name instance_spec \
                        -allowed_object_type_list instance]

  #If I make it hear, instance_objects contains a collections of instances
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_write_patterns_options

Context: patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: analysis, setup

Returns introspection results from either port lists or vector callbacks.

Usage

Common Usage

get_write_patterns_options **-existing_used_ports** | **-additional_port_list** | **-all_used_port_list** | **-port_direction** ***port_direction*** | **-port_offstate** ***port_offstate*** | **-continuous_expect_during_payload** | **-delete_selective_tck_stretching_timeplates_after_write**

Usage for Vector Callbacks

get_write_patterns_options **-vector_callback_list** [-trigger {before_first | after_last | every_cycle}] | **-received_callback_port_list** ***proc_name*** | **-returned_callback_port_list** ***proc_name***

Description

Returns introspection results from either port lists or vector callbacks.

Arguments

- **-existing_used_ports**

  A required switch that returns the current subset of existing design ports as a Tcl list of port names or gate_ids. The list is in the order that the ports are listed in the final pattern file, minus the additional ports. If there is no subset of existing_used_ports active, the full list of default design ports is returned.

- **-additional_port_list**

  A required switch that returns a Tcl list of lists of the additional port names, and port directions. This is the order in which the additional ports are listed in the final pattern file that is written. If there is no additional ports specified, an empty list is returned. The format of the list of lists is as follows:

  ```
  { { port_name direction} { port_name direction} …}
  ```

- **-all_used_port_list**

  A required switch that returns the current subset of existing design ports and any specified additional ports as a Tcl list of lists of the port names and port directions. If no subset of existing design ports is active, then the default list of design ports plus any additional ports are returned. The format of the Tcl list is the same as that used for -additional_port_list.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-port_direction *port_direction***

  A required switch and string pair that returns the port direction for a named port, where the port direction is one of input, output, or inout. This command works on all ports, including any additional ports added with -additional_port.

- **-port_offstate *port_offstate***

  A required switch and value pair that returns the binary offstate state value of '0' or '1' for a named port if the port is a clock port. If the port is not a clock port, then the state value of 'N' is returned.

- **-continuous_expect_during_payload**

  A required switch that returns a Tcl list of lists containing the continuous expect port names and expect values. If there are no continuous expect ports, this returns an empty list. The format of the returned list is as follows:

  {{*<port_name>* *<value>*} {*<port_name>* *<value>*} …}

- **-delete_selective_tck_stretching_timeplates_after_write**

  A required switch that returns the current setting of the -delete_selective_tck_stretching_timeplates_after_write switch, which you specify with the set_write_patterns_options command. The switch returns a value of 1 for the default setting of on, and returns a value of 0 for the off setting.

- **-vector_callback_list** [-trigger {before_first | after_last | every_cycle}]

  A required switch and literal pair that returns the current vector callback proc names as a Tcl list. If there are no callbacks, an empty list is returned. If the -trigger option is used, only the name of the callback for that trigger is returned.

- **-received_callback_port_list *proc_name***

  A required switch and value pair that returns the callback ports as a Tcl list of names in the same order in which the port_values are present in the vector_set. This introspection command can be used within vector_callback procs to have the list of ports and the order that are present in all port_values strings within vectors. The *proc_name* value must be the name of a Tcl proc that was already registered as a callback procedure using the "set_write_patterns_options -vector_callback" command.

- **-returned_callback_port_list *proc_name***

  A required switch and value pair that returns the callback ports as a Tcl list of names in the same order in which the port_values are present in the vector_set. This introspection command can be used within vector_callback procs to have the list of ports and the order that are present in all port_values strings within vectors. The *proc_name* value must be the name of a Tcl proc that was already registered as a callback procedure using the "set_write_patterns_options -vector_callback" command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

This example show the results of this command if called after the port_list has been set.

ANALYSIS> puts [join [get_name_list [get_write_pattern_options -all_used_port_list]] “\n”]

```
ijtag_si input
ijtag_so output
ijtag_tms input
ijtag_tck input
myNewInput input
myNewOutput[1] output
myNewOutput[0] output
```

Related Topics

report_write_patterns_options

set_write_patterns_options

write_patterns

Vector Creation and Modification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_xclock_handling

Context: all contexts

Mode: all modes

Returns the options specified with the set_xclock_handling command.

Usage

get_xclock_handling [-pessimistic_simulation]

Description

Returns the options specified with the set_xclock_handling command, which specifies whether the sequential element model outputs X when any of its clock inputs become X.

Arguments

- -pessimistic_simulation

  An optional switch that returns the setting (“1” (on) or “0” (off)) that affects the output of the edge-triggered sequential element model when the clock port becomes X as shown in Table 8-24 on page 2969.

Related Topics

set_xclock_handling

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_xy_net

Context: patterns -scan_diagnosis, patterns -silicon_insight

Mode: setup, analysis

Returns a Tcl list of nets located at the specified location in the LDB.

Usage

get_xy_instance ***x y*** [-ldb_units]

Description

This command returns a Tcl list of net names that overlap the specified x and y coordinates, as stored in the LDB. You must provide chip-level coordinates, which the tool automatically converts to core-level coordinates if a chip-mapped core-level LDB is currently open. When using a chip-mapped core-level LDB, you must set the layout core instance before invoking this command.

Note

This command only returns results for verbose LDBs (non-compact).

By default, the command assumes the units of the specified coordinates are expressed in microns. You can override the default by specifying the -ldb_units switch.

Arguments

- ***x y***

  Required x and y chip-level coordinates within the LDB. The default is microns.

- -ldb_units

  An optional switch that enables you to specify the coordinates in LDB units rather than microns which is the default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_xy_instance

Context: patterns -scan_diagnosis, patterns -silicon_insight

Mode: setup, analysis

Returns a Tcl list of cell instances located at the specified location in the LDB.

Usage

get_xy_instance ***x y*** [-ldb_units]

Description

This command returns a Tcl list of instance names that overlap the specified x and y coordinates, as stored in the LDB. You must provide chip-level coordinates, which the tool automatically converts to core-level coordinates if a chip-mapped core-level LDB is currently open. When using a chip-mapped core-level LDB, you must set the layout core instance before invoking this command.

By default, the command assumes the specified coordinates are expressed in microns units. You can override the default by specifying the -ldb_units switch.

Arguments

- ***x y***

  Required x and y chip-level coordinates within the LDB. The default is microns.

- -ldb_units

  An optional switch that enables you to specify the coordinates in LDB units rather than the default micron units.

Related Topics

get_instance_polygons

report_instance_polygons

report_net_polygons

report_xy_instance

set_layout_core_instance

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

help

Context: unspecified, all contexts

Mode: all modes

Lists commands and associated usage syntax.

Usage

help [*command_name*] [-manual] [-all]

Description

Lists commands and associated usage syntax.

Specifying the help command without any arguments returns a list of all available commands. Specifying a single command name displays its usage syntax.

Arguments

- *command_name*

  An optional string that specifies the name of the command for which you want help. Entering the first part of a command name returns a list of commands that start with the specified string. The *command_name* can include wildcard characters. For example, you can use an "*" to match zero or more characters at the position of the asterisk in the string.

  You also can use the tab completion method on available commands, which supplies missing characters between underscores. For example, specifying "help se_sy_m" and pressing the Tab key inserts "help set_system_mode" on the command line. Tab completion also works in this same way when invoking any tool command.

  If a single command is returned, its usage syntax also displays.

- -manual

  An optional switch that displays the reference manual description for the specified command. If you use this switch without specifying a command name, the tool opens the Tessent documentation system, giving you access to all the manuals for the product group.

- -all

  An optional switch that displays all command matches, including unavailable commands, which are identified with the "(...unavailable...)" notation. Specifying "help *unavailable_command_name*" displays the reason why a particular command is unavailable.

Examples

Example 1

The following example returns the usage syntax for the history command:

```
help history
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
history [ <count> ] [ -Nonumbers ] [ -Reverse ] [ -Save <savefile> ]
```

Example 2

The following example returns all commands in the current mode that begin with the string “set_design”:

help set_design

```
set_design_include_directories
set_design_macros
set_design_sources
```

Example 3

The following example returns all commands in the current mode that contain the string “design”:

help *design*

```
delete_design
get_current_design
set_current_design
set_design_include_directories
set_design_macros
set_design_sources
write_design
```

Example 4

The following example returns all commands containing the string “design”, including those that are unavailable:

help *design* -all

```
delete_design
find_design_names      (...unavailable...)
get_current_design
set_current_design
set_design_include_directories
set_design_macros
set_design_sources
write_design
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

history

Context: unspecified, all contexts

Mode: all modes

Displays a list of previously run commands.

Usage

history [*list_count*] [-Nonumbers] [-Reverse] [-Save *filename*]

Description

The history command lists commands that you have previously run.

You can use command line editing to recall a previous command and edit it for submission as a new command.

A leading number precedes each command line in the history list that indicates the order in which the commands were entered.

Arguments

- *list_count*

  An optional integer that specifies for the tool to display only the specified number (*list_count*) of most recent run commands. If no *list_count* is specified, the tool displays all previously run commands.

- -Nonumbers

  An optional string that specifies for the tool to display the history list without the leading numbers. This is useful for creating dofiles. The default displays the leading numbers.

- -Reverse

  An optional switch that specifies for the tool to display the history list starting with the most recent command rather than the oldest.

- -Save *filename*

  An optional switch and string pair that specifies the name of a file to which to write the command history. This is useful for creating dofiles.

Examples

Example 1

The following command saves the command history into a dofile for later execution.

```
history -save dofile1.txt
```

Example 2

The following command displays the history list with leading numbers, starting with the oldest command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

history

```
 1  help hist
 2  dof instructor/fault.do
 3  set_system_mode analysis
 4  set_fault_type stuck
 5  add_faults -all
 6  create_patterns
 7  report_statistics
 8  report_faults -class ATPG_UNTESTABLE
 9  analyze_fault /I$20/en -stuck_at 1
10  set_system_mode setup
11  set_system_mode analysis
12  set_fault_type iddq
13  add_faults -all
14  create_patterns
15  report_statistics
16  history
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iApply

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Triggers the retargeting of all queued iRead and iWrite commands.

Usage

iApply [-end_in_pause]

Description

Triggers the retargeting of all queued iRead and iWrite commands.

This command causes the retargeting engine to calculate the optimal sequence of scan loads, primary input stimulations, and primary output observations to satisfy all queued iRead targets and iWrite targets, as illustrated in Figure 5-6.

The number of ParallelIO time frames and Scan time frames used to satisfy all queued up iWrite and iRead commands depends on the number of queued commands and the circuit topology described in the ICL description.

The solution criteria for the iWrite commands is that all specified iWrite values are achieved and present at the end of the iApply time frame. The order in which they become satisfied is unknown and is chosen by the solver in order to minimize the number of clock cycles needed to complete the tasks. The solver is designed to minimize transition on the writable objects and limits them to a single transition per iApply time frame if the ICL circuit permits it.

If it is not possible to achieve all specified iWrite values at the end of the time frame, the specified iWrite command set is said to have no solution. The iRead object only needs to be observed once anywhere within the time frame. The value is strobed at the first opportunity that it becomes observable and is then ignored if re-observed on subsequent Scan or parallel I/O time frames.

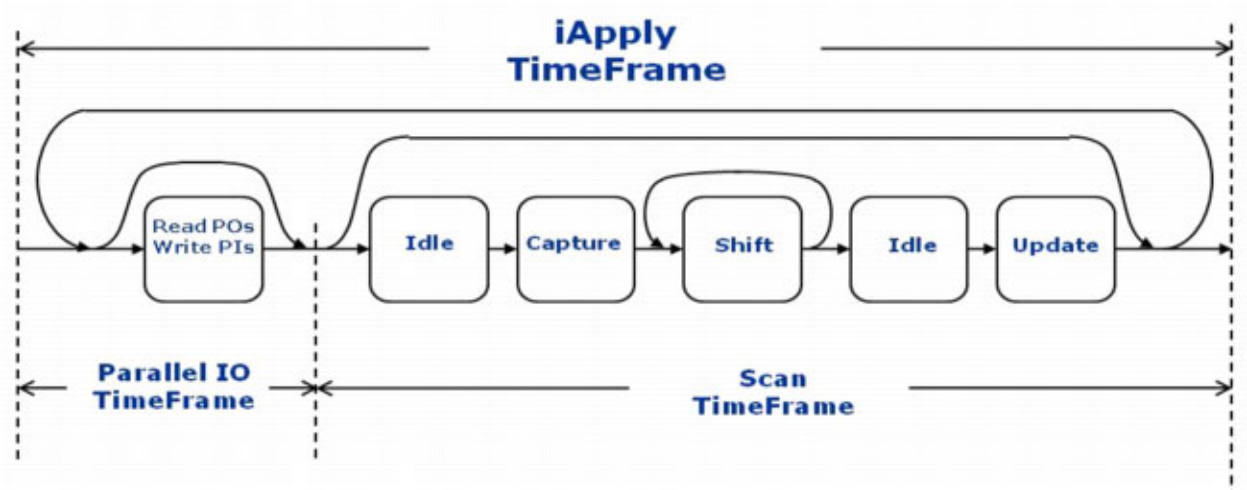
If there is no solution for the retargeting task, the iRead targets and iWrite targets are taken out of the queue and the current state of the circuit is retained. An error message appears in this case. If there is a solution, the solution is appended to the opened pattern set. The circuit state is updated to reflect the situation after the actions within the current iApply time frame have been applied.

You can use the optional -end_in_pause switch to help create IJTAG scan chain integrity tests. This switch enables you to observe the same values at the scan output that have been shifted in

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

before. When you do not use this switch, the shift register content is always overwritten by the capture values before being observed at the scan output.

Figure 5-6. iApply Time Frame



Arguments

- -end_in_pause

  An optional switch that helps you to create IJTAG scan chain integrity tests. The -end_in_pause switch changes the handling of the last scan load of the iApply and the first scan load of the next iApply.

  Note that the following explanation refers to states of a finite state machine (FSM) of an IEEE 1149.1 (JTAG) TAP controller. “IR” is instruction register, and “DR” is data register. For a complete explanation about how the TAP controller operates, refer to the IEEE 1149.1 standard document.

  When you use the -end_in_pause switch, and the last scan load is a “TAP scan load” (capture, shift and update activity is controlled by a TAP state machine), then the TAP state machine ends in the pause-DR or pause-IR state (that is, the tool performs no update-DR/update-IR). The first scan load of the next iApply starts from there and performs another scan load (obviously without capture-DR/capture-IR) before returning to RTI (run test/idle).

  When you use the -end_in_pause switch, and the last scan load is an ordinary scan load (capture, shift, and update activity is controlled by the appropriate CaptureEnPort, ShiftEnPort, UpdateEnPort), then the “update” of the last scan load of the iApply is skipped. The first scan load of the next iApply skips the “capture.” For more information about the CaptureEnPort, ShiftEnPort, UpdateEnPort keywords, refer to “ICL and PDL Modeling” in the *Tessent IJTAG User's Manual*.

  You can use this switch only when the iWrite targets are all part of ScanRegisters. Those ScanRegisters must be configurable to appear as one concatenated scan chain.

Return Values

None

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

In the following example, two iWrite and two iRead commands are queued up when the iApply command is issued. The solver starts from the current state of the ICL circuit and looks for a series of ParallelIO and Scan time frames to have u1.ina equal to “11” and u1.inb equal to “0” at the end of the iApply time frame, and to have observed at least once the u1.out1 and u1.out2 objects.

The number of ParallelIO and Scan time frames needed to achieve the objective depends on the ICL network. If both u1.ina and u1.inb shared a common source in ICL, then the given PDL could not be satisfied because it would be impossible to have u1.in1 high with u1.inb low together at the end of the iApply time frame.

```
iWrite u1.ina 0b11
iWrite u1.inb 0b0
iRead u1.out1
iRead u1.out2 0x7
iApply
```

Example 2

The following example applies a chain integrity test.

```
open_pattern_set chain_pattern
        iWrite myRegister 0b00110011
        iApply -end_in_pause
        iRead myRegister 0b00110011
        iApply
close_pattern_set
```

Related Topics

iRead

iWrite

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iCall

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Calls an iProc registered against the ICL module associated with the specified *effective_icl_instance_path*.

Usage

iCall [icl_instance_path.]**iproc_name** [*arguments…*]

Description

Calls an iProc registered against the ICL module associated with the specified *effective_icl_instance_path*.

The *effective_icl_instance_path* is computed by concatenating the following three components:

- iProc instance path — Path that exists when running commands specified within an iProc; it is set to the effective_icl_instance_path used in the iCall command that called the iProc. This value is empty when the iCall is not specified within an iProc.
- iPrefix path — Path specified as the argument to the iPrefix command since the beginning of the open_pattern_set for commands that are not inside an iProc, or since the beginning of the iProc for commands that are inside an iProc. If no iPrefix command has yet been specified in the current scope, this value is empty.
- icl_instance_path — An optional path specified in front of the iproc_name inside the iCall command, or in front of the icl_object inside other PDL commands such as iWrite, iRead, and iClock.

The effective_icl_instance_path is illustrated in the examples that follow.

The other arguments of this command are passed to the iProc in the same order as they appear in the command line. You must specify at least as many arguments as there are variables without default values in the corresponding iProc specification. For arguments defined with a default value in the associated iProc, their value takes on the default value when omitted.

Error Reporting

Similar to ordinary Tcl procs, iProcs can be nested, and this nesting can even be a mixture of iProcs and ordinary Tcl procs. An iProc can be called from within an ordinary Tcl proc that in turn has been called from within another iProc. If an error occurs in such a situation, it is often desirable to know the exact call stack to find the root-cause of the error. Tessent Shell reuses the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tcl infrastructure to provide this sort of information. The built-in variable “errorInfo” contains useful data about the call stack and the location where the erroneous command can be found.

In case of an error, the Tcl interpreter of Tessent Shell fills the variable appropriately, but it treats iProcs and ordinary Tcl procs in the same way. Therefore, the content of the “errorInfo” variable shows the internal representation of the failing iProc. This internal representation consists of one or more namespace identifiers, the name of the module for which the iProc is registered and the name of the iProc. These parts are separated by double colons.

Example

```
ANALYSIS> puts $errorInfo
can't read "iDoNotExist": no such variable
   while executing
"puts $iDoNotExist"
   (procedure "::mentor::iprocs::module1::run_test" line 21)
   invoked from within
"::mentor::iprocs::module1::run_test {blue} "
   invoked from within
"iCall block2_I1.inst1.run_test blue"
```

The content of the variable “errorInfo” shows a procedure called “::mentor::iprocs::module1::run_test”. This refers to the iProc “run_test”, which is registered for the module “module1” in the iProc namespace “::mentor::iprocs”. The lines above also show the command by which this iProc has been called ("iCall block2_I1.inst1.run_test blue") and the root cause of the error (can't read "iDoNotExist": no such variable).

Arguments

- [icl_instance_path.]**iproc_name**

  Name of an iProc preceded by an optional icl_instance_path and separated by a dot. The specified iProc must have been registered against the ICL module associated with the *effective_icl_instance_path*.

- arguments...

  Additional optional arguments that are passed to the iProc procedure. The number of required arguments depends on the definition of the iProc and on the existence of default values for the iProc arguments.

Return Values

None. Unlike normal Tcl proc, even if the iProc contains the Tcl return command, the specified value is not returned by the iCall command, which is consistent with the IEEE 1687 standard.

Examples

The following example assumes an ICL module called myInstrument1 exists and this module is instantiated four times below the current design with the following instance paths:

core1_i1.myInst1_i1core1_i1.myInst1_i2core1_i2.myInst1_i1core1_i2.myInst1_i2

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The ICL module myInstrument1 has an ICL sub-module called myInstrument1_decode_logic that is instantiated within myInstrument1 as “decode_logic”.

This numbered example illustrates the declarations of iProcs against the myInstrument1 and myInstrument_decode_logic modules, and the use of the iCall command to run them against various instances of myInstrument1.

Line 1 issues the first iProcsForModule command for myInstrument1, which means iProcs declared afterward are registered against that module. For that reason, the iProcs declared at line 2 and 12 are registered against myInstrument1.

Line 24 issues the second iProcsForModule command, which means the next iProcs are registered against the module myInstrument1_decode_logic.

The commands starting at line 41 are not inside an iProc so the iProc instance path value is empty when computing the effective_icl_instance_path for those commands.

The iCall on line 42 has an effective_icl_instance_path of core1_i1.myInst1_i1 because iPrefix path is currently set to core1_i1 (from the iPrefix command on line 41) and icl_instance_path is specified as myInst1_i1.

The iCall on line 42 specifies the value for the cnt argument while the second argument is left to the default value specified within the iProc declaration.

The iCall on line 43 overwrites the value of the second argument. Note that the first argument is required because it has no default values defined in the iProc declaration.

The iCall on line 5 is inside of an iProc, so it has an iProc instance path value that is equal to the effective_icl_instance_path value of the iCall that called the iProc. Looking at the iCall command on line 5 when called by the iCall on line 41, its effective_icl_instance_path is core1_i1.myInst1_i1.decode_logic because iProc instance path is core1_i1.myInst1_i1, iPrefix path is decode_logic, and icl_instance_path is empty.

The ICL instance core1_i1.myInst1_i1.decode_logic is an instance of ICL module myInstrument1_decode_logic and the proc Setup is registered against this module on line 25 because it follows the iProcsForModule command specified on line 24.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
1  iProcsForModule myInstrument1
2  iProc run_testa {cnt {mode red}} {
3     iCall CheckStart
4     iPrefix decode_logic
5     iCall Setup $cnt $mode
6     iCall run $mode
7     iPrefix
8     iCall decode_logic.Status
10    iRead decode_logic.cnt $cnt
11 }
12 iProc CheckStart {} {
13    iRead go Fail
14    iRead done No
15    iApply
16 }
17 iProc run {mode} {
18    set cycles(red) 50
19    set cycles(blue) 90
20    set cycles(green) 70
21
22    iRunLoop $cycles($mode) -tck
23 }
24 iProcsForModule myInstrument1_decode_logic
25 iProc Setup { cnt mode } {
26    iWrite mode $mode
27    if {$mode == "red"} {
28       iWrite p1 0b0
29       iWrite p1(2) 0b1
30    }
31    iApply
32    iWrite enable Yes
33    iApply
34 }
35 iProc Status {} {
36    iRead done Yes
37    iRead go Pass
38    iRead count $cnt
39    iApply
40 }
41
41 iPrefix core1_i1
42 iCall myInst1_i1.run_testa 5
43 iCall myInst1_i2.run_testa 10 blue
44 iPrefix
45 iCall core1_i2.myInst1_i1.run_testa 15 red
```

Related Topics

ALL_MAX_LOOP

iPrefix

iProc

iProcsForModule

iWrite

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iClock

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Checks whether a controlling clock path from a valid clock source to the specified clock port exists.

Usage

iClock [*icl_instance_path*.]***icl_clock_port***

Description

Checks whether a controlling clock path from a valid clock source to the specified clock port exists.

This command takes into account the iClockOverride specifications and the FreqMultiplier and FreqDivider properties in the ICL description.

Prior to issuing this command, clock path sensitization must have been performed using the appropriate iWrite and iApply commands such that SelectedBy sources of ICL ClockMultiplexers seen along the traced path have known constant values.

If clock tracing fails before reaching a primary input, an error is generated. If the primary input is reached but does not have an associated add_clocks command to specify the clock is active with a known period, an error is also generated.

The cumulative effective frequency multiplier is computed along the traced clock path and the final number is stored against the specified clock port. This value is later used to normalize the iRunLoop command to a common tester period as described in the iRunLoop command description.

Arguments

- [*icl_instance_path*.]***icl_clock_port***

  Name of an internal or top-level clock port defined inside the ICL description. The specified clock port must be a port declared as a ClockPort or ToClockPort inside the ICL and it must be found at the computed effective_icl_instance_path. The iCall command description describes how the effective_icl_instance_path value is computed and the factors that affect it.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Return Values

None

Examples

In the following example, an iClock command is first specified to check that the reference source pin of the PLL has a controlling source to a primary input of the current design, and that the port has a specified clock period as defined by an add_clocks command.

The iWrite and iApply commands then program the PLL to a multiplication factor of 1.5 and set the clock multiplexer to the state you require.

The result of the PLL setup is reflected by the iClockOverride command such that the iClock definition on inst1.mbist_i1.clk, which traces to the PLL, properly computes the correct multiplication factor.

The first iRunLoop command specifies for the PLL to lock in less that 200 reference clock cycles. The second iRunLoop results in a loop of 4000 (6000*2/3) clock cycles at the current design boundary. For more information on the iRunLoop scaling algorithm, see the iRunLoop command description.

```
iClock inst1.pll.ref
iWrite inst1.pll_setup.mult 0b0100
iWrite inst1.pll_setup.source 0b01
iApply
iClockOverride inst1.pll.vco -freqMultiplier 3 -freqDivider 2
iRunLoop 200 -sck inst1.pll.ref
iClock inst1.mbist_i1.clk
iRunLoop 6000 -sck inst1.mbist_i1.clk
```

Related Topics

add_clocks

delete_clocks

get_iclock_list

get_iclock_option

iClockOverride

report_clocks

report_icl_modules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iClockOverride

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Models how the functional clocking has been programmed.

Usage

iClockOverride [*icl_instance_path.*]***icl_clock_port*** -source [*icl_instance_path.*]*icl_clock_port* [-freqMultiplier *multiplier*] [-freqDivider *divider*]

iClockOverride [*icl_instance_path.*]***icl_clock_port*** -period *period* [s | ms | us | ns | ps | fs | as]

Description

Models how the functional clocking has been programmed.

System clocking syntax inside ICL is adequate to describe a constant frequency multiplication and division number. With ClockMultiplexers in ICL, it is possible to model alternative sources as long as the select lines of the multiplexers are controlled by ijtag registers. The functional clocking path of ijtag instruments is not required to be controlled by ijtag itself. As long as the current functional clocking is properly modeled, the iRunLoop commands can be normalized correctly.

You should have a way to set up the functional clock using the ijtag network if you need to change it between patterns. You can use the complete syntax available inside the test_setup procedure to program an arbitrary functional clock system. You simply use this iClockOverride command to model how the functional clocking was programmed. Even if the multiplication factor is controlled by ijtag registers, you use the iClockOverride command to describe the effect of the newly written value. This is illustrated in the iClock command example.

You can use the iClockOverride command to override only the source, or only the cumulative multiplication/divider value, or both. If the -source option is not specified, the source is unchanged; it remains the source documented in ICL, or the last iClockOverride -source value that may have been specified within the current open pattern set.

If the -freqDivider and the-freqMultiplier arguments are both specified, both are overridden. Specifying just -freqDivider without -freqMultiplier or vice versa, sets the unspecified option to 1. If neither of the -freqDivider and the-freqMultiplier arguments are specified, the previous -freqDivider and the-freqMultiplier values are preserved, regardless of whether they came from ICL or a previously specified iClockOverride.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- **[icl_instance_path.]icl_clock_port**

  A required value that specifies an internal clock port defined inside the ICL description. The specified clock port must be a port declared as a ToClockPort inside the ICL and it must be found at the computed effective_icl_instance_path. The iCall command description describes how the effective_icl_instance_path value is computed and the factors that affect it.

- -source [icl_instance_path.]icl_clock_source_port

  An optional switch and value pair that specifies an internal or top-level clock port defined inside the ICL description. The specified clock port must be a port declared as a ClockPort or ToClockPort inside the ICL and it must be found at the computed effective_icl_instance_path. The iCall command description describes how the effective_icl_instance_path value is computed and the factors that affect it.

- -freqMultiplier *multiplier*

  An optional switch and value pair that overrides the frequency multiplication factor between the*[icl_instance_path.]**icl_clock_port*** and its source. It's value defaults to 1 when -freqDivider is specified otherwise it defaults to the previously defined value, either from the definition in ICL or from a previous iClockOverride command on the given *[icl_instance_path.]**icl_clock_port***

- -freqDivider *divider*

  An optional switch and value pair that overrides the frequency ratio between the *[icl_instance_path.]**icl_clock_port*** and its source. It's value defaults to 1 when -freqMultiplier is specified otherwise it defaults to the previously defined value, either from the definition in ICL or from a previous iClockOverride command on the given *[icl_instance_path.]**icl_clock_port***

- -period *period*

  An optional switch and value pair with optional time unit that overrides the period of the ToClockPort. The default time unit is "ns". A ToClockPort with a specified period serves as an internal clock source, that is, as an oscillator.

Return Values

None

Examples

The following example assumes both inst1.inst2.sysclock and clk2 are declared as ToClockPorts in the ICL description. The first iClockOverride command only overrides the source and leaves the multiplication and division number unchanged. The second iClockOverride command overrides the frequency multiplication factor to 10 and the frequency division ratio to 1.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
iClockOverride inst1.inst2.sysclock -source clk2
iClockOverride inst1.inst2.sysclock -freqMultiplier 10
```

Related Topics

add_clocks

delete_clocks

get_iclock_list

get_iclock_option

iClock

report_clocks

report_icl_modules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iComparePort

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iTopProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Enables you to compare a value on an arbitrary primary output port or bidirectional port inside an ijtag pattern set.

Usage

iComparePort ***object_spec*** *value* [-after *state_transition_list*]

Description

Enables you to compare a value on an arbitrary primary output port or bidirectional port inside an ijtag pattern set.

An iComparePort or an iForcePort command is only permitted in an iTopProc and not in a normal iProc as it is not retargetable upward.

It is best to keep the iProcs that only interact with the ijtag ports separated from the ones using the iComparePort command so that you maintain the ability to retarget the bulk of your PDL.

The effect of the iComparePort is applied on the next iApply command or the next iRunLoop command and lasts until it is iCompared to an X. You can use the -after switch to delay the compare and have it appear only after a specific state transition.

An iComparePort command remains active until one of the following events happen:

- The iComparePort is overwritten with another iComparePort command or — in case of bidirectional ports — with another iForcePort command.
- The pattern set is closed and the next pattern set is opened without the -no_initial_ireset switch. (Note: this only applies to the context "patterns -ijtag").
- If an iComparePort command is part of an iTopProc called from within an iMerge block, then the lifetime of the iComparePort command is limited to the retargeting of that iTopProc.

  This is done for the following reason: If there are conflicting iComparePort specifications in concurrent iCalls, the associated iTopProcs must be retargeted in series. If the iComparePort specifications survived the end of the iTopProc, the active iComparePort after the iMerge -end would depend on the (undefined) order of the retargeting of the concurrent iCalls. This non-deterministic behavior must be avoided.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can use this command in an iCall used as part of test_setup for ATPG, however, it makes the iTopProc un-retargetable upward. You should avoid doing so at least until you reach the chip level.

Arguments

- ***object_spec***

  A collection containing the top level output or bidirectional port/ports. If you are using this command on a port that is not described in the top ICL module (a non-IJTAG port), you must have read the interface view of the top level Verilog or VHDL module to get the list of ports available to compare. Use "read_verilog -interface_only" or "read_vhdl -interface_only" to read in the port list prior to issuing the set_current_design command as shown here:

  ```
  set_context patterns -ijtag -no_rtl
  read_verilog top.v -interface_only
  read_icl top.icl
  set_current_design top
  ```

- *value*

  A string representing the value to compare '0', '1', 'X' or 'Z'. You can use the 0b prefix to specify a multi-bit binary number or the 0x prefix to specify an hexadecimal number. The X value means to stop comparing the port.

- -after *state_transition_list*

  An optional switch-value pair used to specify a Tcl list of state transitions after which the compare is active. The compare is active in the cycle after one of the listed state transitions has happened and is off otherwise. The list of available state transitions are all the ones supported by the IEEE1149.1 state machine and state names are the same names used in SVF.

  RESET, IDLE,

  DRSELECT, DRCAPTURE, DRSHIFT, DREXIT1, DRPAUSE, DREXIT2, DRUPDATE

  IRSELECT, IRCAPTURE, IRSHIFT, IREXIT1, IRPAUSE, IREXIT2, IRUPDATE

Examples

The following example compares two bits of a bussed port during the entire scan load needed to iWrite the TDR.

```
iComparePort GPIO[3:0] 0b1XX0
iWrite tdr[7:0] 0b0
iApply
```

The following example performs the same comparison as the previous example except that the port is compared when the state machine goes from DRSelect to DRCapture, which corresponds to when the boundary scan cell captures the input ports.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
iComparePort GPIO[3:0] 0b1XX0 -after DRSELECT:DRCAPTURE
iWrite tdr[7:0] 0b0
iApply
```

Related Topics

iApply

iForcePort

iProc

iProcsForModule

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

identify_redundant_faults

Context: dft -edt, patterns -scan

Mode: analysis

Identifies redundant faults among current faults whose redundancy has not yet been determined.

Usage

identify_redundant_faults

Description

Note

By default, the create_patterns command automatically performs redundant fault identification at the conclusion of the run.

Identifies redundant faults among current faults whose redundancy has not yet been determined.

Identifies combinationally redundant (RE) faults from among those faults in the current fault list whose redundancy has not been determined yet.

This command is useful in situations where the tool has not specifically checked some faults for redundancy, but where establishing their redundancy might improve test coverage. The following are examples of such situations:

- When the tool generates patterns using named capture procedures, it does not identify RE faults. You might improve test coverage by using this command at the end of such an ATPG session.
- When the tool simulates external patterns, it does not identify RE faults among the undetected faults. You might improve test coverage by using this command after the simulation.

This command only targets faults not yet proven to be either RE or non-RE. For example, the command does not target a fault currently classified as det_simulation (DS) or that was classified as DS before but is now uncontrolled (UC) or unobserved (UO). The command is meaningful only when the current fault type is stuck-at, bridge, transition, or UDFM.

Arguments

None

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example creates patterns using named capture procedures, then identifies the RE faults:

set_system_mode analysis
set_capture_procedures on -all
create_patterns

```
...
//  There may be unidentified redundant faults when generating patterns
//     with named capture procedures defined, which may lead to lower test
//     coverage being reported.
//  To perform redundant fault identification, invoke the command:
//     identify_redundant_faults
...
Statistics Report
---------------------------------------- ---
                           #faults   #faults
Fault Classes              (coll.)   (total)
-----------------------    -------   -------
  FU (full)                 471724    783223
  -----------------------   ------   -------
  UC (uncontrolled)             39       173
  UO (unobserved)             2748      4490
  DS (det_simulation)       380299    652427
  DI (det_implication)       66661     88877
  PU (posdet_untestable)        26        36
  PT (posdet_testable)         147       223
  BL (blocked)                1907      3010
  AU (atpg_untestable)       19897     33987
-------------------------------------------
Coverage
  ----------------------------
  test_coverage              95.17%    95.05%
  fault_coverage             94.79%    94.68%
  atpg_effectiveness         99.41%    99.40%
-----------------------------------------
#test_patterns                        2736
    #clock_sequential_patterns        2736
#simulated_patterns                   2880
CPU_time (secs)                      793.1
-----------------------------------------
```

identify_redundant_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  deterministic ATPG invoked with comb/seq abort limit = 0/2
//  ---------------------------------------------------------------------
//  Performing redundant fault identification for 22857 faults
//  ---------------------------------------------------------------------
//  # red.   # non-red.  # abort   # remn.   progress  test      process
//  faults     faults     faults    faults             coverage  CPU time
//    261       12977        97       9619    57.92%   95.22%     3.26 sec
//   1361       16215       421       5281    76.90%   95.45%     4.80 sec
//   2846       18269       849       1742    92.38%   95.75%     6.67 sec
//   3121       18896       922          0   100.00%   95.81%     6.89 sec
//
```

report_statistics

```
Statistics Report
--------------------------------- ----------
                            #faults   #faults
Fault Classes               (coll.)   (total)
-----------------------     -------    ------
  FU (full)                  471724    783223
  -----------------------   -------    ------
  UC (uncontrolled)              39       173
  UO (unobserved)              2290      3832
  DS (det_simulation)        380299    652427
  DI (det_implication)        66661     88877
  PU (posdet_untestable)         26        36
  PT (posdet_testable)          145       221
  BL (blocked)                 1907      3010
  RE (redundant)               3121      4322
  AU (atpg_untestable)        17236     30325
-----------------------     -------   -------
Coverage
  ----------------------------
  test_coverage               95.81%    95.58%
  fault_coverage              94.79%    94.68%
  atpg_effectiveness          99.51%    99.49%
-----------------------------------------
#test_patterns                          2736
    #clock_sequential_patterns          2736
#simulated_patterns                     2880
CPU_time (secs)                        802.8
-----------------------------------------
```

Example 2

The following example shows the command flow to identify the redundant faults after simulating external patterns that were created using named capture procedures:

set_system_mode analysis
read_patterns my_external_patt
simulate_patterns
identify_redundant_faults

Related Topics

create_patterns

report_statistics

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_capture_procedures

set_fault_type

simulate_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iForcePort

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iTopProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Enables you to force a constant value to an arbitrary primary input port or bidirectional port inside an ijtag pattern set.

Usage

iForcePort ***object_spec***{*value* [-not_persistent] | -release } [-after *state_transition_list*]

Description

Enables you to force a constant value to an arbitrary primary input port or bidirectional port inside an ijtag pattern set.

An iComparePort or iForcePort command is only permitted in an iTopProc and not in a normal iProc as it is not retargetable upward.

It is best to keep the iProcs that only interact with the ijtag ports separated from the ones using the iForcePort command so that you maintain the ability to retarget the bulk of your PDL.

The effect of the iForcePort command is applied on the next iApply command or the next iRunLoop command. You can use the -after switch to delay the application of the value to the port and have it appear only after a specific state transition.

An iForcePort command without the -not_persistent switch remains active until one of the following events happen:

- The iForcePort is overwritten with another iForcePort command or — in case of bidirectional ports — with another iComparePort command.
- The iForcePort is explicitly terminated by means of the -release switch.
- The pattern set is closed and the next pattern set is opened without the -no_initial_ireset switch. (Note: this only applies to the context "patterns -ijtag").
- If an iForcePort command is part of an iTopProc called from within an iMerge block, then the lifetime of the iForcePort command is limited to the retargeting of that iTopProc.

  This is done for the following reason: If there are conflicting iForcePort specifications in concurrent iCalls, the associated iTopProcs must be retargeted in series. If the iForcePort specifications survived the end of the iTopProc, the active iForcePort after

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

the iMerge -end would depend on the (undefined) order of the retargeting of the concurrent iCalls. This non-deterministic behavior must be avoided.

You can use this command in an iCall used as part of test_setup for ATPG, however, it makes the iTopProc un-retargetable upward. You should avoid doing so until you reach the chip level.

Arguments

- ***object_spec***

  A collection containing the top level input or bidirectional port/ports. If you are using this command on a port that is not described in the top ICL module (a non-IJTAG port), you must have read the interface view of the top level Verilog or VHDL module to get the list of ports available to force. Use “read_verilog -interface_only” or “read_vhdl -interface_only” to read in the port list prior to issuing the set_current_design command as shown here:

  ```
  set_context patterns -ijtag -no_rtl
  read_verilog top.v -interface_only
  read_icl top.icl
  set_current_design top
  ```

  If the port is associated to a pulse waveform in the timeplate, you can only force the port to the off value.

  Asynchronous clocks cannot be forced to any value.

  ICL ports that are part of a ScanInterface cannot be forced to any value.

  ICL ports that are involved in the scan path configuration of active parts of the ICL network cannot be forced to any value.

  If there is a CT0, CT1 or CTZ constraint on the port, then the iForcePort command can only be used in combination with the matching value.

- *value*

  A string representing the value to force as ‘0’, ‘1’, ‘X’ or ’Z’. You can use the 0b prefix to specify a multi-bit binary number or the 0x prefix to specify a hexadecimal number. Just like with the iWrite command, the X value has special meaning. If you use the iForcePort command on a bused port and any of the bits is specified with an X, it means to leave that bit unchanged and preserve its previous value as if the bit was not even part of the iForcePort command as shown here:

  ```
  iForcePort GPIO[3:0] 0b0xx1
  ```

  is equivalent to

  ```
  iForcePort GPIO[3] 0b0
  iForcePort GPIO[0] 0b1
  ```

- -not_persistent

  An optional switch that specifies that the forced value can be overwritten by the IJTAG solver if needed during later retargeting operations. This option has no effect when the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iForcePort command is used on a port that is not described in ICL as the solver never interacts with those ports.

- -release

  An optional switch that specifies that the specified bits are now no longer persistent and the solver is enabled to change their values if needed during future retargeting operations. Using "iForcePort P1 -release" is equivalent to issuing "iForcePort P1 <current_value> -not_persistent" but it does not require you to know the current value.

- -after *state_transition_list*

  An optional switch-value pair used to specify a Tcl list of state transitions after which the value is applied to the port. The list of available state transitions are all the ones supported by the IEEE1149.1 state machine, and state names are the same names used in SVF.RESET, IDLE, DRSELECT, DRCAPTURE, DRSHIFT, DREXIT1, DRPAUSE, DREXIT2, DRUPDATE, IRSELECT, IRCAPTURE, IRSHIFT, IREXIT1, IRPAUSE, IREXIT2, IRUPDATE

Examples

Example 1

The following example forces 2 bits of a bussed port:

iForcePort GPIO[3:0] 0b1XX0
iApply

It is equivalent to specifying

```
iForcePort GPIO[3] 0b1
iForcePort GPIO[0] 0b0
iApply
```

Example 2

The following example applies the value 1 to GPIO[3] when the state machine goes from Shift-DR to Exit1-DR and the value 0 to GPIO[0] when the state machine goes from Exit1-DR to Update-DR.

iForcePort GPIO[3] 0b1 -after DRSHIFT:DREXIT1
iForcePort GPIO[0] 0b0 -after DREXIT1:DRUPDATE
iApply

Related Topics

add_clocks

iApply

iComparePort

iProc

iProcsForModule

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

open_pattern_set

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iMerge

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Specifies the beginning and the end of a set of iCall commands that are meant to be processed in parallel, as much as possible, in the final representation of the test patterns.

Usage

iMerge {**-begin**[-error_on_conflict] | **-end**}

Description

Specifies the beginning and the end of a set of iCall commands that are meant to be processed in parallel, as much as possible, in the final representation of the test patterns.

iCall is the only PDL command which is permitted in between iMerge -begin and iMerge -end.

In the iProcs called by the iCalls, Tessent IJTAG identifies any resource conflicts, such as two PDL commands writing to the same register, and all conflicts arising from reservations done by means of the command iTake. All conflicting commands are then processed serially. For the remaining iCall commands, Tessent IJTAG tries to find an optimal parallel solution.

Arguments

- **-begin**

  A required switch that marks the beginning of the set of iCall commands to be processed in parallel.

- -error_on_conflict

  An optional switch that specifies to issue an error when there is a conflict rather than process the conflicting events serially. In the case of nested iMerge blocks, you only need to specify this switch on the outermost iMerge specification; the specification applies to all iMerge processes that are called from within the block.

  Refer to "iMerge Conflict Reporting" in the *Tessent IJTAG User's Manual* for more information.

- **-end**

  A required switch that marks the end of the set of iCall commands to be processed in parallel.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Return Values

None

Examples

```
iMerge -begin -error_on_conflict
iCall proc1
iCall proc2
iCall proc3
…
iMerge -end
```

Related Topics

iCall

iTake

iPrefix

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

import_clocks

Contexts: patterns, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: setup

Imports functional clocks defined during the pre-DFT design rule checking.

Usage

import_clocks [-core *core_name*] [-verbose]

Description

This command imports functional clocks defined during the pre-DFT design rule checking—see "Pre-DFT Clock Rules (DFT_C Rules)" on page 3280.

This command is called in setup mode before you perform design rule checks. Use this command to import functional clocks that you already defined during the pre-DFT design rule checking if you are doing transition fault capture with these clocks. There is no need to use this command for stuck-at fault patterns.

If the tool encounters a missing clock port while running this command, it reports a warning about any missing ports and imports all existing clocks.

Note

There is no need to use this command if you use the "import_scan_mode -fast_capture_mode on" command. However, if you are using third-party scan insertion, there is no scan_mode. So, you must use this command to include the functional clocks for correct transition fault patterns in Verilog simulations.

Arguments

- -core *core_name*

  An optional switch that specify the core name from which functional clocks should be imported. If the -core switch is not specified, the tool attempts to match a core description against the module name of the specified module(s) or instance(s). If the tool does not find a direct match for a core description and a module or instance, the tool attempts to identify a match using the tool's default module matching options, which are consistent with what synthesis tools typically use. For information on the default module matching options and how they can be changed, see the description for the "set_module_matching_options -suffix_pattern_list" option.

  If the tool is unable to match a specified module or instance, it issues an error with a reminder that the user can explicitly specify which core description to use with the -core switch.

- -verbose

  An optional switch that outputs a note to stdout for each clock added by the command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

import_dfm

Context: patterns -scan_diagnosis

Mode: analysis

Imports a Calibre-compatible results database (RDB) file into a violations database (VDB).

Usage

import_dfm **-rdb** ***rdb_file*** **-layer** ***layer_name*** **-type** {**cell** | **open** | **bridge** | **cell_open** | **cell_bridge** | **cell_ton** | **cell_toff**} [-sample *n*] [-replace]

Description

Imports a Calibre-compatible results database (RDB) file into a violations database (VDB).

The import_dfm command verifies that the DFM rule violations in the VDB match the layout geometries in the LDB. It then populates the VDB with the DFM rule violations.

Note

DFM shapes that fail verification are still imported into the VDB and used during annotation.

You must have previously opened the LDB and VDB with the open_layout command. Otherwise, Tessent Diagnosis issues an error message.

For more information, see "Diagnosis for Design for Manufacturability Analysis."

Arguments

- **-rdb** ***rdb_file***

  A required switch and string pair that specifies the Calibre-compatible DFM RDB file.

- **-layer** ***layer_name***

  A required switch and string pair that specifies the layer at which the polygons in the RDB file should be compared to diagnosis suspect bounding boxes. The layer name must match one of the layers that appear in the diagnosis report when you have enabled at least one of layout-aware diagnosis and cell-aware diagnosis.

  For cell suspects, the layer name is always CELL. For open and bridge suspects, the layer names come from the LEF/DEF files. For cell_open, cell_bridge, cell_ton, and cell_toff suspects, the layer names come from the Tessent CellModelGen UDFM file.

  Note

  You can override the default layer names used in the UDFM file. For more information, see the *Tessent CellModelGen Tool Reference*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-type** {**cell** | **open** | **bridge** | **cell_open** | **cell_bridge** | **cell_ton** | **cell_toff**}

  A required switch and literal that specifies the type of DFM violation that you want to report, where cell_open, cell_bridge, cell_ton, and cell_toff only apply to cell-aware defects for cell-aware diagnosis. The cell_bridge violation supports inter-layer bridge defects.

- -sample *n*

  An optional switch and string pair that specifies a percentage value from 0 to 100. This value specifies the number of shapes that are verified against the LDB. Setting this value to 0 prohibits verification.

  This switch has no effect on diagnosis. Use it to improve verification runtimes with the caveat that lowering the value decreases the verification accuracy.

- -replace

  An optional switch that specifies to remove from the VDB the DFM rule with the same rule name, and layer, and to replace it with the new RDB information.

Examples

The following example opens an LDB, creates the directory for Tessent Diagnosis to write the VDB into, and then imports the specified RDB files into the VDB created in the *design.ldb* directory. For OPEN violations, it verifies 10 percent of the rule violations in the VDB during verification. In addition, it imports inter-layer bridge violations for layers M0-M1.

```
open_layout design.ldb

import_dfm -rdb id_100.rdb -layer route_2 -type OPEN -sample 10 -replace
import_dfm -rdb inter_layer_bridges_M0_M1.rdb -layer M0-M1 -type CELL_BRIDGE -replace
```

Related Topics

annotate_diagnosis

delete_dfm

report_dfm_rules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

import_patterns_from_svf

Context: patterns -ijtag

Mode: analysis

Enables you to integrate the actions described in the SVF file into the currently open pattern set.

Usage

import_patterns_from_svf ***filename*** [-scan_interface *scan_interface*]
[-maintain_model_state on | off]

Description

Enables you to integrate the actions described in the SVF file into the currently open pattern set.

The import_patterns_from_svf command can only be used if a pattern set is currently open. The appropriate actions from the SVF file are appended to the existing retargeted actions. The internal state of the ICL network is updated accordingly. All comments in the SVF file are converted into iNote statements unless the comment starts with !!, in which case they are ignored.

The import_patterns_from_svf command can be used either if there is a complete ICL description without any blackboxes, or if the top module is a blackbox. import_patterns_from_svf cannot be used if one of the instances, which are accessed by the SVF patterns, is a blackbox.

An ICL blackbox module is a module that contains only ports, scan interfaces and organizational elements like parameters, attributes, enums and aliases, but none of the following:

- DataRegister or ScanRegister
- DataMux, ClockMux or ScanMux
- LogicSignal
- OneHotDataGroup or OneHotScanGroup
- Instance
- AccessLink
- The Source property of an output port
- The Enable property of an output port

If the imported SVF file contains SIR or SDR commands, and you create retargeted PDL with the "write_patterns -pdl" command, the output PDL contains illegal references to blackboxed register names (tessent_blackbox_data_register, tessent_blackbox_instruction_register) if the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

top module is a blackbox. These register names can also be found in the results of the get_pattern_set_data command.

The import_patterns_from_svf command cannot be used in an iMerge block or in an iProc. At the time when this command is used, there must not be any queued iWrite, iRead, iForcePort or iComparePort commands.

The import_patterns_from_svf command has the following limitations:

- The SVF import must not leave the TAP controller in RESET state.
- ENDDR IRPAUSE and ENDDR RESET are not supported. The SVF command ENDDR can be used with the arguments DRPAUSE and IDLE only.
- ENDIR DRPAUSE and ENDIR RESET are not supported. The SVF command ENDIR can be used with the arguments IRPAUSE and IDLE only.
- The SVF command RUNTEST can only be used in its simplest form: RUNTEST *run_count* [TCK|SCK]. The *run_state*, *min_time*, *max_time* and *end_state* arguments are not supported.
- The commands RUNTEST, SDR and SIR cannot be used if the TAP controller is in RESET state.
- The STATE command does not accept several arguments, that is, it is not possible to specify the TAP state transitions the TAP controller should choose to get to the specified end state. Only the end state itself can be specified.
- TRST Z is not supported. TRST only accepts ON, OFF and ABSENT.
- Native SVF does not have a possibility to express the measurement of high impedance at an output port. Therefore write_patterns -svf creates a dedicated annotation for that purpose: “!TESSENT_PRAGMA expect_z”. This annotation is not understood by import_patterns_from_svf, it is simply ignored.

Arguments

- ***filename***

  A required string that specifies the name of a SVF file to be read. The file can be gzip-compressed or uncompressed

- -scan_interface *scan_interface*

  An optional switch and string pair that specifies the name of the TAP scan interface in the ICL top module. If this switch is omitted, there must be a unique TAP scan interface in the ICL top module. A scan interface is identified as a TAP scan interface by the existence of a TMS port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

A pattern set can have active scan interfaces and inactive scan interfaces. If there are several TAP scan interfaces, but only one of them is active, then the -scan_interface switch can be omitted.

- -maintain_model_state on | off

  An optional switch and literal pair that specifies how the SVF reader operates. When set to off, the reader operates in "blackbox mode", in that it does not apply the SDR (Scan Data Register) and SIR (Scan Instruction Register) commands to the internal state of the IJTAG network, nor does it check if the internal state of the IJTAG network matches the length of the provided SDR/SIR data. The default value is on.

Examples

The following example converts an SVF file to a PDL file:

```
SETUP> set_context patterns -ijtag -no_rtl
SETUP> read_icl data/icl/*.icl
SETUP> set_current_design chip
SETUP> add_clocks ClkA -pulse_always -period 10ns
SETUP> set_system_mode analysis
ANALYSIS> open_pattern_set test1 -tester_period 100ns
ANALYSIS> import_patterns_from_svf jtagtest1.svf
ANALYSIS> close_pattern_set
ANALYSIS> write_patterns jtagtest1.pdl -pattern_sets test1 -pdl
ANALYSIS> exit -f
```

Related Topics

iComparePort

iForcePort

iMerge

iProc

iRead

iWrite

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

import_scan_mode

Context: dft -edt, dft -edt -logic_bist, dft -test_points, patterns -scan

Mode: setup

Imports configuration settings from tcd_scan.

Usage

import_scan_mode [-core *core_name*] [*scan_mode_name*] [-no_reset_design] [-fast_capture_mode {on | off}] [-verbose] [-silent]

Description

Imports the following configuration settings from tcd_scan:

- Adds scan chains used in the scan mode.
- Adds EDT instruments used in the scan mode.
- Adds OCC instruments used in the scan mode.
- Configures scan clocks used in the scan mode.
- Configures DFT signals used in the scan mode.
- Adds scan clock pulses to shift procedure.
- Adds scan enable signal forces to shift procedure.

Note

To override default parameter values of imported scan instruments, use the set_core_instance_parameters command.

By default, the tool resets the design when you run the import_scan_mode command unless you specify the -no_reset_design switch. In addition, the tool uses an auto-generated test procedure file. Alternatively, you can use a specific test procedure file with the set_procfile_name command, which you specify after using the import_scan_mode command.

Arguments

- -core *core_name*

  An optional switch that specify the core name from which the scan mode should be imported. If the -core switch is not specified, the tool attempts to match a core description against the module name of the specified module(s) or instance(s). If the tool does not find a direct match for a core description and a module or instance, the tool attempts to identify a match using the tool's default module matching options, which are consistent with what synthesis tools typically use. For information on the default module matching options and how they can be changed, see the description for the "set_module_matching_options -suffix_pattern_list" option.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the tool is unable to match a specified module or instance, it reports an error with a reminder that you can explicitly specify which core description to use with the -core switch.

- *scan_mode_name*

  An optional string that specifies the scan modes to import. This switch is only required when there are multiple scan modes. If only one scan mode is specified in the TSDB, it is imported by default.

- -no_reset_design

  An optional switch that specifies that the design should not be reset. By default, the design is reset when you use the import_scan_mode command.

- -fast_capture_mode on | off

  An optional switch that specifies using fast capture mode. The default is off.

- -verbose

  An optional switch that displays additional information about the configuration settings being imported.

- -silent

  An optional switch that specifies to suppress warning messages.

Examples

Example 1

The following example overrides an EDT instrument parameter on all EDT instruments imported with the import_scan_mode command:

```
import_scan_mode edt_mode
set_core_instance_parameters -instrument_type edt \
    -parameter_values {edt_low_power_shift_en on }
```

Example 2

This example does not reset the design and imports the design in the only mode that the TSDB has:

```
import_scan_mode -no_reset_design
```

For designs with multiple scan modes, you need to specify the *scan_mode_name* to import.

Example 3

This example imports the design in *scan_mode_name* “edt_mode”. The tool uses an auto-generated test procedure file and resets the design. The -verbose switch shows additional configuration settings being imported.

```
import_scan_mode edt_mode -verbose
```

Example 4

This example shows how the import_scan_mode command uses the int_mode as the *scan_mode_name*. Additionally, the example demonstrates how to use a specific test procedure

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

file with the set_procfile_name command, which you specify after using the import_scan_mode command:

```
import_scan_mode int_mode -verbose
set_procfile_name my_custom.testproc
```

Related Topics

report_scan_modes

set_core_instance_parameters

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

index_collection

Context: unspecified, all contexts

Mode: all modes

Extracts a single object at a specified index in a collection and creates a new collection containing only that object.

Usage

index_collection ***collection index***

Description

Extracts a single object at a specified index in a collection and creates a new collection containing only that object.

The base collection remains unchanged.

Arguments

- ***collection***

  A required string that specifies the collection to be searched.

- ***index***

  A required value that specifies the index into the collection. Permitted values are integers from 0 to "*sizeof_collection* - 1". Also permitted is the keyword "end", which is the equivalent of "sizeof_collection - 1".

Examples

The following example shows the first and last object in a collection being extracted.

set c1 [get_instances {u1 u2 u3}]

```
{u1 u2 u3}
```

get_name_list [index_collection $c1 0]

```
u1
```

get_name_list [index_collection $c1 [expr [sizeof_collection $c1] -1]]

```
u3
```

get_name_list [index_collection $c1 end]

```
u3
```

get_name_list [index_collection $c1 end-1]

```
u2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

add_to_collection

append_to_collection

compare_collections

copy_collection

filter_collection

foreach_in_collection

is_collection

range_collection

remove_from_collection

sizeof_collection

sort_collection

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iNote

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Inserts a note in the opened pattern set.

Usage

iNote [-status | -comment] ***string***

Description

The iNote command inserts a note in the opened pattern set, attached to the time corresponding to where the iNote command is run. When the pattern set is written to a pattern file using the write_patterns command, the note appears as an annotation in the pattern file at the appropriate location. For simulation testbenches, the note is displayed when the cycle containing the annotation is simulated.

Using verilog_insert Pragma Within an iNote String

The following pragma can be used as the start of the iNote string argument to insert native Verilog code inside the Verilog testbench. The Verilog code specified in the pragma is run at the time when the iNote annotation would normally be inserted. See "Example 2."

The pragma uses the following format:

> iNote "tessent_pragma verilog_insert *native verilog statement suitable within an initial block*"

The native Verilog statement following the pragma must be code that is permitted to go inside an initial block. This code must be terminated with a semicolon or the end keyword following the normal Verilog grammar rules. "Example 2" shows the insertion of a while loop and the triggering of an event.

Using iWrite/iRead Variables Pragma Within an iNote String

You can use the iNote command to define iWrite variables within the PDL to make it easier to change values on the tester using patching. You can also use it to define iRead variables within the PDL to make it easier to map TDO miscompares into meaningful information.

The "PDL Retargeting With Symbolic Variables" section of the *Tessent IJTAG User's Manual* describes the special iNote string used to define the symbolic variable and how the information is forwarded into an output patterns file such as STIL.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- -status

  An optional switch that is used to denote important milestones in the test flow.

- -comment

  An optional switch that is used to provide detailed comments on any PDL that may be used to generate test vectors.

  Note

  There is no real difference between the -status and -comment switches. They help clarify the intention of the iNote, but the behavior of the pattern is exactly the same.

  The iNote command never translates the -status and -comment switches into Annotations of the STIL file. If you want these switches translated, include them in the message string, such as iNote “-status message” or iNote -status “-status message” and likewise for -comment.

- ***string***

  The comment string or note to be added to the pattern set at the proper time. This must be a single normal Tcl string, Multiple words must be enclosed in quotation marks or braces.

Return Values

None

Examples

Example 1

In the following example, the first iNote command is attached to the beginning of the pattern. The iNote string contains a variable that is substituted before being sent to the iNote command. The second iNote command is attached to the time corresponding to the time reached to run the previous iApply actions. The last iNote appears after the loop.

```
iNote -status "Setting up controller instance $inst"
iWrite mode blue; iApply
iNote -comment “Running controller”
iWrite enable On; iApply
iRunLoop 30 -tck
iNote -comment “Checking results”
iRead results pass; iApply
```

Example 2

The following example makes use of the special insert_verilog tessent_pragma (shown in green in the example code block) to perform two tasks. As previously described, the text that follows the “tessent_pragma insert_verilog” prefix in the iNote string is native Verilog code that is

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

suitable for being inserted inside an initial Verilog block. The example uses one special iNote to insert the following line at a specific time in the testbench:

```
while (dut.cm.pll.lock !== 1'b1) #100;
```

This code while construct makes the testbench wait for the lock output pin on the PLL instance to become 1 before continuing with the rest of the operation. As described in the comments in the dofile example, this is used to mimic the effect of a STIL MatchLoop that would be used on the tester.

The second special iNote in the example is to trigger a Verilog event in the testbench:

```
-> check_clock${::clock_id_cnt}_period;
```

The event triggered by this command lets the Verilog code that checks the period of the clock do its jobs. See the “always @ (check_clock`(id)_period)” statement in the check_clock_string defined in the dofile. This string is processed by the “string map” Tcl command to replace the ‘(id), ‘(path) and ‘(period) values before being sent to a file. The tool later includes that file in the testbench using the SIM_MODULE_INCLUDE parameter in the write_design command. The SIM_INSTANCE_NAME parameter is also used in the write_design command, so the instance name of the chip within the testbench is constant, and can be used in the Verilog code inserted into the testbench. The “dut” string in the instance path shown above is the instance name supplied using the SIM_INSTANCE_NAME parameter.

```
set_context patterns -ijtag
read_design my_chip -view interface
set_current_design my_chip

add_clocks clk -period 9.5ns

set_system_mode analysis
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# This section contains a Verilog code block that is processed by the
# Tcl "string map" command to replace `(id), `(path) and `(period) by
# specific values. See string map usage in the check_lock iProc below.

set check_clock_string {
  real clock`(id)_edge1_time, clock`(id)_edge2_time,
       measured_clock`(id)_period;
  event check_clock`(id)_period;

  always @(check_clock`(id)_period) begin
    # Measure the realtime for the next two clock edges
    @ (posedge dut.`(path)) clock`(id)_edge1_time = $realtime;
    @ (posedge dut.`(path)) clock`(id)_edge2_time = $realtime;
    measured_clock`(id)_period = clock`(id)_edge2_time -
                                 clock`(id)_edge1_time;
    # Increment the compare event counter
    tb._compare_count = tb._compare_count + 1;
    $timeformat(-9, 3, " ns", 10);
    if (1.01*measured_clock`(id)_period< `(period) || \
        0.99*measured_clock`(id)_period > `(period)) begin
      # Increment the miscompare counter when the period is not within
      # 1% of the expected period.
      tb._compare_fail_count = tb._compare_fail_count+1;
      $display(
        " Clock '`(path)' has wrong period: expected %0t, actual %0t",
        `(period), measured_clock`(id)_period
      );
    end else begin
      $display(" Clock '`(path)' has expected period of %0t",
               measured_clock`(id)_period
      );
    end
  end
}
# This iProc is used to reset the PLL
iProcsForModule pll
iProc reset {} {
  iWrite rst 0b1
  iApply
  iWrite rst 0b0
  iApply
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# This iProc is used to check that the PLL is locked.
iProc check_lock {} {
  iRead lock 0b1
  iApply
  iClock ref
  iClock vco
  set icl_scope [get_icl_scope -icall]
  # Use the "hierarchical_design_instance" attribute to map the ICL pin
  # into the verilog design. Use to_verilog_path helper proc to convert
  # slashes into dots.
  set verilog_scope [::mentor::to_verilog_path \
      [lindex [get_attribute_value_list $icl_scope
                    -name hierarchical_design_instance] 0]]
  # Use get_clock_option to get expected period of PLL input and output
  # pins from the ICL description and the add_clocks period specified as
  # top of the dofile
  set ref_period \
        [expr {round([get_iclock_option ${icl_scope}.ref -period
                                        -in ns]*1000)/1000.0}]
  set vco_period \
        [expr {round([get_iclock_option ${icl_scope}.vco -period
                                        -in ns]*1000)/1000.0}]
  # Use the echo command to take result of "string map" and save it to a
  # file so that it can be inserted into the testbench. See
  # "write_patterns -parameter_list" option below where SIM_MODULE_INCLUDE
  # is used. The content of the ::check_clock_string is define above.
  echo [string map [list `(id) $::clock_id_cnt \
                         `(path) ${verilog_scope}.ref \
                        `(period) $ref_period] \
                  $::check_clock_string] > check_clock_set1.v
  # Initialize ::clock_id_cnt if it does not already exists
  if {![info exists ::clock_id_cnt]} {set ::clock_id_cnt 0}
  # Use iNote pragma to trigger the check of the clock period
  iNote "tessent_pragma verilog_insert \
         -> check_clock${::clock_id_cnt}_period;"
  incr ::clock_id_cnt
  # Repeat same method for the vco output
  echo [string map [list `(id) $::clock_id_cnt \
                        `(path) ${verilog_scope}.vco \
                        `(period) $vco_period] \
                  $::check_clock_string] >> check_clock_set1.v
  iNote "Measuring clock period of ${verilog_scope}.vco"
  iNote "tessent_pragma verilog_insert \
         -> check_clock${::clock_id_cnt}_period;"
  incr ::clock_id_cnt
}
# Create two pattern sets. In simulation they are written out together but
# in STIL they are written out separately so that P2, which checks the
# lock #of the PLL, can be used inside a STIL MatchLoop.
open_pattern_set p1
  iCall cm_pll.reset
  # Use iNote pragma to insert while loop in verilog testbench to
  # imitate the behavior of the MatchLoop. It checks the lock pin every
  # 100ns until it becomes 1.
  iNote "tessent_pragma verilog_insert \
         while (dut.cm.pll.lock !== 1'b1) #100;"
  iCall cm_pll.check_lock
  iRunLoop -time 100ns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
close_pattern_set -network_end_state initial

# Uses SIM_MODULE_INCLUDE parameter to have the file created above
# included into the testbench.
# Note that is it no longer needed to specify the SIM_INSTANCE_NAME
# parameter because the macro SIM_INSTANCE_NAME is added in the
# testbench and is used in the iNote above to get the DUT instance
# name of the chip: `SIM_INSTANCE_NAME.
write_patterns pll_setup.v -repl \
               -verilog \
               -pattern_sets {p1 p2} \
               -param_list {SIM_MODULE_INCLUDE check_clock_set1.v}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

insert_test_logic

Context: dft -scan, dft -test_points

Mode: analysis

Inserts the test structures you define into the netlist and stitches up the scan chains to increase the design's testability.

Usage

Context: dft -test_points

insert_test_logic [-verbose]
[-write_in_tsdb { on | off | auto }]
[-design_identifier *design_identifier*]
[-write_insertion_script *script_filename* [-replace]]
[-insertion [dc | genus]]

Context: dft -scan

insert_test_logic [-verbose] [-write_in_tsdb { on | off | auto }]
[-design_identifier *design_identifier*]

Context: dft -scan -test_points

insert_test_logic [-verbose]
[-write_in_tsdb { on | off | auto }]
[-design_identifier *design_identifier*]
[-write_insertion_script *script_filename* [-replace]]

Description

Inserts the test structures you define into the netlist and stitches up the scan chains to increase the design's testability.

The tool inserts test structures into a design to increase the fault coverage (testability) of the design. Specifically, the tool attempts to increase a design's fault coverage with the insert_test_logic command by doing the following:

- Identifies sequential cells and replaces them with corresponding scan cells, and stitches them together into a scan chain. (Referred to as scan cell replacement and stitching.)
- Supports the adding of both system-defined and user-defined test points.
- Supports the automatic adding of test logic.

For more information on scan replacement, test points and test logic, refer to Test Point Analysis and Insertion in the *Tessent Scan and ATPG User's Manual*. For information about handling pre-inserted scan cells, refer to "How to Specify Existing Scan Cells" in the *Tessent Scan and ATPG User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can also use the insert_test_logic command with the -write_insertion_script switch to generate insertion scripts for test points for Design Compiler (DC) or RTL Compiler (RC). For more information, see "How to Insert Test Points and Perform Scan Stitching Using Third-Party Tools" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- -verbose

  An optional switch that specifies to the tool to output detailed information when the command is unsuccessful.

- -write_in_tsdb on| off | auto

  An optional switch and literal pair that controls whether the extracted ICL is automatically written out in the specified TSDB output directory.

  The auto value means "on" when the directory returned by get_tsdb_output_directory already exists, otherwise auto means "off."

  When set to "on," the DFT inserted design is written out into the following directory:

  ```
  tsdb_output_directory/dft_inserted_designs/
  module_name_design_id.dft_inserted_design
  ```

  The *design_id* value is the one returned by get_context -design_id. See the dft_inserted_designs section of the Tessent Shell Data Base (TSDB) chapter for more information.If you have an elaborated ICL view in memory, it is exported to the new DFT inserted design directory with an updated tessent_design_id attribute value matching the current design id. The concatenated PDL file is also copied from its original source.

  When the tool has completed writing the design to the TSDB, it reads the ICL variants from the original DFT-inserted design subdirectory. All the variants are updated and written to the new DFT-inserted design subdirectory alongside the default ICL file.

- -design_identifier *design_identifier*

  An optional switch and value pair that sets the *design_identifier*.

  Note

  This switch is ignored if you have previously set the design_identifier using the "set_context -design_identifier *string*" command.

- -write_insertion_script *script_filename*

  An optional switch that enables the tool to generate the script for inserting test points for Design Compiler (DC) or Genus™ Synthesis Solution. In the combined "dft -scan -test_points" context, you must run the "set_insert_test_logic_options -generate_third_party_script" command in Setup mode before running the "insert_test_logic -write_insertion_script" command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -replace

  An optional switch that specifies to replace the output file generated by the insertion script, if the file already exists.

  This switch is only available when also using the -write_insertion_script switch.

- -insertion dc | genus

  An optional switch that specifies the target insertion tool of the insertion script.

  dc — Generates the script for the Design Compiler (DC). This is the default.

  genus — Generates the script for Genus Synthesis Solution.

  This switch is not available in combined “dft -scan -test_points” context, where you can specify the target insertion tool using the “set_insert_test_logic_options -generate_third_party_script” command.

Related Topics

add_scan_instances

add_tied_signals

analyze_wrapper_cells

report_scan_chains

set_scan_signals

set_test_logic

set_test_point_insertion_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

intercept_connection

Context: dft

Mode: insertion

A command that uses the get_dft_cell command to obtain a cell with the specified function name and uses it to intercept a connection to a pin, port, or net.

Usage

intercept_connection **node -cell_function_name** ***function_name***
[-technology *dft_cell_selection_name*]
[-input2 *input2_source*] [-select *select_source*]
[-leaf_instance_prefix *prefix_name*]
[-only_when_has_functional_source]

Description

A command that uses the get_dft_cell command to obtain a cell with the specified function name and uses it to intercept a connection to a pin, port, or net.

This command performs interception inside both uniquified and not uniquified modules.

Note

The intercept_connection command does not uniquify modules. You are responsible for module uniquification before using this command.

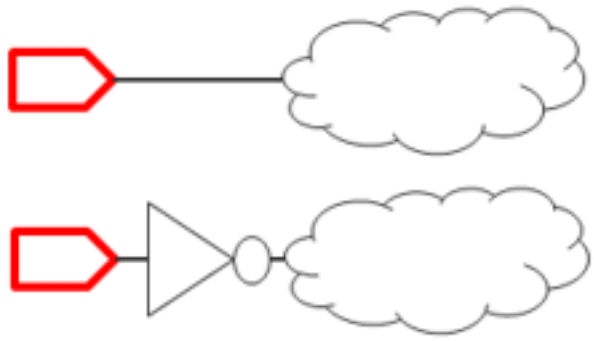
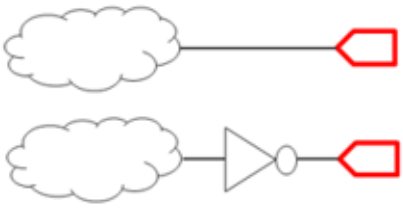
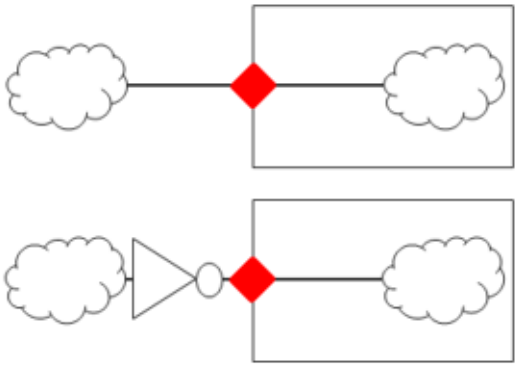
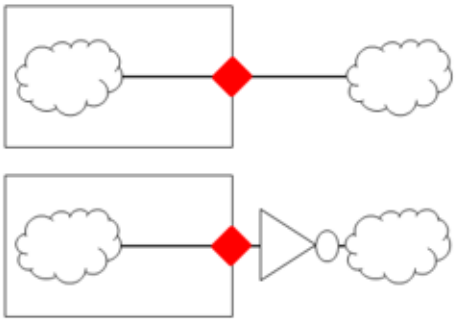
Arguments

- **node**

  A required value that is either a port, pin, or net name in the design or a collection of one port, pin, or net object. Based on the type of object and its direction attribute, in the case of a pin or a port, the source or destination connection of the object is intercepted. Table 5-25shows the interception that is performed based on the node type. Note that an error is generated if the node is an inout pin or port, or if the node is a net corresponding to an inout pin or port.

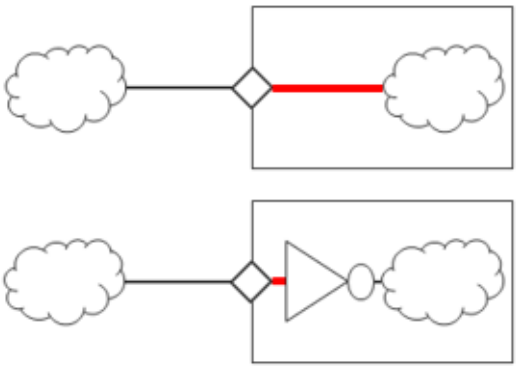
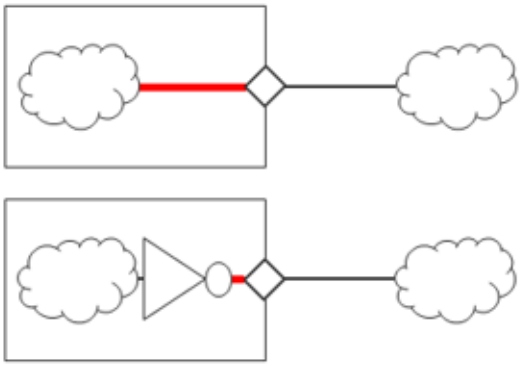
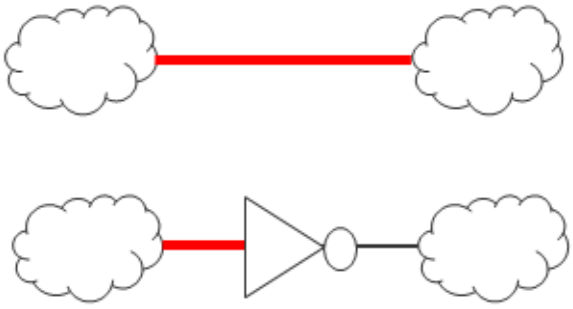
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-25. Interception Based on Node Type

Node Type	Interception Done	Illustration
Input port	The node is a driver. The intercepting cell ends up between the input port and its original fanout.	
Output port	The node is a sink. The intercepting cell ends up between the original fan-in and the output port	Before and after interception: 
Input pin	The node is a sink. The intercepting cell ends up between the original fan-in and the input pin	Before and after interception: 
Output pin	The node is a driver. The intercepting cell ends up between the output pin and its original fanout.	Before and after interception: 

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-25. Interception Based on Node Type (cont.)

Node Type	Interception Done	Illustration
Net corresponding to an input pin or an input port	The node is a driver. The intercepting cell ends up between the input pin or port and its original fanout.	Before and after interception: 
Net corresponding to an output pin or an output port	The node is a sink. The intercepting cell ends up between the original fan-in and the output pin or port.	Before and after interception: 
Net not corresponding to a pin or a port	The node is a driver. The intercepting cell ends up between the driver of the net and its original fanout. Only single driver nets can be intercepted.	Before and after interception: 

- **-cell_function_name *function_name***

  A required switch and string pair that specifies the function of the cell to be used for the interception. Table 5-26 shows the permitted cell function names and how the cell function affects the need to specify the -input2 and -select options.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-26. Permitted Cell Function Names

Function Name	Results
and	The interception is performed with the 2 input AND cell returned by the "get_dft_cell and" command. The -input2 option is required to specify the source of the second input. In the *Tessent Cell Library Manual*, see AND Cells for more details about the "and" cell_type and Cell Selection to know how to control which AND cell is used based on the specified -technology option.
buffer	The interception is performed with the BUFFER cell returned by the "get_dft_cell buffer" command. In the *Tessent Cell Library Manual*, see Buffer and Input Pad for more details about the "buffer" cell_type and Cell Selection to know how to control which BUFFER cell is used based on the specified -technology option.
inverter	The interception is performed with the INVERTER cell returned by the "get_dft_cell inverter" command. In the *Tessent Cell Library Manual*, see Inverter for more details about the "inverter" cell_type and Cell Selection to know how to control which INVERTER cell is used based on the specified -technology option.
mux	The interception is performed with the MUX cell returned by the "get_dft_cell mux" command. The -input2 and -select options are required to specify the source of the second data input and the select input. In the *Tessent Cell Library Manual*, see MUX Cell for more details about the "mux" cell_type and Cell Selection to know how to control which MUX cell is used based on the specified -technology option.
or	The interception is performed with the 2 input OR cell returned by the "get_dft_cell or" command. The -input2 option is required to specify the source of the second input. In the *Tessent Cell Library Manual*, see OR Gate for more details about the "or" cell_type and Cell Selection to know how to control which OR cell is used based on the specified -technology option.
clock_and	The interception is performed with the 2 input clock AND cell returned by the "get_dft_cell clock_and" command. The -input2 option is required to specify the source of the second input. In the *Tessent Cell Library Manual*, see AND Cells for more details about the "clock_and" cell_type and Cell Selection to know how to control which AND cell is used based on the specified -technology option. The clock type is not learned and must be explicitly specified in the library. If one of the inputs has the function clock_in, it is selected for the interception.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-26. Permitted Cell Function Names (cont.)

Function Name	Results
clock_buffer	The interception is performed with the clock BUFFER cell returned by the "get_dft_cell clock_buffer" command. In the *Tessent Cell Library Manual*, see Buffer and Input Pad for more details about the "clock_buffer" cell_type and Cell Selection to know how to control which BUFFER cell is used based on the specified -technology option. The clock type is not learned and must be explicitly specified in the library.
clock_inverter	The interception is performed with the clock INVERTER cell returned by the "get_dft_cell clock_inverter" command. In the *Tessent Cell Library Manual*, see Inverter for more details about the "clock_inverter" cell_type and Cell Selection to know how to control which INVERTER cell is used based on the specified -technology option. The clock type is not learned and must be explicitly specified in the library.
clock_mux	The interception is performed with the MUX cell returned by the "get_dft_cell clock_mux" command. The -input2 and -select options are required to specify the source of the second data input and the select input. In the *Tessent Cell Library Manual*, see MUX Cell for more details about the "clock_mux" cell_type and Cell Selection to know how to control which MUX cell is used based on the specified -technology option.
clock_or	The interception is performed with the 2 input clock OR cell returned by the "get_dft_cell clock_or" command. The -input2 option is required to specify the source of the second input. In the *Tessent Cell Library Manual*, see OR Gate for more details about the "clock_or" cell_type and Cell Selection to know how to control which OR cell is used based on the specified -technology option. The clock type is not learned and must be explicitly specified in the library. If one of the inputs has the function clock_in, it is selected for the interception.

- -technology *dft_cell_selection_name*

  An optional switch and value pair that specifies from which dft_cell_selection list the cell is to be used. You only need to specify this option if your library has more than one dft_cell_selection wrapper. If you have none, one is learned for the simple cell types, but you have no access to the clock cells unless you have set their cell_type attribute in the library. See "Model-Level Attribute Descriptions" in the *Tessent Cell Library Manual* for information on how to specify the cell_type attribute.

- -input2 *input2_source*

  An optional switch that specifies a node to connect to the second input of the cell. The source must be a valid pin, port, or net object, or a name, or the constant value 0 or 1. You should use a constant value if you want to insert a gate now and then connect the extra pins at some later time. Table 5-26 on page 1343 describes which values for -cell_function_name makes the specification of this option mandatory.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -select *select_source*

  An optional switch and value pair that specifies a node to connect to the select input of the cell. The source must be a valid pin, port, or net object, or a name, or the constant value 0 or 1. You should use a constant value if you want to insert a gate now and then connect the extra pins at some later time. Table 5-26 describes which values for -cell_function_name make the specification of the -select option mandatory.

- -leaf_instance_prefix *prefix_name*

  This optional switch and value pair specifies the prefix to used when instantiating the cell. When this is not specified, the default is “insertion_”. The leaf instance name is constructed by concatenating the prefix_name with the cell_function_name. When a conflict is detected during instantiation, the tool automatically adds a “_<int>” suffix to uniquify the leaf name. You can use a specific prefix if you want to easily find the inserted cells in a synthesis or layout script by using a command such as “get_cells *prefix_name* -hier”.

- -only_when_has_functional_source

  This optional switch specifies to skip the instantiation of the cell and interception of the node when the attribute has_functional_source is 0 for the node. Instead, the node is disconnected and driven by the specified -input2 value. You cannot specify this option when -cell_function_name refers to a single input cell as there is no -input2 value to connect to in that case.

Examples

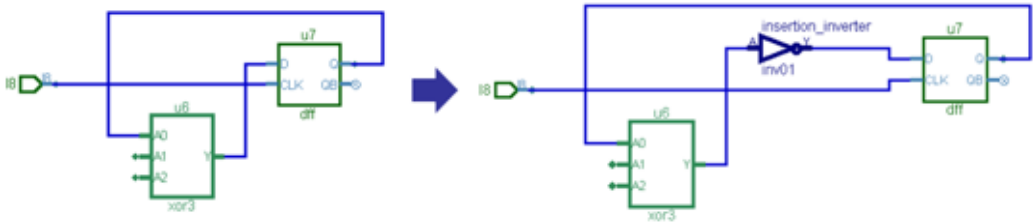
Example 1

The following example inserts a simple inverter gate by intercepting the existing connection between the u7/D pin and the u6/Y pin, shown on the left side of Figure 5-7. After the interception, the inverter A pin is connected to the u6/Y pin and the inverter Y pin is connected to the u7/D pin as shown on the right side of Figure 5-7.

INSERTION> intercept_connection u7/D -cell_function_name inverter

```
{insertion_inverter}
```

Figure 5-7. Inverter Interception



Example 2

The following example inserts a multiplexer and specifies to connect the second multiplexer input pin to 1, and to connect the multiplexer select pin to scan_en.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
INSERTION> intercept_connection u4/u3/D -cell_function_name mux -input2 1 \
           -select scan_en
{u4/insertion_mux_1}
```

This example does the same thing as first example above but specifies the -only_when_has_functional_source option. Because this option is specified, the tool connects the pin u1/u3/D directly to a constant 1 if its has_functional_source attribute evaluates to 0. In this case, the commands returns null as no intercepting cell was instantiated.

```
INSERTION> intercept_connection u4/u3/D -cell_function_name mux -input2 1 \
           -select scan_en -only_when_has_functional_source
{}
```

Example 3

The following example script shows how to cascade three gates on a set of nodes. The example illustrates how the command is capable of handling non-unique designs. The script works whether module “rep3” is uniquified or not. Note how the example inserts the first gate level on all nets, and then adds two more levels in a row on each net. This illustrates that it does not matter if you insert all gate levels on each net or if you add one level at a time. In both cases, the result is identical.

Original design:

```
module ex (ina, outa, c1, c2, c3);
  input c1, c2, c3;
  input [3:0] ina;
  output [3:0] outa;
  rep3 rep3_0 (.ina(ina[0]), .outa(outa[0]));
  rep3 rep3_1 (.ina(ina[1]), .outa(outa[1]));
  rep3 rep3_2 (.ina(ina[2]), .outa(outa[2]));
  rep1 rep1 (.ina(ina[3]), .outa(outa[3]));
endmodule
module rep3 (ina, outa);
  input ina;
  output outa;
  assign outa = ina;
endmodule
module rep1 (ina, outa);
  input ina;
  output outa;
  assign outa = ina;
endmodule
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The dofile:

```
set nets [get_nets -of_pins [get_pins -direction input]]
foreach_in_collection net $nets {
  intercept_connection $net -cell or -input2 c1}
foreach_in_collection net $nets {
  foreach var {2 3} {
   intercept_connection $net -cell [lindex {and or} [expr $var-2]] \
            -input2 c$var -leaf_inst_prefix myprefix_
  }
}
```

The modified design:

```
module ex (ina, outa, c1, c2, c3);
  input c1, c2, c3;
  input [3:0] ina;
  output [3:0] outa;
  rep3 rep3_0 (.ina(ina[0]), .outa(outa[0]), .c1(c1), .c2(c2), .c3(c3));
  rep3 rep3_1 (.ina(ina[1]), .outa(outa[1]), .c1(c1), .c2(c2), .c3(c3));
  rep3 rep3_2 (.ina(ina[2]), .outa(outa[2]), .c1(c1), .c2(c2), .c3(c3));
  rep1 rep1 (.ina(ina[3]), .outa(outa[3]), .c1(c1), .c2(c2), .c3(c3));
endmodule
module rep3 (ina, outa, c1, c2, c3);
  input ina;
  output outa;
  input c1, c2, c3;
  wire c1, c2, c3;
  wire ina_ts1, ina_ts2;
  OR210 insertion_or(.A(ina_ts1), .B(c1), .Y(outa));
  AN210 myprefix_and(.A(ina_ts2), .B(c2), .Y(ina_ts1));
  OR210 myprefix_or(.A(ina), .B(c3), .Y(ina_ts2));
endmodule
module rep1 (ina, outa, c1, c2, c3);
  input ina;
  output outa;
  input c1, c2, c3;
  wire c1, c2, c3;
  wire ina_ts1, ina_ts2;
  OR210 insertion_or(.A(ina_ts1), .B(c1), .Y(outa));
  AN210 myprefix_and(.A(ina_ts2), .B(c2), .Y(ina_ts1));
  OR210 myprefix_or(.A(ina), .B(c3), .Y(ina_ts2));
endmodule
```

Related Topics

get_dft_cell

move_connections

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iOverrideScanInterface

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Imposes user-specified behavior on the operation of a ScanInterface.

Usage

iOverrideScanInterface ***scan_interface_list*** [-capture {on | off}] [-update {on | off}] [-broadcast {on | off}]

Description

Imposes user-specified behavior on the operation of a ScanInterface.

The iOverrideScanInterface command imposes user-specified behavior on the following three aspects of the operation of a ScanInterface:

- Whether or not it permits the capture of new data into the scan registers connected to that interface.
- Whether or not it permits the update of new data shifted into the scan registers connected to that interface.
- Whether or not it participates in scan broadcasting.

Arguments

- ***scan_interface_list***

  A required list of ICL ScanInterfaces to which the following settings apply. Only ordinary ScanInterfaces are permitted, no TAP ScanInterfaces, because the TAP state machine does not permit omitting the Capture-DR/Capture-IR cycle or the Update-DR/Update-IR cycle.

  To be able to disable the capture or update functionality on a ScanInterface of an internal instrument, the corresponding CaptureEnPort or UpdateEnPort of the ScanInterface need to be suppressible through a DataMux. The IJTAG engine automatically performs the necessary operations to suppress the capture or update enable signal before the operations of the next iApply.

- -capture {on | off}

  An optional switch and literal pair that specifies whether or not the affected scan registers capture data from their parallel inputs, that is, from their specified CaptureSource. If this switch is set to off, the retargeting software ensures that the affected registers cannot be used

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

to observe iRead targets on their parallel inputs. Either a different possibility to observe the target can be found or the retargeting software issues an error message.

The "affected registers" are those registers that are on the path between the ScanInPort of a specified client ScanInterface and the corresponding ScanOutPort of that ScanInterface.

At startup time, the capture functionality is switched on for all scan interfaces. This default setting is reapplied by the iReset command and when a pattern set is opened without the -no_initial_ireset switch.

- -update {on | off}

  An optional switch and literal pair that specifies whether or not the affected scan registers pass on their serially loaded data to their parallel outputs. If this switch is set to off, the retargeting software ensures that the affected registers cannot be used to justify a certain value at their parallel outputs. Either a different possibility to justify the value can be found, or the retargeting engine issues an error message.

  At startup time, the update functionality is switched on for all scan interfaces. This default setting is reapplied by the iReset command and when a pattern set is opened without the -no_initial_ireset switch.

- -broadcast {on | off}

  An optional switch and literal pair that indicates to the IJTAG retargeting engine whether the affected broadcast group (as described in the IEEE 1687-2014 standard) is activated or deactivated.

  In an IJTAG network that supports broadcast, two steps are necessary to utilize its broadcast functionality. First, the network needs to be configured into the correct state for broadcast, for example, through an iProc command. Secondly, you must run the "iOverrideScanInterface -broadcast on" command to inform the IJTAG engine about the active broadcast group. This command requires the list of Scan Interfaces considered as a broadcast group. The IJTAG engine then keeps track of the network state in all broadcast locations to know the current state for later reference.

  The ICLNetworkVerify patterns do not exercise or verify the IJTAG broadcast feature. Therefore, test patterns that utilize IJTAG broadcast must be simulated with care to ensure that the hardware implementation matches the implied behavior when using this switch.

  Broadcasting conforms to IEEE 1687-2014, except Tessent does not impose the rule that you must identify each scan interface in the AllowBroadcastOnScanInterface list in the ICL of the associated ICL instance.

  You must enable broadcast mode by running the "iOverrideScanInterface -broadcast on" command, because the retargeting software does not do it for you. Broadcast mode remains in effect until you disable it by running "iOverrideScanInterface -broadcast off". When broadcasting, the command should contain only a list of scan interfaces having ScanInPort sharing a common source. You should configure multiplexers in the path between the common source and the ScanInPorts to select the path to a common source in the scan chain.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Broadcasting involves one target scan interface and one or multiple shadow scan interfaces. All scan registers behind the scan interfaces receive a copy of the scan data. A broadcast group's hardware implementation must activate all the scan interfaces' select ports when the target interface is on the active scan path. The scan chain behind an active target interface must be longer or equal in size to all other scan chains behind the other scan interfaces.

The retargeting software assigns the broadcast values during each scan load. The number of bits always matches the number of bits in the currently active scan path in the target Scan Interface.

Return Values

None.

Examples

The following example disables capture on scan interface client1:

```
iOverrideScanInterface client1 -capture off
```

The following example disables update on two scan interfaces:

```
iOverrideScanInterface c2 c3 -update off
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iPDLLevel

Context: all contexts

Mode: all modes

Identifies the level and version of PDL commands.

Usage

iPDLLevel ***level*** [-version *version*]

Description

Identifies the level and version of PDL commands.

The iPDLLevel command is mandated by the IEEE 1687-2014 standard to be the first command in the PDL file, and it specifies that all the subsequent commands are PDL commands of the specified level and version.

Note

Tessent Shell does not verify the compliance of subsequent commands with the specified PDL level. Tessent Shell accepts Level-0 PDL as well as Level-1 PDL, independent of the specification within the iPDLLevel command.

Arguments

- ***level***

  A required value that specifies the PDL level. The permitted values are 0 and 1.

- -version *version*

  An optional switch and string pair that specifies the PDL version. Currently, the only permitted value is STD_1687_2014.

Return Values

None.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iPrefix

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Sets the iPrefix path that is used to compute the effective_icl_instance_path for the iCall command and other PDL commands.

Usage

iPrefix [i*cl_instance_path*]

Description

Sets the iPrefix path that is used to compute the effective_icl_instance_path for the iCall command and other PDL commands.

The effective_icl_instance_path is described in detail in the iCall command description.

This command is not affected by the previous iPrefix command. If the icl_instance_path argument is omitted, the iPrefix path value is set to empty. If the path is specified and the iPrefix command is inside an iProc, it must be a valid instance path relative to the current iProc instance path. If the path is specified and the iPrefix command is not inside an iProc, it must be a valid instance path relative to the current design.

Arguments

- *icl_instance_path*

  An optional string that defines the new value for iPrefix path used to compute the effective_icl_instance_path for the iCall command and other PDL commands such as the iRead, iWrite, and iClock commands.

Return Values

None

Examples

In the following example, the iPrefix path value is first set to block1_i1.inst1_i1. It is then set to block1_i2 with the second invocation of the iPrefix command. Note how the second iPrefix command completely replaces the value of the iPrefix path and is not influenced by the previous value of iPrefix path. Only the iCall, iClock, iRead, iWrite, iRunLoop, and iClockOverride commands are affected by the iPrefix command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
iPrefix block1_i1.inst1_i1
iWrite A 0b0
iPrefix block1_i2
iWrite inst1_i1.A 0b0
```

Related Topics

iCall

iClock

iClockOverride

iRead

iWrite

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iProc

Context: all contexts

Mode: setup, analysis (patterns contexts only), insertion

Prerequisites: The iProcsForModule has been specified.

Specifies a PDL procedure that can run later when referenced by the iCall command.

Usage

iProc ***name variables body***

Description

Specifies a PDL procedure that can run later when referenced by the iCall command.

Analogous to the Tcl command “proc”, the specification consists of the procedure name, the procedure variables, and the procedure body.

The specified procedure is registered against a specific ICL module that must have been previously loaded by read_icl and specified by iProcsForModule. Because the iProc is registered against a certain module, several iProcs with the same name may exist, each associated to a different ICL module. If an iProc is specified with a name that already exists for the current iProcsForModule, the new specification replaces the existing one. If an ICL module is deleted, the related iProcs are deleted too.

When the procedure is called using the iCall command, the variables in the argument list are substituted with the values specified in the iCall command. The tool runs the commands inside the iProc body in the scope of the instance against which the iProc is called.

Arguments

- ***name***

  A required name for the iProc, which must be a simple scalar identifier. A simple scalar identifier starts with a letter and contains only letters, decimal digits, and underscores.

- ***variables***

  A required Tcl list of variable name or variable-value pairs enclosed in braces ({}). The list of variables can contain either simple identifiers or identifier-value pairs, where the value specifies the default value of the variable. All simple identifiers must be in the list before the identifier-value pairs.

- ***body***

  A required list of commands to be run when the procedure is called by means of the iCall command.

Return Values

None

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example shows an iProc called run_testa that is registered against the myInstrument1 ICL module. The iProc has two arguments: cnt and mode. The mode argument is optional because it has a specified default value of “red”. The body of the iProc contains standard Tcl syntax and PDL commands. The cnt argument is specified before the mode argument because it has no specified default value.

```
iProcsForModule myInstrument1

iProc run_testa {cnt {mode red}} {
  set cycles(red) 50
  set cycles(blue) 90
  set cycles(green) 70
  iRead go Fail
  iRead done No
  iApply
  iWrite mode $mode
  if {$mode == "red"} {
    iWrite p1 0b0
    iWrite p1(2) 0b1
  }
  iApply
  iWrite enable Yes
  iApply
  iRunLoop $cycles($mode) -tck
  iRead done Yes
  iRead go Pass
  iRead count $cnt
  iApply
}
```

Related Topics

delete_iprocs

get_iproc_argument_list

get_iproc_list

iCall

iProcsForModule

iTopProc

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iProcsForModule

Context: all contexts

Mode: setup, analysis (patterns contexts only), insertion

Prerequisites: The corresponding ICL module already loaded.

Specifies the ICL module for which subsequent iProc commands refer to.

Usage

iProcsForModule ***icl_module_name*** [-iProcNameSpace *name_space*]

Description

Specifies the ICL module for which subsequent iProc commands refer to.

An iProc is a PDL procedure written for a specific ICL module.

All references to ICL objects within the iProc refer to the instance path relative to the module it is written for. Once you have issued the iProcsForModule command, all iProcs defined afterward are registered against the specified ICL module. You cannot define any iProc until the iProcsForModule is set.

The iProcsForModule is specified once for each ICL module that you have iProcs to register against it. The iProcsForModule can only be specified for ICL modules that already have been loaded using the read_icl command.

To be sure that you load iProcs against the correct module, you can store all iProcs associated to a given ICL module inside a single file and add the proper iProcsForModule at the top of the file. With this organization, you can load the iProcs for any module as soon as the specified ICL module has been read in using the read_icl command.

Arguments

- ***icl_module_name***

  A required name of a valid ICL module that has already been loaded into memory using the read_icl command.

- -iProcNameSpace *name_space*

  An optional switch and string that creates a sub name space within each name space created from the ICL module name. When *name_space* is not specified, it defaults to the value set by the iUseProcNameSpace command.

Return Values

None

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

In the following example, the file myCore.icl is read in; the file contains the entire ICL description of myCore. A child module of myCore is myInstrument1. The file *myInstrument1.iprocs* contains an iProcsForModule command that specifies that myProc defined afterward is meant to be registered against this module.

Sourcing the file myInstrument1.iprocs loads the iProcs defined in it. One of the instances of myInstrument1 within myCore is u2.u3. The iCall u2.u3.myProc command runs the iProc called myProc defined for module myInstrument1 against the u2.u3 ICL instance of that module. The solver retargets the PDL found within that iProc to the boundary of myCore.

read_icl myCore.icl
source myInstrument1.iprocs
set_current_design myCore
set_system_mode analysis
open_pattern_set P1
iCall u2.u3.myProc
close_pattern_set

myInstrument1.iprocs file contains

```
iProcsForModule myInstrument1
iProc myProc {} {
   iRead u1.R 0b0
   iApply
}
```

Related Topics

delete_iprocs

get_iproc_argument_list

get_iproc_list

iCall

iProc

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iPulseClock

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

In the “patterns -ijtag” context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the “dft”, “patterns -scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iTopProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Starts or restarts a synchronous clock that previously has been forced to its off-state by means of the iForcePort command.

Usage

iPulseClock ***object_spec***

Description

Inside an IJTAG pattern set, iPulseClock can be used to start or restart a synchronous clock that previously has been forced to its off-state by means of the iForcePort command.

An iPulseClock command is only permitted in an iTopProc and not in a normal iProc as it is not retargetable upward. It is best to keep the iProcs that only interact with the ICL ports separated from the iTopProcs using the iPulseClock command so that you maintain the ability to retarget the bulk of your PDL.

The effect of the iPulseClock command is applied on the next iApply command or the next iRunLoop command.

Arguments

- ***object_spec***

  A collection containing the top level clock/clocks that is/are going to be restarted. If you are using this command on a port that is not described in the top ICL module (a non-IJTAG port), you must have read the interface view of the top level Verilog or VHDL module to get the list of ports available to pulse. Furthermore, the port must have been specified as a clock port by means of the add_clocks command. Use “read_verilog -interface_only” or “read_vhdl -interface_only” to read in the port list prior to issuing the set_current_design command as shown here:

```
set_context patterns -ijtag -no_rtl
read_verilog top.v -interface_only
read_icl top.icl
set_current_design top
add_clocks 0 clkRZ
add_clocks 1 clkRO
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Asynchronous clocks and internal clocks cannot be used as an argument of the iPulseClock command. Constrained ports or ports that have not been subject to an add_clocks command also cannot be used.

Examples

The following example forces two clocks to their off-states, waits for 1000 TCK cycles and writes a value to an ICL port. Then it starts the clocks pulsing, waits for 2000 TCK cycles and writes another value to the ICL port.

```
set_context patterns -ijtag
read_icl top.icl
read_verilog top.v
set_current_design top
add_clocks 0 clk1
add_clocks 1 clk2
set_system_mode analysis

open_pattern_set patset1 -tck_ratio 4
  iForcePort {clk1 clk2} 0b01
  iRunLoop 1000
  iWrite din 0b1
  iApply
  iPulseClock {clk1 clk2}
  iRunLoop 2000
  iWrite din 0b0
  iApply
close_pattern_set
```

Related Topics

add_clocks

iApply

iForcePort

iRunLoop

iTopProc

open_pattern_set

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iRead

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Adds a read operation to the command queue to be solved by the next iApply command.

Usage

iRead [***icl_instance_path***.]***readable_icl_object*** [*value*]

Description

Adds a read operation to the command queue to be solved by the next iApply command.

You can specify an optional expected value to which the read value is compared to when observed. The iApply command description describes how the iRead commands are interpreted by the iApply command.

For a description of how to associate symbolic variables with iRead commands, see "PDL Retargeting With Symbolic Variables" in the *Tessent IJTAG User's Manual*.

Arguments

- [***icl_instance_path***.]***readable_icl_object***

  Specifies the ICL objects that are to be read. You can specify to add read operations for the following ICL objects: DataRegisters, ScanRegisters, DataOutPorts, DataInPorts, or an alias of any of these objects. The ICL objects must be found at the computed effective_icl_instance_path. The effective_icl_instance_path is described in detail in the iCall command description.

- *value*

  An optional constant number represented with Tcl binary, hexadecimal, or integer syntax or with a string corresponding to an enumeration symbol associated to the readable_icl_object in the ICL description.

Return Values

None

Examples

The following example requests the readable_icl_objects done and go to be read and compared against the value represented by the enumeration symbol called Yes and Pass defined in ICL.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The ICL object count is scheduled to be read but not compared to any expected value. The three requested read operations are solved when the iApply command runs.

```
iRead done Yes
iRead go Pass
iRead count
iRead status 0b1
iApply
```

Related Topics

iApply

iCall

iPrefix

iWrite

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iRelease

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Releases a resource previously taken by means of iTake.

Usage

iRelease [*icl_resource*]

Description

iRelease enables the release of the ownership of a resource that has been reserved with iTake. After iRelease of the resource, the resource may be freely altered upon merging with other iProcs, and the retargeting software can also make use of the resource again, for example, to reconfigure the scan network or to justify observation paths.

Arguments

- *icl_resource*

  An optional value that specifies the name of a register, a port or an instance. If an instance is specified, then all registers, ports and instances in *icl_resource* are released. If you issue the command without this argument, the effect is the same as if an empty string was passed. In this case, the effective instance is the one that is determined by the ICL instance path of the currently processed iProc and by the iPrefix command. All registers and ports of this instance are released.

Return Values

None.

Examples

```
iRelease RegA
iRelease
```

Related Topics

iCall

iMerge

iTake

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iReset

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Adds a sequence of actions to the current pattern set that is required to set the ICL network into the reset state.

Usage

iReset [-sync]

Description

Adds a sequence of actions to the current pattern set that is required to set the ICL network into the reset state.

The exact sequence performed depends on the presence of specific ports in the current design. Each ResetPort is pulsed high and each TRSTPort is pulsed low. If there are TMSPorts without a corresponding TRSTPort, TCK is pulsed five times with TMS held high followed by an extra pulse of TCK with TMS low.

After an iReset command, the entire ICL is in its reset state and the TAP finite state machines (when present) are in RunTestIdle state.

You should only use the iReset command once at the beginning of a pattern set. It is illegal to include an iReset inside an instrument PDL as its effect is global and destroys any ongoing action from all instruments. The iReset command is only permitted after open_pattern_set, in an iProc associated with the current design, or in test_setup/test_end procedures (the latter only permitted in contexts other than "patterns -ijtag"). Placing it inside an instrument-level iProc prevents this iProc from being iCalled from a higher level module.

If there is no explicit iReset at the beginning of an open pattern set, an iReset is implied unless the -no_initial_ireset option is used with the open_pattern_set command. Refer to the open_pattern_set command description to understand when and how the -no_initial_ireset option should be used.

Arguments

- -sync

  An optional switch that forces a synchronous reset even if there are TRSTPorts in the current design. TCK is pulsed five times with TMS held high followed by an extra pulse of TCK with TMS low. If there is no TMS port, the -sync switch is ignored.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Return Values

None

Examples

The following examples set the ICL network into the reset state:

```
iReset
iReset -sync
```

Related Topics

iSuspendPdlRetargeting

iResumePdlRetargeting

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iResumePdlRetargeting

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Returns the IJTAG retargeter to the normal mode of operation after a previous run of the iSuspendPdlRetargeting command.

Usage

iResumePdlRetargeting

Description

The iResumePdlRetargeting command returns the IJTAG retargeter to the normal mode of operation after a previous run of the iSuspendPdlRetargeting command. You are responsible for guaranteeing the correct behavior of the IJTAG retargeter after the suspension.

If the state of the IJTAG network changed to a known and well-defined new state, use the iState command to update the internal IJTAG simulator state. If the state of the IJTAG network was affected but the network state is unknown or not well-defined, you must use the iReset command to reset the IJTAG network. The iReset command returns the IJTAG retargeter to the normal mode of operation.

It is possible to drive the IJTAG network during the PDL retargeting suspension. However, in this case, it is your responsibility to ensure that all control signals are driven correctly. It is also your responsibility to return the entire IJTAG network to the same state that it was in before the PDL retargeting suspension. Siemens EDA recommends that you do not perform any IJTAG operations during the suspension.

You can also change the state of the TAP's finite state machine (FSM) during the suspension. In this case, you should also restore the TAP’s FSM to the idle state before resuming the PDL retargeter.

The iResumePdlRetargeting command automatically releases iForcePort statements that the tool does not allow during normal retargeting operations.

Arguments

None

Examples

Example 1

The following example suspends IJTAG retargeting to perform custom non-IJTAG chip initialization operations. The script then runs the iResumePdlRetargeting command to return to normal operation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
open_pattern_set initialization
  iCall initChip

  iSuspendPdlRetargeting

  iCall NonIjtagSetupProcedure
  iNote "Wait for security code processing"
  iRunLoop 64 -sck clock1
  iComparePort securityCircuitActive 0b1
  iApply

  iResumePdlRetargeting

  iCall chip.test_setup
close_pattern_set
```

Example 2

The following example suspends IJTAG retargeting to perform custom operations on the IJTAG client interface. The script then runs the iResumePdlRetargeting command to return to normal operation.

```
open_pattern_set initialization
  iCall initCore

  iSuspendPdlRetargeting
  iForcePort securityGPIO 0b11110000
  iApply

  iForcePort ijtag_se 0b1
  iForcePort ijtag_si 0b0
  iApply

  iRunLoop 8 -tck
  iForcePort ijtag_se 0b0
  iForcePort ijtag_ue 0b1
  iApply

  iRunLoop 1 -tck
  iForcePort ijtag_ue 0b0
  iComparePort securityCircuitActive 0b1
  iApply

  iState core1_inst.securityReg 0b11111111
  iResumePdlRetargeting
  iCall core1_inst.setup
close_pattern_set
```

Related Topics

iSuspendPdlRetargeting

iReset

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iRunLoop

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Creates a vector loop of a given duration.

Usage

iRunLoop {**count** [-tck | -sck [*icl_instance_path.*]*icl_clock_port*]} | {-time *tvalue*}

Description

Creates a vector loop of a given duration.

In the generated pattern, the loop is converted into a loop of tester periods.

All clocks defined with the add_clocks commands toggle at their specified period in a free running fashion during the entire pattern. When using the -tck option, the iRunLoop count is expressed in terms of TCK periods and TCK is toggling during the loop. When the -sck [icl_instance_path.]icl_clock_port option is used, TCK is not toggling.

The specified [icl_instance_path.]icl_clock_port must have been specified in a prior iClock command such that its controlling source was verified and its effective cumulative multiplication factor computed. The size of the loop in the pattern depends on the computed effective multiplication factor as well as the ratio between the clock source period and the tester period. Table 5-27 outlines the resulting loop size in the generated pattern based on the various options.

Table 5-27. RunLoop Scaling Algorithm

Condition	Results
open_pattern_set -tck_ratio = 1 iRunLoop X -tck	TCK is defined with a pulsing waveform with the falling edge preceding the rising edge. The loop count is X and is over a single vector where TCK is set to pulse
open_pattern_set -tck_ratio = 2 iRunLoop X -tck	TCK is defined with a data waveform. The loop count is X and is over 2 vectors where TCK is low for the first vector and high for the second one

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 5-27. RunLoop Scaling Algorithm (cont.)

Condition	Results
open_pattern_set -tck_ratio = 4, 8, 16,... iRunLoop X -tck	TCK is defined with a data waveform. The loop count is X and is over <tck_ratio> vectors where TCK is low for the first and last quarter of vectors and high for the middle ones.
iRunLoop X -sck <clk> where <clk> has a source that was defined as an asynchronous clock using the "add_clocks -period" command	TCK is inactive during the loop. The loop count is Y and is over a single vector. Y = X / <effective multiplication factor> * (<async clock period>/<tester period>)
iRunLoop X -sck <clk> where <clk> has a source that was defined as a synchronous clock and the waveform defined for this clock has 2 edges	TCK is inactive during the loop. The loop count is Y and is over a single vector. Y = X / <effective multiplication factor>
iRunLoop X -sck <clk> where <clk> has a source that was defined as a synchronous clock and the waveform defined for this clock has 2*n edges	TCK is inactive during the loop. The loop count is Y and is over a single vector. Y = X / <effective multiplication factor>*1/n

Arguments

- **count**

  Required integer that specifies the number of clock cycles to be seen on TCK or on [icl_instance_path.]icl_clock_port. It is not required when the -time switch is used.

- -tck

  Optional switch that specifies that count refers to the test clock TCK.

- -sck [*icl_instance_path*.]*icl_clock_port*

  Optional switch and value pair that specifies for which clock pin the count is for. The value [*icl_instance_path*.]*icl_clock_port* specifies an internal or top-level clock port defined inside the ICL description. The specified clock port must be a port declared as a ClockPort or ToClockPort inside the ICL and it must be found at the computed effective_icl_instance_path. The effective_icl_instance_path is described in detail in the iCall command description. An iClock command on the [icl_instance_path.]icl_clock_port must have been issued prior to issuing the iRunLoop command.

- -time *tvalue*

  Optional switch and value pair that specifies a time delay *tvalue* instead of a cycle count. When TCK is OFF, iRunLoop -time is converted to a single vector loop, with a loop count

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

of *tvalue*/tester_period rounded up to the next integer. If TCK is asynchronous free-running, TCK is ON and the loop count must be a multiple of the TCK ratio.

Return Values

None

Examples

Example 1

The following example specifies a run loop of 10000 tck period. The tck clock is toggling during the loop when the iRunLoop command is using the -tck option.

```
iRunLoop 10000 -tck
```

Example 2

The following example specifies a run loop of a 1000 system clock cycles on the clk pin of the u1 instance. The iClock command checks that the clock port is controlled by a valid source and computes the cumulative multiplication factor between the primary input and the clock port. The run loop is automatically adjusted such that the u1.clk port receives a minimum of 1000 cycles at the u1 instance. The TCK port may or may not be toggling.

```
iClock u1.clk
iRunLoop 1000 -sck u1.clk
```

Example 3

The following examples specify run loops that must last for at least the specified time.

```
iRunLoop -time 2000ns
iRunLoop -time 4.56E+2ns
iRunLoop -time 5.67E-1us
```

Related Topics

add_clocks

ALL_MAX_LOOP

delete_clocks

iClock

iClockOverride

report_clocks

report_icl_modules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_collection

Context: unspecified, all contexts

Mode: all modes

Indicates whether string is a collection handle.

Usage

is_collection ***string*** [-with_object *obj_type*] [-single_type_objects]

Description

Indicates whether string is a collection handle.

This command returns true if the provided string is a collection handle. You use this command inside a Tcl proc to check if the passed in argument is a collection of objects versus names of objects. See the example below for an example usage.

You can also use this command to check if the collection contains only one type of objects or objects of a specific type.

Arguments

- ***string***

  A required string that specifies the collection whose existence is to be verified. This command returns true if the collection exists.

- -with_object *obj_type*

  An optional switch and string that specifies whether the collection specified by string contains at least one object of the type specified by *obj_type*. This command returns true if at least one object of the type specified by *obj_type* exists.

- -single_type_objects

  An optional switch that specifies whether the collection specified by string contains objects of all the same type. This command return true only if all objects are of the same type.

Examples

The following example uses the is_collection command to verify that the net argument is already a collection of nets and, if not, the get_nets command is used to create the collection of nets assuming <net> is instead a Tcl list of net names. If the passed-in argument is a collection, it is checked to verify that it only contain net objects. With this code, the proc myProc can be invoked with either a Tcl list of names of nets, or a collection of nets and the proc automatically adapts.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
proc myProc {net} {
   if {[is_collection $net]} {
      if {![is_collection $net -with_object net -single_type_objects]} {
         return -code error "<net> contains other object types than net"
      }
   } else {
      set net [get_nets $net]
   }
   #$net is definitely a collection of nets
   ...
}
```

Related Topics

add_to_collection

append_to_collection

compare_collections

filter_collection

foreach_in_collection

index_collection

range_collection

remove_from_collection

sizeof_collection

sort_collection

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_valid_object_name

Context: unspecified, all contexts

Mode: all modes

Indicates whether a string is a valid object name.

Usage

is_valid_object_name ***string*** [-language {verilog | systemverilog | vhdl}]

Description

This command returns a value of "1" if the provided string is a valid Verilog, SystemVerilog, or VHDL object name. Otherwise, a value of "0" is returned.

Arguments

- ***string***

  A required argument that specifies the string to test for validity.

- -language {verilog | systemverilog | vhdl}

  An optional switch and string that specifies the language to use for the validity check. The default is Verilog.

Examples

The following examples check whether the given strings are valid object names in the specified languages.

```
is_valid_object_name inst1 // verilog (default)
1
is_valid_object_name -language verilog inst1[1]
1
is_valid_object_name -language systemverilog inst1[2]
1
is_valid_object_name -language vhdl inst1[5]
1
is_valid_object_name -language vhdl inst1[5]#
0
is_valid_object_name -language verilog out
1
is_valid_object_name -language vhdl out // reserved word
0
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iScan

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Adds a scan operation to the command queue for the next iApply command to solve.

Usage

iScan [-ir] ***scan_interface*** [-chain *chain*] ***length***
[-si *scan_in_data*] [-so *scan_out_data*]
[-skip_model_state_simulation]

Description

Refer to ICL and PDL Limitations in the *Tessent IJTAG User's Manual* for the limitations of this command.

The iScan command instructs the tool to add a scan load to the list of commands for the next iApply to process. This scan load occurs at the end of the next iApply. In other words, this scan load occurs after all the actions required to retarget other commands (such as iRead and iWrite) that are in the queue for the next iApply.

When you do not use the -skip_model_state_simulation switch, you must configure a scan path in the IJTAG network. A properly configured scan path must travel from the ScanInPort to the ScanOutPort of the specified ScanInterface. This scan path must exist after the iApply has retargeted any other commands. In other words, you can add the commands for the scan path configuration and the iScan command that uses the resulting scan path into the same iApply command.

When there are no other commands queued for the current iApply command, such as iRead or iWrite, the scan path must already exist and be properly configured at the beginning of the iApply for the iScan command to run.

An iApply command can only target one ScanInterface with its queue of iScan commands. An iApply command can schedule multiple iScan commands for only one ScanInterface, in which case, each scan chain specification must be different.

The specified ScanInterface must be a client or client-TAP ScanInterface of the ICL module that is associated with the current design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- -ir

  An optional switch that specifies the performance of an instruction register (IR) scan load instead of a data register (DR) scan load. Use this switch only when the specified ScanInterface is a client-TAP ScanInterface.

- ***scan_interface***

  A required string that contains the name of the ScanInterface for the scan load.

  Use the name “unnamed” when the tool creates the ScanInterface during ICL elaboration. This is the case when the following conditions are both true:

  - There is no explicitly specified ScanInterface in the ICL associated with the current design.
  - There is a combination of ports that unambiguously form a valid client or client-TAP ScanInterface.

  For this case, the tool creates the ScanInterface internally as part of the ICL data model even though there is no explicit ScanInterface specification in the ICL.

- -chain *chain*

  An optional switch and value pair that specifies the name of the chain for the scan load. Although it is optional for most circumstances, it is required for ScanInterfaces with Chain specifications.

  In order to specify the scan load data for all chains of one ScanInterface, the tool requires one iScan command per chain. When the iScan command for one of the chains is missing, the tool either loads values from the previous scan load through this chain or loads default values when no scan load has previously happened. For ScanInterfaces that do not have “Chain” specifications in the ICL, you can either omit the -chain switch or use the chain “default” name.

- ***length***

  A required integer that specifies the number of scan cycles of the scan load.

  Because PDL numbers do not contain size information, the number of scan cycles cannot be derived from the -si or -so specifications. It must be specified separately.

  If ***length*** is greater than the length of the currently configured scan path in the ICL network, overshifting happens, and a warning appears. If ***length*** is less than the length of the currently configured scan path in the ICL network (undershifting), and an error appears. The tool does not support partial scan loads.

- -si *scan_in_data*

  An optional switch and value pair that specifies the data to load into the chain.

  You must provide the data as a PDL number in either binary, decimal, or hexadecimal format. It must not contain unknown X values. It may optionally use spaces and newlines

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

within the number for readability, in which case it must be enclosed in braces or double quotes to prevent Tcl parsing as multiple arguments.

The number of specified bits must not be larger than the specified ***length***. If the number of specified bits is smaller than the specified ***length***, the *scan_in_data* is left-padded with zeros until it matches the specified ***length***.

The leftmost bit (most significant bit) of the padded *scan_in_data* corresponds to the register bit that is closest to the ScanInPort of the ScanInterface. The rightmost bit (least significant bit) corresponds to the register bit that is closest to the ScanOutPort.

- -so *scan_out_data*

  An optional switch and value pair that specifies the data to observe and compare at the output of the chain.

  You must provide the data as a PDL number in either binary, decimal, or hexadecimal format. It can contain unknown X values representing “do not compare”. It may optionally use spaces and newlines within the number for readability, in which case it must be enclosed in braces or double quotes to prevent Tcl parsing as multiple arguments.

  The number of specified bits must not be larger than the specified ***length***. If the number of specified bits is smaller than the specified ***length***, the *scan_out_data* is left-padded with X values until it matches the specified ***length***.

  The leftmost bit (most significant bit) of the padded *scan_out_data* corresponds to the register bit that is closest to the ScanInPort of the ScanInterface. The rightmost bit (least significant bit) corresponds to the register bit that is closest to the ScanOutPort.

- -skip_model_state_simulation

  An optional switch that instructs the tool to alter its normal operation in the following ways:

  - Ignore the network configuration.
  - Do not check for overshifting and undershifting.
  - Do not apply the scan load values to the state of the ICL network model.
  - Do not simulate the effect of the scan load on the ICL network.
  - Transform the -si data directly into stimulus values for the ScanInPort of the ScanInterface.
  - Transform the -so data directly into response values for the ScanOutPort of the ScanInterface.

  The -skip_model_state_simulation switch is useful when there are incomplete ICL descriptions or in situations where the state of the internal model is irrelevant to reduce the run time of IJTAG retargeting. An example is when the development of the related ICL is not complete. Another example is when you do not want to load all the ICL for performance reasons.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Caution

IJTAG retargeting after an “iScan -skip_model_state_simulation” command can result in undefined behavior.

After you use the -skip_model_state_simulation switch, the state of the internal model (a simulation model of the IJTAG network created and used by the retargeter) will probably not match the state of an actual IJTAG network in silicon after running patterns. In other words, the virtual simulation will probably not match the real design in silicon.

Therefore, all IJTAG retargeting activities that use iRead and iWrite commands after an iScan command employing the -skip_model_state_simulation switch can result in undefined behavior. You are responsible for ensuring that the state of the internal model for the IJTAG retargeter matches the state that is established by the generated IJTAG patterns. One way to achieve this is by resetting the network.

Examples

Example 1

The design for this example has a TAP controller in an ICL instance named tap_inst. The ICL ScanInterface that provides access to the IJTAG network is named top_intf.

For the instruction 0b110011, the TAP controller configures its DR multiplexer to arrange TDRs with a combined length of 47 on one scan path between TDI and TDO.

The example uses the iScan command to load new values and to observe the capture values of those TDRs. For this example, the 32 bits that are closest to TDO are important. Therefore, the code specifies hexadecimal numbers with eight digits only. The tool pads the remaining 15 bits with 0s (stimuli) and Xs (responses).

The example closes by using the iApply command to trigger pattern retargeting for the programming of the instruction register and for the scan load through the TDR chain.

```
iWrite tap_inst.IR 0b110011
iScan top_intf 47 -si 0xDCBAABCD -so 0x00117X32
iApply
```

Example 2

Alternatively, the iScan command can program the instruction register of the previous example. For this case, this example uses two iApply commands because an iApply command can process only one iScan-related scan load. The iScan command must use the -ir switch because the first scan load is an instruction register scan load.

```
iScan -ir top_intf 6 -si 0b110011
iApply
iScan top_intf 47 -si 0xDCBAABCD -so 0x00117X32
iApply
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

isolate_icl_node

Context: patterns -silicon_insight -ijtag

Mode: setup

Configures the network to setup the shortest path from TDI to TDO that includes the specified element.

Usage

isolate_icl_node ***node_name***

Description

The tool configures the shortest path (lowest number of flops) from TDI to TDO.

Use this command when manually debugging IJTAG networks as described in "Diagnosing and Debugging IJTAG Networks" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

Arguments

- ***node_name***

  A required string that specifies the SIB, scan mux, or scan register to isolate.

Examples

```
isolate_icl_node sib_I4@0                    // SIB
isolate_icl_node block3_I1.M1@3              // scan mux
isolate_icl_node block1_I1.tdr1.R[7:0]       // scan register
```

Related Topics

analyze_icl_suspect

assert_icl_sib

create_icl_flush_pattern

create_icl_setup_pattern

deassert_icl_sib

get_icl_network

get_icl_scan_mux

report_icl_network

set_icl_network

set_icl_scan_mux

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iState

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc or iTopProc that is called during the processing of a test procedure like test_setup or test_end. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Manipulates the state of the internal model for the IJTAG retargeter directly, without creating any additional vectors in the currently opened pattern set.

Usage

iState ***icl_writable_object value*** [-LastWrittenValue]

Description

Refer to "ICL and PDL Limitations" in the *Tessent IJTAG User's Manual* for the limitations of this command.

Use the iState command to manipulate the state of the internal model for the IJTAG retargeter directly. The tool does not create any additional vectors by using the iState command.

Caution

The iState command presents a high risk of putting the IJTAG network into an inconsistent state. It must be used with great care.

There is no possibility of checking if the specified values exist in the design. The IJTAG retargeter does not perform any plausibility checks. It relies on the correctness of the user specifications. It does not perform any actions to establish the purported network state. It does not generate any patterns.

Use this command in situations where your process uses non-IJTAG mechanisms to create patterns that affect the state of the internal model for the IJTAG network. In such scenarios, you must inform the IJTAG retargeter about the effect of those non-IJTAG patterns on the network. You must do this to ensure that the internal model's state can match a network after applying those non-IJTAG patterns.

One example relevant to Tessent Shell is using the native procedure file commands, cycle, force, and pulse, in a test_setup procedure between test_setup iCall commands. For more information on those commands, see "Procedure Definition" in the *Tessent Shell User's Manual*. When those commands change the IJTAG network state, the retargeter must also know about those changes before it can correctly process the next iCall.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***icl_writable_object***

  A required string that contains one of the following:

  - The name of an ICL input port of the ICL module that is associated with the current design. Its port function must be one of DataInPort, SelectPort, AddressPort, WriteEnPort, or ReadEnPort.
  - The name of an ICL ScanRegister, optionally including the instance path, and optionally including an index or range specification.
  - The name of an ICL DataRegister, optionally including the instance path, and optionally including an index or range specification.

- ***value***

  A required PDL value to modify the state of the internal model for the IJTAG retargeter. The value can contain Xs.

  The width of the value must be less than or equal to the number of bits of the specified ***icl_writable_object***. When the width of the value is less than the number of bits of the specified ***icl_writable_object***, the value is left-padded either with 0s (if the leftmost specified bit is 0 or 1) or with Xs (if the leftmost specified bit is X).

  Like all PDL commands dealing with ICL targets and PDL values, the leftmost bit of the padded value refers to the leftmost bit of the ***icl_writable_object***. The rightmost bit of the padded value refers to the rightmost bit of the ***icl_writable_object***.

  Additionally, the value can be an enumeration symbol of an Enum specification associated with the icl_writable_object. This is a Tessent Shell-specific extension of the IEEE 1687 standard.

- -LastWrittenValue

  This switch has no effect even though the IEEE 1687 standard describes it. Use it to emphasize that the command refers to the assumed written state instead of the most recent read or miscompare value. The functionality of the iState command is exactly the same with or without this switch.

Examples

For this example, a process unknown to the IJTAG retargeter changes three parts of the design:

- The content of the scan register named inst1.scanreg.
- The content of the data register named inst1.inst2.datareg.
- The stimulus values of the primary ports data_in[3:0].

Additionally, this change becomes effective after applying the patterns resulting from the retargeting of the iCall proc1 command and before applying the patterns resulting from the retargeting of the iCall proc2 command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

It is your responsibility to ensure that those changes actually happen in the design when the patterns are applied on an ATE or during testbench simulation.

The example shows the iState commands informing the IJTAG retargeter about the changes.

```
iCall proc1

iState inst1.scanreg[15:0] 0x7F3C
iState inst1.inst2.datareg[7:0] 0b11100111
iState data_in[3:0] 0b0110 -LastWrittenValue

iCall proc2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iSuspendPdlRetargeting

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Enables the ability to force ScanInterface ports to a particular value using iForcePort and iPulseClock commands. Enables the creation of delays using an iRunLoop command. Facilitates applying stimulus to the ScanInterface and other ports of the current design.

Usage

iSuspendPdlRetargeting

Description

The iSuspendPdlRetargeting command enables the ability to force ScanInterface ports to a particular value using iForcePort and iPulseClock commands. The iSuspendPdlRetargeting command enables the creation of delays using an iRunLoop command. It facilitates the ability to apply stimulus data to all ports of the current design, including ScanInterface ports.

The iSuspendPdlRetargeting command suspends the use of all commands that perform PDL retargeting. It suspends the use of all commands that require a consistent model state of the IJTAG network. It suspends the use of iCalls for all modules except the current design until an iReset or an iResumePdlRetargeting command.

Return the IJTAG retargeter to the normal mode of operation by using an iReset command or an iResumePdlRetargeting command. Siemens EDA recommends that you use the iReset command whenever possible.

The iSuspendPdlRetargeting command works during test procedure iCall processing and in the patterns -ijtag context. Generating the correct vectors for the primary ports without PDL retargeting is the main goal during this phase of pattern generation or test procedure processing.

The iSuspendPdlRetargeting command suspends the use of the following list of commands:

- create_icl_verification_patterns
- create_ssn_continuity_patterns
- iCall (unless it is associated with the current design)
- iClock *clk* (unless *clk* is an input port of the current design)
- iClockOverride
- iMerge
- import_patterns_from_svf
- iOverrideScanInterface
- iRead

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- iRelease
- iRunLoop -sck *clk* (unless *clk* is an input port of the current design)
- iScan
- iState
- iTake
- iWrite

Use an iSuspendPdlRetargeting command outside of an iProc, or inside an iTopProc. Use it between the open_pattern_set and close_pattern_set commands.

You cannot use the iSuspendPdlRetargeting command in the body of an iProc. For this case, the tool returns an error while processing the iProc. You can expect that an iProc is retargetable without knowing the content of the iProc body. However, an iProc becomes non-retargetable if it contains an iSuspendPdlRetargeting command.

Do not use the iSuspendPdlRetargeting command at any level of the iMerge processing. Neither directly between iMerge -begin and iMerge -end, nor in an iCall used by an iMerge block.

The iSuspendPdlRetargeting command cannot be used in InSystemTest.

An open_pattern_set command with the -no_initial_ireset switch keeps the “suspended retargeting” state of the previous pattern set. The other variants of the open_pattern_set command (including the default that has an implied iReset) reset the retargeting engine and instruct the retargeter to assume the “not suspended” state.

At the end of the test_setup procedure, the network requires reset activity when the test_setup_network_end_state (set by the set_ijtag_retargeting_options command) is not in the keep state. Retargeting must not suspend when the tool is establishing the network end state. Otherwise, an error will occur during the system mode transition from setup to analysis mode.

When SSN is active, do not suspend retargeting when the tool encounters the end of the test_setup, ssn_setup, ssn_end, or test_end test procedures. Otherwise, an error will occur during the system mode transition from setup to analysis mode. The IJTAG network must be in a defined state for any SSN-related operations.

At the end of an IJTAG pattern set, the network requires reset activity when the network_end_state (set by the close_pattern_set command or the equivalent PatternsSpecification/Patterns/TestStep/AdvancedOptions/network_end_state) is not in the keep state. Retargeting must not suspend when the tool tries to establish the network end state. Otherwise, an error will occur when running the close_pattern_set or process_patterns_specification commands.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

None

Return Values

None

Examples

The example test_setup procedure below consists of four parts:

1. Ordinary PDL retargeting.
2. Suspension of PDL retargeting to apply iForcePort commands.
3. Perform an iReset to return to ordinary mode of operation. Previous initialization steps are assumed to be effective due to logic that is not included in the ICL model.
4. Ordinary PDL retargeting again.

```
iProcsForModule chip

iTopProc chipsetup args {

   # 1. PDL retargeting
   iCall inst1.clamp_enable
   iCall inst2.clamp_enable
   iCall inst3.clamp_enable

   # 2. Suspend PDL retargeting
   iSuspendPdlRetargeting
   iForcePort TM 0b1
   iApply
   iForcePort TM 0b0
   iApply
   iForcePort TM 0b1
   iApply
   iForcePort TM 0b0
   iRunLoop -time 50ms

   # 3. Reset
   iReset

   # 4. PDL retargeting
   iCall inst1.setup_controller
   iCall inst2.program_instrument
   iCall inst3.verify_measurement
}
```

Call the procedure from a dofile using the following command:

```
set_test_setup_icall chipsetup
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

iReset

iResumePdlRetargeting

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iTake

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Takes ownership of ICL resources to prevent the retargeting software from altering their states during the processing of subsequent iApply commands or during the processing of concurrent iProcs in iMerge parallelization.

Usage

iTake [*icl_resource*] [-persistent]

Description

An iTake command followed by an iWrite command to the associated taken resource permanently prevents Tessent IJTAG from altering the value during the processing of subsequent iApply commands. This is particularly useful if the affected resources are also involved in the configuration of the scan paths or in the justification of observation paths, in which case the retargeting software usually changes the states in its sole discretion. A state on a port or the content of a register that is taken by an iTake command becomes unchangeable by the retargeting software after the iApply that runs an iWrite to that port or register. Prior to being taken and written, the retargeting software is free to change the value (for example, to reconfigure the scan network or to justify observation paths). Once written by an iWrite command in the PDL, the constraint can be lifted only by an iRelease command for that resource. The resources remain taken until the end of the iProc that contains the iTake command, if they are not explicitly released by means of iRelease before.

In the context of an iCall command that is to be merged with other iCall commands by means of iMerge, an iTake command applies the same restrictions to the resources as in the single PDL command stream, but in addition, it reserves the resource for exclusive use by the iProc containing the iTake. Once an iProc has taken a resource, values assigned to the resource with iWrite are not permitted to be altered by concurrent iCalls until the resource is explicitly released with iRelease. Clock frequencies on ports that are taken as a resource cannot be changed by concurrent iCalls, either.

Arguments

- *icl_resource*

  An optional value that specifies the name of a register, a port or an instance. If an instance is specified, then all registers, ports and instances in *icl_resource* are taken. If you issue the command without this argument, the effect is the same as if an empty string was passed. In this case, the effective instance is the one that is determined by the ICL instance path of the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

currently processed iProc and by the iPrefix command. All registers and ports of this instance are taken.

- -persistent

  An optional switch that specifies that the iTake is not released at the end of the iCall that invoked it. The iTake carries forward into the following pattern set if it was created with the "open_pattern_set -no_initial_reset" command.

  Use this switch when you want to set a value on an object and prevent the solver from changing that value in the subsequent iCall. For example, if you want to fix an object to a value in an iCall referenced by a ProcedureStep wrapper, and you want the effect to persist in the following ProcedureStep and TestStep sequences, use "iTake -persistent" in the first iProc otherwise its effect is cleared once the iProc is done and does not carry over to the next iCalls.

Return Values

None.

Examples

iTake RegA
iTake

Related Topics

iCall

iMerge

iPrefix

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iTopProc

Context: all contexts

Mode: setup, analysis (patterns contexts only), insertion

Prerequisites: The iProcsForModule has been specified.

Specifies a PDL procedure that can run later when referenced by the iCall command, when its associated ICL module is the top level module.

Usage

iTopProc ***name variables body***

Description

The iTopProc command is identical to the iProc command except that it can only be iCalled against the top level ICL module. An iTopProc is never retargetable upward.

The purpose of an iTopProc is to define an iProc that enables the iForcePort and iComparePort commands inside it. Those two commands are not retargetable upward, therefore they cannot be used inside an iProc that is meant to be retargetable.

You can only iCall an iTopProc when the associated ICL module is the current ICL top module, that is, when it matches the current design. You cannot iCall an iTopProc from within an iProc. However, you can iCall an iProc or an iTopProc inside an iTopProc.

Arguments

- ***name***

  A required name for the iProc, which must be a simple scalar identifier. A simple scalar identifier starts with a letter and contains only letters, decimal digits, and underscores.

- ***variables***

  A required Tcl list of variable name or variable-value pairs enclosed in braces ({}). The list of variables can contain either simple identifiers or identifier-value pairs, where the value specifies the default value of the variable. All simple identifiers must be in the list before the identifier-value pairs.

- ***body***

  A required list of commands to be run when the procedure is called by means of the iCall command.

Return Values

None

Related Topics

delete_iprocs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_iproc_argument_list

get_iproc_list

iCall

iProc

iProcsForModule

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iUseProcNameSpace

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Specifies a name space that is used for all subsequent iCalls made within the iProc in which the iUseProcNameSpace command appears.

Usage

iUseProcNameSpace [*name_space*]

Description

Specifies a name space that is used for all subsequent iCalls made within the iProc in which the iUseProcNameSpace command appears.

The scope of this command is limited to within the iProc in which it appears. Invoking this command a second time within the same iProc changes the name space to the new value of *name_space*.

If you invoke iUseProcNameSpace with the - (single dash) argument or without the *name_space* argument, it resets previous name space specifications to mentor::iprocs, which is the default.

Arguments

- *name_space*

  An optional string that specifies the name space.

  The name space name can be - (single dash) or it must be formed from an initial upper or lower case alphabetic character followed by an optional sequence of upper or lower case alphanumeric or underscore characters ([a-zA-Z][a-zA-Z0-9_]*).

Return Values

None

Examples

```
iUseProcNameSpace MyNameSpace //set namespace to MyNameSpace
iUseProcNameSpace // return to root name space
```

Related Topics

get_iproc_argument_list

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_iproc_list

iCall

iProc

iProcsForModule

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iWrite

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the "patterns -ijtag" context, to use this command you must have previously opened a pattern set with the open_pattern_set command. In the "dft", "patterns -scan", and "patterns -scan_retargeting" contexts, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see "IJTAG and ATPG in Tessent Shell" in the *Tessent IJTAG User's Manual*.

Adds a write operation to the command queue to be solved by the next iApply command.

Usage

iWrite [***icl_instance_path***.]***writable_icl_object*** *value*

Description

Adds a write operation to the command queue to be solved by the next iApply command.

The ICL objects that can be written are DataRegisters, ScanRegisters, DataOutPorts, DataInPorts or an alias of any of these objects. A value to be written must be specified as the second argument. How the iWrite commands are interpreted by the iApply command is described in the iApply command description.

For a description of how to associate symbolic variables with iWrite commands, see "PDL Retargeting With Symbolic Variables" in the *Tessent IJTAG User's Manual*.

Arguments

- [***icl_instance_path***.]***writable_icl_object***

  Specifies the ICL objects that are to be written. You can specify to add write operations for the following ICL objects: DataRegisters, ScanRegisters, DataOutPorts, DataInPorts, or an alias of any of these objects. The ICL objects must be found at the computed effective_icl_instance_path. The effective_icl_instance_path is described in detail in the iCall command description.

- *value*

  A required constant number represented with Tcl binary, hexadecimal, or integer syntax or with a string corresponding to an enumeration symbol associated to the written objects in the ICL description.

  The X value has special meaning. If you use the iWritePort command on a bused port and any of the bit is specified with an X value, it means to leave that bit unchanged and preserve its previous value as if the bit was not even part of the iWrite command as shown here

  ```
  iWrite DI[3:0] 0b0xx1
  ```

  is equivalent to

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
iWrite DI[3] 0b0
iWrite DI[0] 0b1
```

Return Values

None

Examples

The following example requests the writable_icl_object mode to be written with the value represented by the enumeration symbol goNogo defined in ICL. The two requested write operations are solved when the iApply command is run.

iWrite mode goNogo
iWrite Count 0xa6
iApply

Related Topics

iApply

iCall

iPrefix

iRead

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

launch_sid_tester

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Directs Tessent Shell to launch the SID tester process on a local host using the selected_tester and the port number as specified in the configuration data file.

Usage

launch_sid_tester **-cdp** ***cdp_path*** [-new]

Arguments

- **-cdp** ***cpd_path***

  A required switch and string pair that specifies a CDP.

- -new

  An optional switch that specifies to create a new CDP at the location specified by the -cdp switch. By default, the tool opens an existing CDP.

Description

Directs Tessent Shell to launch the SID tester process on a local host using the selected_tester and the port number as specified in the configuration data file. Additionally, this command specifies an existing CDP or initializes a new CDP.

Note

You can only work with one SID tester at a time. To launch a new SID tester you must first shutdown the current SID tester with the shutdown_sid_tester command.

During the launch, Tessent Shell passes the hardware and pin map configuration data to the SID tester so that the SID tester can perform the proper setup tasks.

You must have loaded the configuration data file with the read_config_data command prior to running this command.

Note

SVF-based CDPs are configured with asynchronous clocking. The tool does not provide clock waveforms to the DUT, therefore you must source the appropriate clocks yourself.

Examples

The following example initializes a new CDP named “cdp1” and launches the SID tester process.

```
launch_sid_tester -cdp ./cdp1 -new
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Refer to "Running Non-Tessent IJTAG Instruments" in *Tessent SiliconInsight User's Manual for Tessent Shell* for a sample usage scenario.

Related Topics

read_config_data

shutdown_sid_tester

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

lock_current_registration

Context: unspecified, all contexts

Mode: all modes

Locks all previously registered commands and attributes preventing them from being unregistered and modified.

Usage

lock_current_registration

Description

Locks all previously registered commands and attributes preventing them from being unregistered and modified.

After verifying that any registered commands and attributes work correctly, you can make them into a plugin that Tessent Shell automatically loads during initialization. Place your files in a directory called <*plugin_dir*>/*tcl_modules*/<*my_dir*> where <*plugin_dir*> and <*my_dir*> are arbitrary directory names. Set the TESSENT_PLUGIN_PATH environment variable to point to the <*plugin_dir*> directory and any files with a *.tcl* extension found inside <*plugin_dir*>/*tcl_modules*/<*my_dir*> are evaluated into the Tcl namespace ::tcl_modules::<*my_dir*>. Like the PATH environment variable, you can specify more that one search location by separating multiple directory names with a colon.

If you name your *plugin_dir* directory name “tessent_plugin” and place it one level above the bin directory of Tessent Shell, it is automatically loaded unless the environment variable TESSENT_PLUGIN_IGNORE_DEFAULT_PATH is defined. Once the plugins are loaded, the command lock_current_registration is issued so that they cannot be unregistered accidentally.

Arguments

None

Examples

The following example registers a string attribute called myAtt for the object type pin and then issues a lock_current_registration command to make it impossible to unregister the attribute.

```
register_attribute -name myAtt -value_type string -object_type pin
lock_current_registration
```

Related Topics

register_attribute

register_tcl_command

unregister_attribute

unregister_tcl_command

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

macrotest

Context: dft -edt, patterns -scan

Mode: analysis

Generates manufacturing test patterns for embedded logic and memories referred to as macros.

Usage

macrotest {-**MULtiple_macros** ***macro_filename***}
[-L_H | -NO_L_H] [-VARY_observe_by_pattern] [-VERBose | -NO_VERBose]
[-VERIfy | -NO_VERIfy] [-MASK_nonobservation_sites]
[-NO_FIll_patterns] [-TE_observation_only | -LE_observation_only]
[-NO_REPOrt_observation_candidates | -REPOrt_observation_candidates]
[-FAILure_analysis] [-EXHAUSTIvely_search]
[-SKIP_PATterns *num_to_skip*] [-FIRST_OBS_pattern]

or

macrotest {{***ID#*** | ***pin_pathname*** | ***instance_name***} ***pattern_filename***}
[-L_H | -NO_L_H] [-VARY_observe_by_pattern] [-VERBose | -NO_VERBose]
[-VERIfy | -NO_VERIfy] [-MASK_nonobservation_sites]
[-NO_FIll_patterns] [-TE_observation_only | -LE_observation_only]
[-NO_REPOrt_observation_candidates | -REPOrt_observation_candidates]
[-FAILure_analysis] [-EXHAUSTIvely_search]
[-SKIP_PATterns *num_to_skip*] [-FIRST_OBS_pattern]

Description

Runs the MacroTest utility to automatically generate manufacturing test patterns for embedded logic and memories referred to as macros. This utility is useful for applying specific patterns that test the insides of a macro, thereby improving overall IC test quality.

Restriction

The macrotest command does not work with SSN.

Note

Although a macro is typically embedded logic or a memory, it can also be a disjoint set of internal sites or a single block of hardware represented by an instance in HDL.

MacroTest reads patterns from the text file you provide and converts them into scan-based manufacturing test patterns. These test patterns, when applied to the chip, deliver the underlying patterns to the appropriate embedded location through intervening logic. The following is a brief summary of the capabilities of MacroTest:

- Has no impact on area or performance.
- Enables you to test multiple macros in parallel.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Supports user-selected scan observation points.
- Enables you to define macro output values that do not require observation.
- Supports synchronous memories; for example, supports positive (or negative) edge-triggered memories embedded between positive (or negative) edge-triggered scan chains.

As a special form of the run command, the macrotest command uses all existing ATPG and cell constraints and limits.

Note

To obtain a high degree of precision for resolving any conversion problems, MacroTest simulates one pattern at a time, rather than 32 at a time.

The observability of some patterns may not be verified. Pin(s) and pattern are reported and whether or not observability was verified, and then processing stops. Typically, this is because a scan chain clock and an input on the macro are shared. Also, pulsing the macro pin pulses the scan clock, causing the scan cells to update, making the observability invalid. In such cases, specify a sequential MacroTest pattern such that you specify the macro input values to read, including a pulse of the shared clock, in the first cycle of the pattern. Specify the known output values in the second cycle. MacroTest uses the sequential engine for such patterns, permitting a full cycle for the values to come out of the macro and propagate to the scan cells where they are captured. Such operation of a shared clock should let you verify the macro's observability.

If the macro being tested is a RAM, the MacroTest patterns might perform several writes followed by several reads. In this situation, each write and read becomes a separate scan pattern with its own scan load. If loading the scan chain causes spurious writes into the RAM, Tessent FastScan destroys the data, making the reads access invalid data rather than the data that was written; therefore, the macro should be able to hold state while operating the scan chain to enable a sequence of inputs to be converted to a valid sequence of scan patterns (with a scan load per scan pattern).

For the rare case where macro outputs have paths to both leading edge and trailing edge capturing scan cells, the macrotest command includes switches that let you specify either trailing edge or leading edge observation. When you prepare the MacroTest patterns file, you can specify when to capture or observe so that the pattern matches the selection. For more information, see the -Te_observation_only and -Le_observation_only switches.

For more information about using the macrotest command, see "MacroTest Overview" in the *Tessent Scan and ATPG User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

When testing multiple macros, each macro opens a pattern file. When the number of open files reaches your operating system-specific limit, no more files can be opened. This may force MacroTest to fail. To avoid this problem, you may need to issue the appropriate version of limit, unlimit, or ulimit Linux shell commands before running MacroTest to increase the maximum number of files that can be open at one time. Check your Linux documentation for the appropriate command.

Arguments

- **-MULtiple_macros *macro_filename***

  A switch and string pair that tests multiple macros in parallel using the macrotest commands in the ***macro_filename*** file. The file must contain only macrotest commands, one for each of the macros to test in parallel. Siemens EDA recommends that you first test each macro individually in a separate run. Individual runs permit analysis if success is not possible for any single macro. When you test each macro successfully, add its corresponding command to ***macro_filename*** and remove it from the dofile (or command line).

  Note

  If you successfully tested a macro individually, you can probably test it successfully in parallel. However, Siemens EDA cannot guarantee this due to the enormous variety of possible embeddings.

  If you use the -Multiple_macros switch, you must include in the file all macros you want to be simultaneously tested. You cannot use this switch and also specify a pattern_filename or individual macro (by its ID#, pin_pathname or instance_name) on the same command line. You can test one subset of all macros using one -Multiple_macros run, with one macro_filename containing those macros and their pattern file specifications, and then test another subset by using another -Multiple_macros run. If you use other arguments with the -Multiple_macros switch, then macrotest uses those arguments as the default arguments for all macros in the file. For an example that uses this switch, see the Examples section at the end of this command description.

- ***ID#***

  An integer that specifies the gate identification number of the object to use in the macrotest run. The value of the *ID#* argument is the unique identification number that is automatically assigned to a gate within the design during the model flattening process.

- ***pin_pathname***

  A string that specifies the name of the input or output pin of a Tessent Cell library model to use in the macrotest run.

- ***instance_name***

  A string that specifies the name of the instance to use in the macrotest run.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ***pattern_filename***

  A string that specifies the name of the file containing the set of patterns to be applied to the macro.

- -L_H

  An optional switch that represents {L,H} as {LO,HI} output values in the macro patterns file. An error is issued if a 0 or 1 output value is specified in the macro patterns file. This is the default.

- -NO_L_H

  An optional switch that uses {0,1} to specify {LO,HI} output values in the patterns file, rather than the default {L,H}. If the default {L,H} is used, checking is done to ensure that the pin direction matches (output pin for L,H; input pin for 0,1). If the option -NO_L_h is issued, no such checking is possible.

- -VARY_observe_by_pattern

  An optional switch that, if issued, causes MacroTest to assume that a macro may need to vary its observation path or change where it is observed at after the first successful observation path is discovered by MacroTest on the first pattern with known outputs. The default is to assume the same path can be reused without calling ATPG to verify that on each pattern.

- -VERBose

  An optional switch that causes all informative messages to be issued. Use this option when testing any macro for the first time, so that all information about the progress and possible issues are conveyed. This is the default.

- -NO_VERBose

  An optional switch that turns off the default verbose output. When using this option, important warning and informative messages may be left out.

- -VERIfy

  An optional switch that causes one extra simulation per pattern to verify each macro's controllability (the macro input values) and observability (the values captured into a scan cell or observed at a chip pin). The observability check verifies that complementing all macro outputs causes each scan cell or chip pin where observation is occurring to change its value if a defect caused the observed macro output to be faulty. This is the default, and you should never turn off this check to solve problems of failure to verify a pattern, which halts the macrotest run. Instead, you must debug the issue causing the failure. Failure to verify control values typically causes simulation or tester mismatches. Failure to observe typically means that defects in the macro cannot be detected on a tester, which defeats the entire purpose of creating macrotest patterns.

- -NO_VERIfy

  An optional switch that refrains from performing an extra simulation per pattern to verify that the macro input values were delivered by the pattern, and that changing the macro

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

outputs changes the observation sites. This switch should never be used rather than investigating the cause of verification failure. Failure to control of observe the macro either makes the pattern useless or predicts the wrong value and halts the tester.

- -MASK_nonobservation_sites

  An optional switch that places an X in all scan cells except the ones used for observation on a pattern-by-pattern basis.

  You need to specify both the -NO_FIll_patterns and the -MASK_nonobservation_sites switches to force only the scan cells used by macrotest to observe macro outputs to have known values for the scan unload. If you specify the -NO_FIll_patterns switch without also specifying the -MASK_nonobservation_sites switch, all values that propagate to scan cells are captured, regardless of whether they are from the macro outputs or from the known PIs and scan cells that fanout to both the macro inputs and other scan cells; that is, scan cells other than those observing macro outputs may have known scan-out values.

- -NO_FIll_patterns

  An optional switch that causes the unspecified bits in scan chains to remain at a value X, so that only the values needed to test the macro appear as known values. Using this option, you can see which scan chain and primary input values are needed to test the macro by using the write_patterns command. You can use the -MASK_nonobservation_sites option to convert captured values that are not needed but known to X in the scan-out.

- -TE_observation_only

  An optional switch that observes at only one of the trailing edge (TE) sites when a macro has both trailing edge and leading edge observation sites. The default is to observe at any site.

- -LE_observation_only

  An optional switch that observes at only one of the leading edge (LE) sites when a macro has both trailing edge and leading edge observation sites. The default is to observe at any site.

- -NO_REport_observation_candidates

  An optional switch that refrains from reporting the reachable observation sites. In some cases, the number of possible scan cells that an output can go to is large and the report long, so the default is not to report the candidates.

- -REport_observation_candidates

  An optional switch that reports the reachable observation sites, and whether those sites are already eliminated as observation candidates due to inability to capture. For example, a cell constraint to observe X prevents capture. If you use this switch, a list of possible observation sites (candidates) per macro output pin is displayed at the beginning of the macrotest run. The -Report_observation_candidates switch is sometimes useful for discovering that different macros in a multiple macro run have the same observation site possibilities. It also helps explain why it may be impossible to observe both macros at the same time.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The observe_at syntax in the macrotest patterns file enables you to specify where to observe one or more of the outputs of a macro for all patterns. The -Report_observation_candidates switch is useful for selecting the name(s) to use as observation site(s).

Note

To quickly test many macros at once, correct observe_at guidance can reduce the runtime. Incorrect guidance leads to failure to succeed where success might have been possible. Sometimes, guidance is necessary to obtain macrotest patterns in a reasonable runtime.

- -FAILure_analysis

  An optional switch that specifies the tool to determine why ATPG cannot succeed. Because this failure analysis can consume significant processing time, the default is to avoid failure analysis for rapid debug iteration. You should use this switch only as a last resort to determine why the embedding prevents justification of the macrotest pattern that causes ATPG to fail.

- [-EXHAUSTIvely_search]

  An optional switch that performs an exhaustive search. For some situations, an exhaustive search is needed; however, MacroTest can run for cpu weeks when hundreds of macros are tested together in one run. This switch only solves problems where one macro interacts with another macro; this switch does not solve problems where a single macro conflicts with its own needed boundary values or netlist requirements surrounding that macro. By default, this switch is omitted due to the infrequency of such debilitating macro interactions and the typical runtime cost to find them if they occur.

- -SKIP_PATterns *num_to_skip*

  An optional switch and integer pair that skips the first *num_to_skip* patterns. This option is used for analyzing problems after pattern creation when a MacroTest run fails. For example, if the third pattern fails to be converted (the transcript shows two patterns converted and simulated and then reports a failure), specify 2 for *num_to_skip*, so that MacroTest skips the first two patterns and starts on the third pattern. This enables you to quickly iterate on the problematic pattern. The -SKIP_PATterns *num_to_skip* option is not used for production patterns.

- -FIRST_OBS_pattern

  An optional switch used only for analysis when MacroTest fails. This option skips all of the control-only patterns and processes the first pattern with known (observable) macro outputs as if it were the first pattern in order to speed iterations and get to the problematic patterns sooner.

Examples

The following example tests all of the macros in parallel (those with a macrotest command in the “macrofile” file). The example changes the default, whereby output 0 values in MacroTest patterns are indicated by L, and output 1 values are indicated by H. Instead, 0 and 1 are used respectively. Note that macrotest commands inside the macrofile can override the -NO_L_H

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

switch below by selecting an alternative option. For example, one or more of the commands could use -L_H if those macros have files that use L and H to indicate 0 and 1 output values.

macrotest -multiple_macros macrofile -NO_L_H

For more examples, refer to "MacroTest Overview" in the *Tessent Scan and ATPG User's Manual*.

Related Topics

add_faults

set_contention_check

set_gate_report

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

map_tsdb_container_paths

Context: dft, patterns

Mode: setup, analysis, insertion

Maps Tessent Shell Database (TSDB) container relative or absolute file paths to logical names and logical names to absolute paths.

Usage

map_tsdb_container_paths ***path_list***
[-map_to {absolute_paths | logical_names}]
[-config_object *config_object*]
[-error_string_prefix *error_string_prefix*]

Description

The map_tsdb_container_paths command helps you access files from a TSDB from your dofile without having to know the location of the TSDB.

Arguments

- ***path_list***

  A required Tcl list of relative file paths, absolute file paths, or file paths with a logical name prefix. The recognized logical name prefixes are:

  - dft_inserted_design(<module_name>_<design_id>)
  - instrument(<module_name>_<design_is>_<ip_type>)

  The *ip_type* value is any of the directory name suffixes found in the *<tsdb>/instruments* directory, such as "ijtag" for the directory that contains the IJTAG nodes.

  See "Tessent Shell Data Base (TSDB)" on page 4807 for more information on TSDB container directories.

- -map_to {absolute_paths | logical_names}

  An optional switch and string pair to choose what sort of mapping to perform.

  absolute_paths — Use this option to map a Tcl list of logical names to absolute paths. Elements not starting with one of the recognized logical name prefixes are returned as-is. This is the default.

  logical_names — Use this option to map a Tcl list of filenames to logical names. Any path not pointing to elements within a *.dft_inserted_design* or *.instruments* directory within an open TSDB is returned unchanged.

  The following rules apply:

  - Maps a Tcl list of file paths to logical names when you specify the -map_to switch as "logical_name."

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Maps a Tcl list of logical names to absolute paths when the -map_to switch defaults to or is specified as “absolute_paths.”
- Any non-logical name remains unchanged when the -map_to switch defaults to or is specified as “absolute_paths.”
- Any path not pointing to elements within a *.dft_inserted_design* or *.instruments* directory within an open TSDB remains unchanged even when you specify the -map_to switch as “logical_name.”

  See “Tessent Shell Data Base (TSDB)” on page 4807 for more information on TSDB container directories.

- -config_object *config_object*

  An optional switch and string pair to associate an error message with the specified configuration element. Use this option if you are using this command within a proc that validates the value of a property. The error message, generated when a logical name is detected but no associated directory is found within the open TSDBs, is attached to the configuration element specified with this switch.

- -error_string_prefix *error_string_prefix*

  An optional switch and string pair to add a prefix to any error messages the command encounters. Use this option if you are using the map_tsdb_container_paths command within a proc that validates a value, and you want the error message to be prefixed with a string of your choice to provide more context to the error message. The error message is generated when a logical name is detected, but no associated directory is found within the open TSDBs.

  Tip

  Include the “\n” character in your string to split it over several lines.

Examples

In this example, the command maps the logical name of a *dft_inserted_design* directory to copy the SDC file contained within it to your synthesis workspace.

```
file copy [map_tsdb_container_paths dft_inserted_design(blocka_rtl2)/
blocka.sdc] synthesis_workspace
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

merge_cdp

Context: patterns -silicon_insight

Mode: setup, analysis

Merges legacy LV flow and Tessent-Shell-based ESOE CDPs into the current Tessent Shell-based CDP.

Usage

merge_cdp ***cdp_path*** [-replace]

Description

You use the resulting CDP along with the process_memory_failures command to convert failure log files produced from previously migrated raw STIL or raw SVF ESOE failure log files to bitmap files. Refer to "Performing Diagnosis From an ATE Test Program in Offline ATE Mode" in the *Tessent SiliconInsight User's Manual for Tessent Shell* for details.

When importing an LV flow or Tessent-Shell-based CDP into the current Tessent Shell CDP using this command, the tool does not perform CDP verification. Upon successful execution of this command, all tests originating from the merged CDP become accessible to Tessent SiliconInsight for Tessent Shell.

Arguments

- ***cdp_path***

  A required string that specifies the path to the LV flow or Tessent Shell-based CDP that you want to merge.

- -replace

  An optional switch that causes the tool to replace the existing tests in the current Tessent Shell-based CDP when there are name conflicts with the test names in the CDP that you are merging.

Examples

As shown in the following example, specify the merge_cdp command after specifying the setup specification and invoking Tessent SiliconInsight.

```
set_context patterns -silicon_insight
set_silicon_insight_option -cdp_verification off
set_silicon_insight_option -ignore_uncontacted_pin_data on
read_config_data si.setup
start_silicon_insight
merge_cdp my_old_cdp
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

migrate_layout

Context: patterns -scan_diagnosis

Mode: analysis

Generates an LDB directory hierarchy that contains the specified preexisting LDB.

Usage

migrate_layout ***ldb_directory*** **-ladb** ***preexisting_ldb*** [-vdb *preexisting_vdb*]

Description

Generates an LDB directory hierarchy that contains the specified preexisting LDB.

Use this command to copy a preexisting LDB into the LDB directory hierarchy that became available with the Tessent Diagnosis 2013.2 release. This command enables you to upgrade to the LDB directory without regenerating the LDB from the LEF/DEF files. The tool automatically copies the preexisting LDB into the new LDB directory.

For the DFM diagnosis flow, use this command to additionally copy a preexisting violations database (VDB), which was generated automatically during DFM diagnosis, into the new LDB directory.

Arguments

- ***ldb_directory***

  A required string that specifies the pathname to a new LDB directory.

- **-ladb** ***preexisting_ldb***

  A required switch and string pair that specifies the pathname to a preexisting LDB.

- -vdb *preexisting_vdb*

  An optional switch and string pair that specifies the pathname to a preexisting VDB.

Examples

The following example generates a LDB directory named mylayoutdb. It then copies the example2.ldb and example_dfm.vdb databases into the LDB directory.

```
migrate_layout newlayoutdb -ladb preexisting.ldb -vdb preexisting_dfm.vdb
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

move_config_element

Context: dft

Mode: analysis

Moves a configuration element from one location to another.

Usage

move_config_element ***object_spec***
[-in_wrapper *wrapper_name_or_object*]
[-before *before*| -after *after* | -first | -last]
[-replace]

Description

Moves a configuration element from one location to another.

Repeatable and non-repeatable wrappers and repeatable properties can be moved from one wrapper to another.

Arguments

- ***object_spec***

  A value that specifies the name of a configuration element or a collection containing one configuration element to be moved. When using the -in_wrapper option, the name is the relative pathname with respect to the specified wrapper. When not using the -in_wrapper option, the name is the complete pathname with respect to the root of the partition. To move an element from one wrapper to the next, you must use the get_config_elements or the "get_config_value -object" command to get the object and use it as the object_spec. The moved element must not already exist in the destination wrapper unless the -replace option is used, in which case the existing element is deleted before being replaced by the moved one.

  When the permitted content of the destination parent wrapper is defined with Wrapper, the metadata definition of the source and destination element must be identical (both definition must reference the same metadata definition). If this is not the case, an error message is generated.

- -in_wrapper *wrapper_name_or_object*

  An option and value pair that specifies that the configuration element is to be moved inside a specific wrapper. The wrapper_object_spec can be the complete name of a wrapper relative to the root of the partition or a collection containing a single wrapper element as returned by get_config_elements or "get_config_value -object". When the -in_wrapper is not specified, the destination parent wrapper is assumed to be the same as the source parent wrapper unless the -before or -after option is specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -before *before*

  An optional switch and value pair that specifies the configuration element in front of which to move the new configuration element. When the -before option is used with a collection containing a configuration element object, the -in_wrapper option cannot be used because the parent wrapper is extracted from the object.

  When config_object_spec is a name, it must be a name relative to the wrapper specified with the -in_wrapper option. If the -in_wrapper option is not specified, it is a name relative to the root of the partition.

  The -before option is mutually exclusive with the -after, -first, and -last options. When none of those four option is specified, -last is assumed.

- -after *after*

  An optional switch and value pair that specifies a configuration element after which to move the new configuration element. When the -after option is used with a collection containing a configuration element object, the -in_wrapper option cannot be used because the parent wrapper is extracted from the object.

  When config_object_spec is a name, it must be a name relative to the wrapper specified with the -in_wrapper option. If the -in_wrapper option is not specified, it is a name relative to the root of the partition.

  The -after option is mutually exclusive to the -before, -first and -last options. When none of those four option is specified, -last is assumed.

- -first

  An optional switch that specifies to move the new configuration element before any other element in the parent wrapper.

  The -first option is mutually exclusive to the -before, -after and -last options. When none of these four options is specified, -last is assumed.

- -last

  An optional switch that specifies to move the new configuration element after all other elements in the parent wrapper.

  The -last option is mutually exclusive to the -before, -after, and -first options. When none of those four option is specified, -last is assumed.

- -replace

  An optional switch that suppresses the error that is normally generated when moving a configuration element from one wrapper to another and the leaf_name conflicts with an existing element in the destination wrapper. The error only occurs if the name is defined as not repeatable in the metadata. Instead of an error, the existing element is first deleted.

Examples

The following example moves configuration elements within and across parent wrappers.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_transcript_style input_only
set dft [add_config_element DftSpecification(moda,rtl)]

```
{/DftSpecification(moda,rtl)}
```

set ijtag [add_config_element IjtagNetwork -in_wrapper $dft]

```
{/DftSpecification(moda,rtl)/IjtagNetwork}
```

set host [add_config_element HostScanInterface(1) -in_wrapper $ijtag]

```
{/DftSpecification(moda,rtl)/IjtagNetwork/HostScanInterface(1)}
```

set sib1 [add_config_element sib(1) -in_wrapper $host]

```
{/DftSpecification(moda,rtl)/IjtagNetwork/HostScanInterface(1)/Sib(1)}
```

set sib2 [add_config_element sib(2) -in_wrapper $host]

```
{/DftSpecification(moda,rtl)/IjtagNetwork/HostScanInterface(1)/Sib(2)}
```

set sib3 [add_config_element sib(3) -in_wrapper $host]

```
{/DftSpecification(moda,rtl)/IjtagNetwork/HostScanInterface(1)/Sib(3)}
```

set sib4 [add_config_element sib(4) -in_wrapper $host]

```
{/DftSpecification(moda,rtl)/IjtagNetwork/HostScanInterface(1)/Sib(4)}
```

report_config_data $dft

```
DftSpecification(moda,rtl) {
  IjtagNetwork {
    HostScanInterface(1) {
      Sib(1) {
      }
      Sib(2) {
      }
      Sib(3) {
      }
      Sib(4) {
      }
    }
  }
}
```

move_config_element $sib3 -in_wrapper $sib2

report_config_data $dftDftSpecification(moda,rtl)

```
> report_config_data $dftDftSpecification(moda,rtl)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{
  IjtagNetwork {
    HostScanInterface(1) {
      Sib(1) {
      }
      Sib(2) {
        Sib(3) {
        }
      }
      Sib(4) {
      }
    }
  }
}
```

#Because -in_wrapper is not specified, $sib one is moved within its parent wrapper

move_config_element $sib1 -last

report_config_data $dftDftSpecification(moda,rtl)

```
 {
  IjtagNetwork {
    HostScanInterface(1) {
      Sib(2) {
        Sib(3) {
        }
      }
      Sib(4) {
      }
      Sib(1) {
      }
    }
  }
}
```

#example showing the use of a config element name in the -in_wrapper option

move_config_element $sib1 -in_wrapper \ /DftSpecification(moda,rtl)/

```
IjtagNetwork/HostScanInterface(1)/Sib(2)
```

report_config_data $dftDftSpecification(moda,rtl)

```
{
  IjtagNetwork {
    HostScanInterface(1) {
      Sib(2) {
        Sib(3) {
        }
        Sib(1) {
        }
      }
      Sib(4) {
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

#Because -in_wrapper is not specified, $sib one is moved within its parent wrapper

move_config_element $sib1 -first

report_config_data $dftDftSpecification(moda,rtl)

```
{
  IjtagNetwork {
    HostScanInterface(1) {
      Sib(2) {
        Sib(1) {
        }
        Sib(3) {
        }
      }
      Sib(4) {
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

move_connections

Context: all contexts

Mode: insertion

Moves connections from pin or net objects to pin objects.

Usage

move_connections **-from *obj_spec1* -to *obj_spec2*** [-net_name *net_name*] [-silent]

Description

Moves connections from pin, port or net objects to pin objects.

Without an explicit type obtained from get_nets, get_pins or get_ports, the -from ***obj_spec1*** argument is assumed to be a net when the -to ***obj_spec2*** argument is one level below the hierarchy of ***obj_spec1***. For example, the following command moves the u1/a pin to the net by the name of u1/a, instead of the pin, because u1/u2/b is below u1.

```
move_connections -from u1/a -to u1/u2/b
```

In another example, u2/b is not below u1. In this case, the command creates the connection to the u2/b pin instead of to the net.

```
move_connections -from u1/a -to u2/b
```

The specific behavior of the move_connections command varies based on the object type found inside ***obj_spec1***.

If the object within ***obj_spec1*** is:

- A pin

  The command moves the net connected to this pin to the pin specified in ***obj_spec2*** and the pin in ***obj_spec1*** is left unconnected.

- An input port or a net that is also an input port

  The command renames the net everywhere it is used and then connects it to the pin in ***obj_spec2***. This leaves the input port with no fanout. The renamed net name is the specified -net_name value or the original net name. When you do not specify the -net_name option, the tool uniquifies the original name using the _ts# suffix.

- A net that is also an output port

  The command renames the net only at its source and then connects it to the pin in ***obj_spec2***. This leaves all original net fanout sink connections untouched and undriven. The renamed net name is either the -net_name option value, when specified, or the original net name, uniquified with the _ts# suffix. This enables inserting a gate that intercepts the original port driver's total fanout.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- An output port

  The command renames the same-named port net everywhere it is used and then connects it to the pin in ***obj_spec2***. If the port driver originally fanned out to other sinks within the instance, this leaves only the output port branch dangling. The renamed net name is either the -net_name option value, when specified, or the original net name, uniquified with the _ts# suffix. This enables inserting a gate that intercepts only the output port branch of the original driver's total fanout.

- A net that is not also a port

  The command renames the net everywhere it is used, except at its driver source, and then connects it to the pin in ***obj_spec2***. This leaves the driver source connected to the original net with no fanout. The renamed net name is the specified -net_name value or the original net name. When you do not specify the -net_name option, the tool uniquifies the original name using the _ts# suffix by default. You can change the default with "set_insertion_options -net_uniquification_suffix".

Special Cases to Consider

- If ***obj_spec1*** has an inherent direction (sink or source), then its direction must match the ***obj_spec2*** direction, otherwise an error is given. For that matter:
  - An input port or port/net is a source.
  - An output port or port/net is a sink.
  - An input pin is a sink.
  - An output pin is a source.
- If ***obj_spec2***'s parent instance is below ***obj_spec1***, then:
  - If ***obj_spec1*** is an explicit *output* or *inout* pin (that is, [get_pins …]):
    - If ***obj_spec2*** is an input pin, then ***obj_spec1*** fan-in is intercepted from inside its instance with the same behavior as if it were an output port (see above).
    - If ***obj_spec2*** is an output pin, then ***obj_spec1*** connection is moved from outside, creating a new output port on its instance, which is now driven by ***obj_spec2***. ***obj_spec1*** gets disconnected in its parent.
  - If ***obj_spec1*** is an explicit *input* pin (that is, [get_pins …]):
    - ***obj_spec1*** connection is moved from outside its instance, creating a new input port on that instance, fanning out to ***obj_spec2***.
  - If ***obj_spec1*** type is unspecified and matches the name of:
    - An output port: for backwards-compatibility, it is assumed to be an output port in the above connection algorithms.
    - An input or output pin, then it is assumed to be a net.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The ***obj_spec1*** must be a Tcl list of one or more pin or net names, or a collection of one or more pin or net objects. The ***obj_spec2*** must be a Tcl list of one or more pin names or a collection of one or more pin objects. The ***obj_spec1*** and ***obj_spec2*** must have exactly the same size.

Note

The -from and -to objects can exist at different levels of hierarchy, but a move across hierarchy is not recommended because it is possible to create a lot of unnecessary ports through the hierarchy. You should limit moves to within a single hierarchy by instantiating the intercepting cell in the parent instance of the node to be intercepted.

This command returns a collection containing the net object that ends up being connected to the pin object in ***obj_spec2***. If the -from object is a pin, the command returns the nets that were moved. If the -from object is a net, the command returns the nets that were created.

If an object does not exist in ***obj_spec1*** or ***obj_spec2*** and the -silent option is specified, the object is ignored and the move action skipped. If an object does not exist and the -silent option is not specified, the tool generates an error.

Exercise caution when using the move_connections command inside a non-uniquified design. You must only perform the move or disconnection once per module and once per generate loop count. For information about how to perform this robustly, see how the parent_instance and the leaf_name_hash attributes are used in Example 5 in the get_dft_cell command description. For information about the leaf_name_hash attribute, refer to the its description in "Instance" on page 3668.

Arguments

- **-from *obj_spec1***

  A required switch and value pair that specify the object to be moved. The ***obj_spec1*** must be a Tcl list of one or more port, pin or net names, or a collection of one or more port, pin or net objects.

- **-to *obj_spec2***

  A required switch and value pair that specifies the pin to which the net is moved to. The ***obj_spec2*** must be a Tcl list of one or more pin names, or a collection of one or more pin objects. The ***obj_spec1*** and ***obj_spec2*** must have exactly the same size.

- -net_name *name*

  An optional argument that specifies the base net name to create when moving a net. If you do not use this switch, the tool uses the original net name uniquified using the _ts# suffix.

- -silent

  An optional argument that ignores specified objects that do not exist and aborts the move_connections execution without generating an error message. If the move_connections command does not run successfully, the returned net object is null.

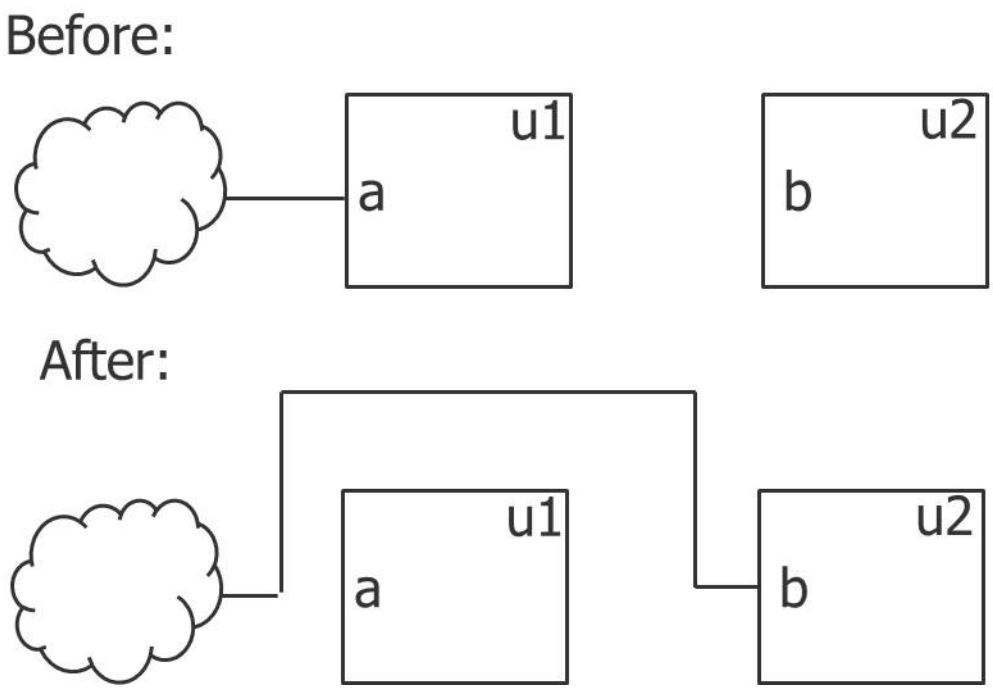
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

This following example moves the net connected to input pin u1/a to the input pin u2/b. After the operation, pin u1/a is left with no driver.

```
move_connections -from u1/a -to u2/b
```



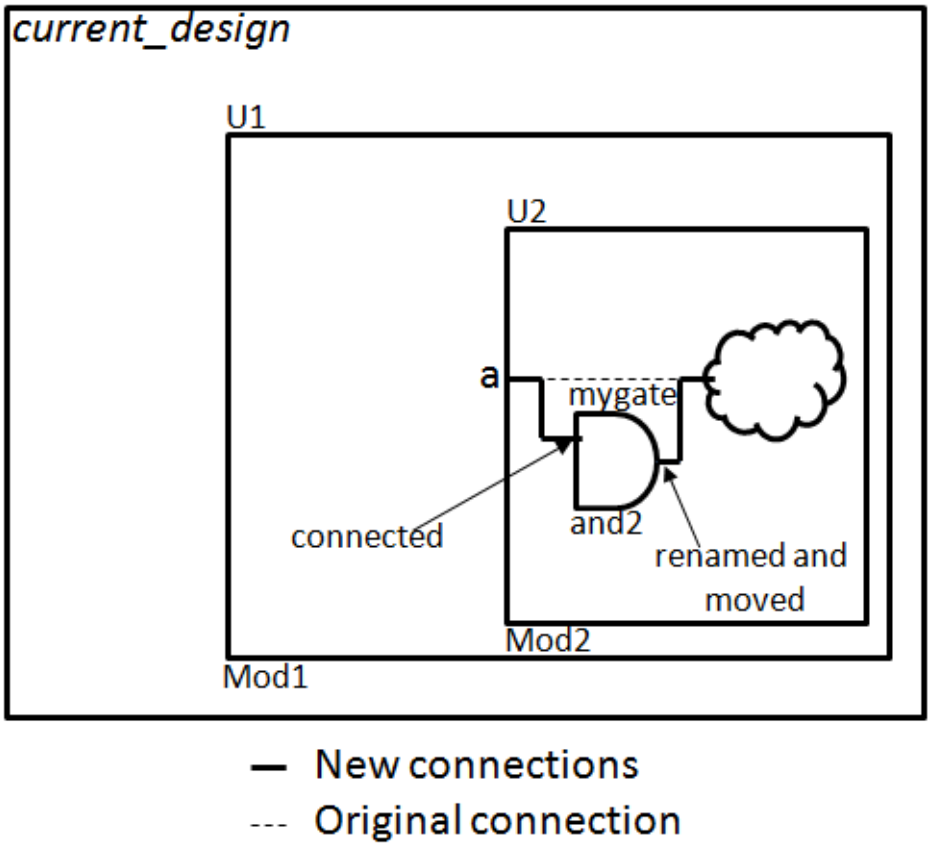
Example 2

The following example shows how to intercept an input pin the way it is done when you use the intercept_connection command. By using the get_nets introspection command, the net named u1/u2/a is returned instead of the pin as in the previous example. The parent_instance of the net automatically returns u1/u2, which results in placement of the intercepting gate inside the module as required.

```
set net [get_net u1/u2/a]
set gate_inst_name [lindex [get_attribute_value_list $net -name parent_instance] 0]/mygate
set gate_inst [create_instance $gate_inst_name -of_mod and2 -allow_inst_uniquification]
move_connections -from $net -to [get_pins y -of_instance $gate_inst]
create_connections $net [get_pins a -of_instance $gate_inst]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 5-8. Intercepting an Input Pin From Inside An Instance



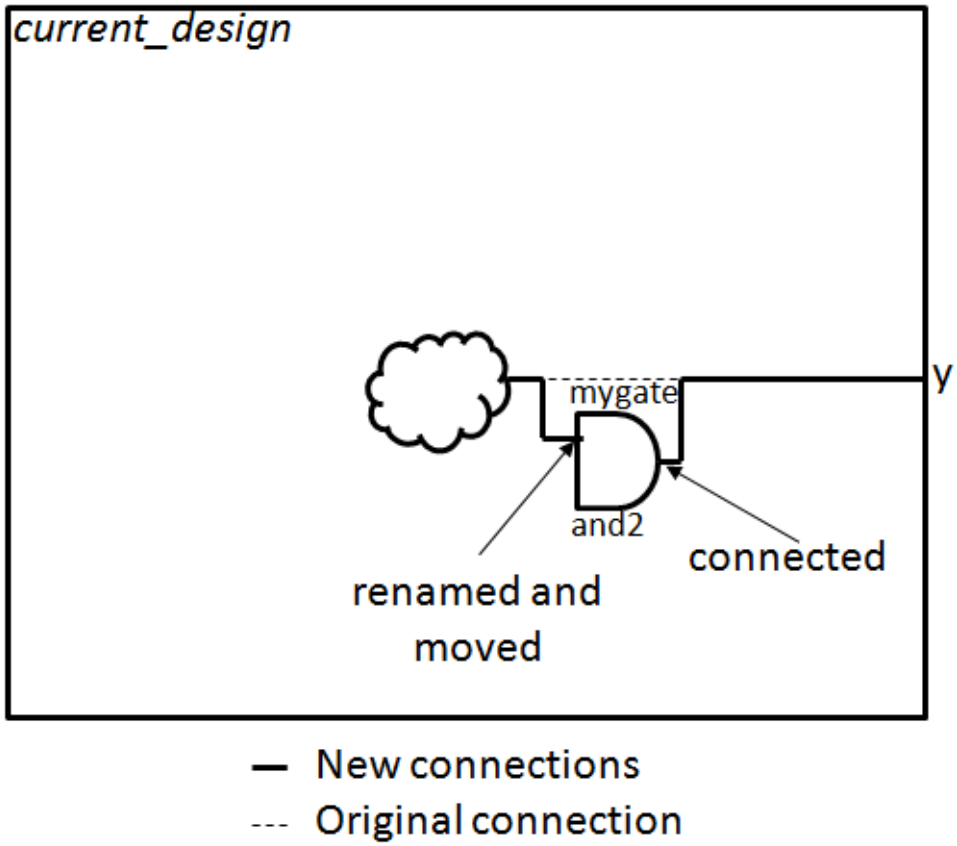
Example 3

This following example shows how to intercept an output port from the inside of the current design. The parent_instance of the port automatically returns a null value, which results in the intercepting gate being placed inside the root module as required. The net names created by the move_connections and create_connections commands are derived from the net named being moved or connected. You can use the -net_name option if you want a specific net name to be used instead.

```
set port y
set gate_inst_name [lindex [get_attribute_value_list $port -name parent_instance] 0]/mygate
set gate_inst [create_instance $gate_inst_name -of_mod mygate -allow_inst_uniquification]
move_connections -from $port -to [get_pins a -of_instance $gate_inst]
create_connections $port [get_pins y -of_instance $gate_inst]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 5-9. Intercepting an Output Port From Inside the Current Design



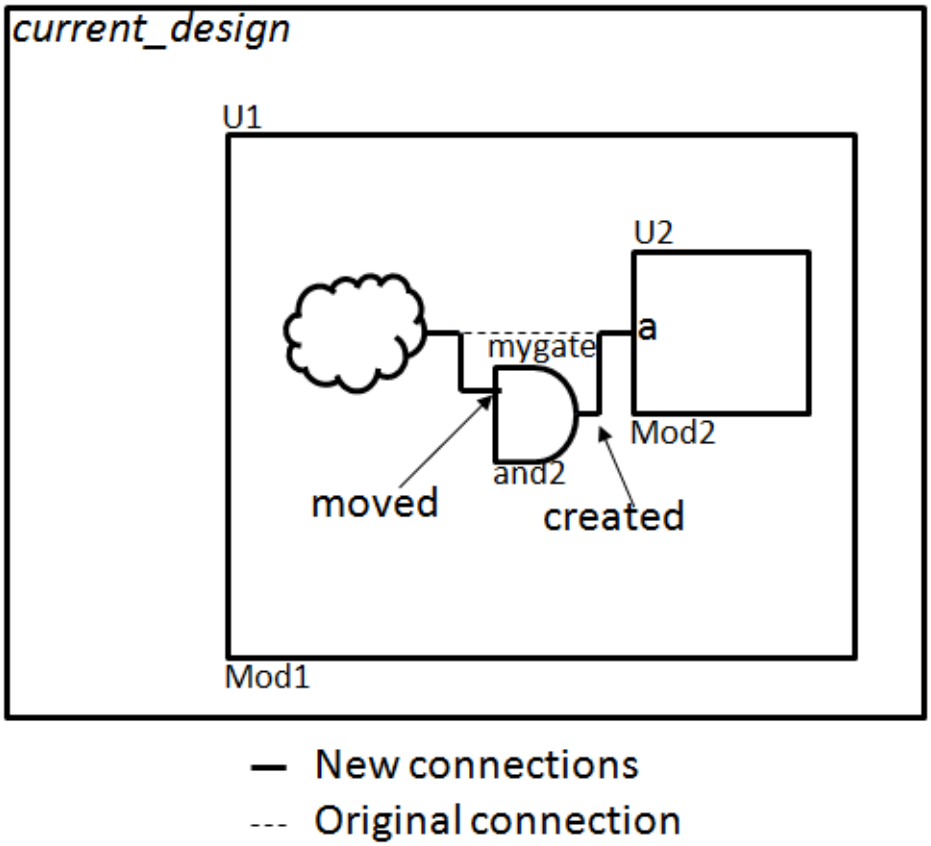
Example 4

This following example shows how to intercept an input pin from the outside of the instance. This example also shows the use of the parent_instance attribute to properly determine the location of the intercepting gate that must be placed in the same hierarchy level as the intercepted net. Notice that the return value of the create_instance command is used to update the gate_inst value in case the original name already existed inside the design, and the create_instance command created a new uniquified name. The net that was connected to the pin u1/u2/a ends up connected to pin “a” of $gate_inst. Following the move_connections operation, pin u1/u2/a is left open. The create_connections command is then used to reconnect it to the output of the intercepting gate.

```
set pin u1/u2/a
set gate_inst_name [lindex [get_attribute_value_list $pin -name parent_instance] 0]/mygate
set gate_inst [create_instance $gate_inst_name -of_mod and2 -allow_inst_uniquification]
move_connections -from $pin -to [get_pins a -of_instance $gate_inst]
create_connections $pin [get_pins y -of_instance $gate_inst]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 5-10. Intercepting an Input Pin From Outside An Instance



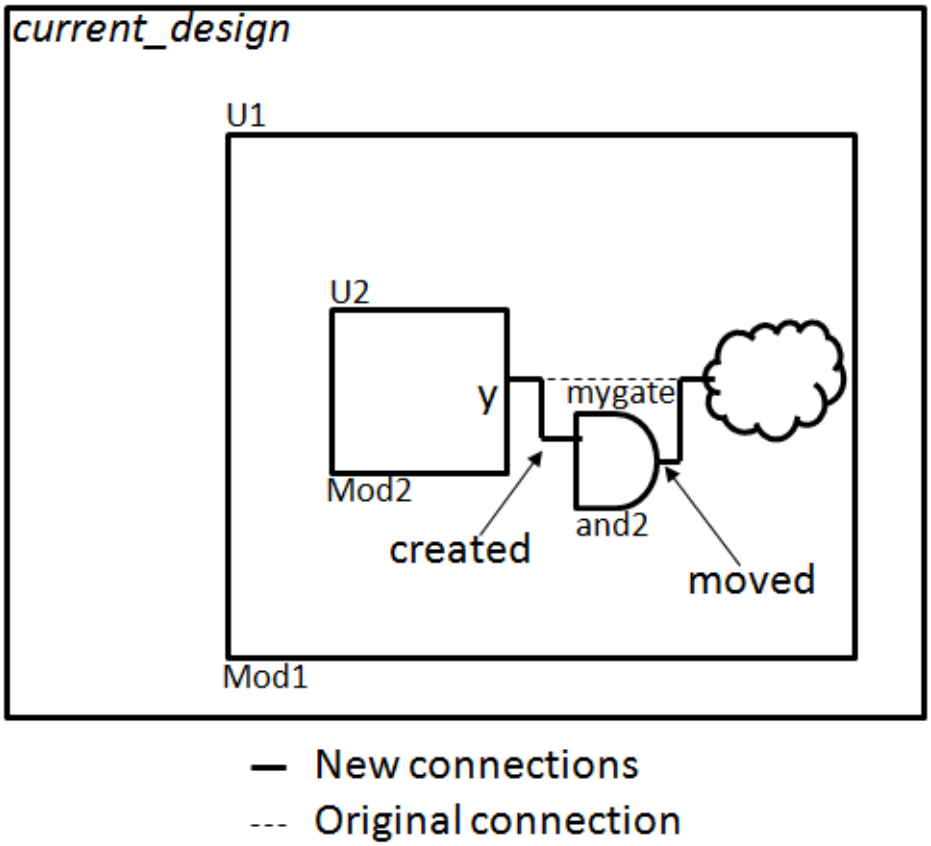
Example 5

The following example shows how to intercept an output pin from the outside of the instance. This example also shows the use of the parent_instance attribute to properly determine the location of the intercepting gate that must be placed in the same hierarchy level as the intercepted net. Notice that the return value of the create_instance command is used to update the gate_inst value in case the original name already existed inside the design, and the create_instance command created a new uniquified name. The net that was connected to pin u1/u2/y ends up connected to pin y of $gate_inst. Following the move_connections, pin u1/u2/y is left open. The create_connections command is then used to reconnect it to the input of the intercepting gate.

```
set pin u1/u2/y
set gate_inst_name [lindex [get_attribute_value_list $pin -name parent_instance] 0]/mygate
set gate_inst [create_instance $gate_inst_name -of_mod and2 -allow_inst_uniquification]
move_connections -from $pin -to [get_pins y -of_instance $gate_inst]
create_connections $pin [get_pins a -of_instance $gate_inst]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 5-11. Intercepting an Output Pin From Outside An Instance



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

no_transcript

Context: unspecified, all contexts

Mode: all modes

Turns off transcripting for the specified commands.

Usage

no_transcript { ***command1 command2 ...*** }

Description

Turns off transcripting for the specified commands.

This command is useful when you are writing Tcl procedures in which you do not want your embedded commands to show up in the transcript or log file. The no_transcript command is also not transcripted when running the Tcl procedure using the “source” command.

Arguments

- ***command1 command2 ...***

  A required string that specifies the commands that you do not want to appear in the transcript or log file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following excerpt from a Tcl procedure (see get_dft_cell, Example 5), shows a usage example for the no_transcript command:

```
##################################################################
### Section 5: Retrieves leaf instance name of gate if node is a
### repeated module instance and skips the interception
#################################################################

if {[info exists insertion_parent_leaf_instance_array($key)] && \
      [get_insertion_option -auto_uniquify_edited_modules] eq "off"} {
   set leaf_cell_inst "$insertion_parent_leaf_instance_array($key)"
   if {$leaf_block_id ne ""} {
      set cell_inst [get_instance \
         "${parent_instance_name}/${leaf_block_id}.${leaf_cell_inst}"]
   } else {
      set cell_inst [get_instance \
         "${parent_instance_name}/${leaf_cell_inst}"]
   }
} else {
   if {$leaf_block_id ne ""} {
      set cell_inst_name \
      "${parent_instance_name}/${leaf_block_id}.insertion_${cell_sel}"
   } else {
      set cell_inst_name \
         "${parent_instance_name}/insertion_${cell_sel}"
   }
   no_transcript {
      set cell_inst [create_instance $cell_inst_name -of_module \
         $dft_cell -allow_instance_uniquification]
   }

   set leaf_name [lindex [get_attribute_value_list $cell_inst -name \
      leaf_name] 0]
   set leaf_block_id [lindex [get_attribute_value_list $cell_inst \
      -name leaf_block_id] 0]
   if {$leaf_block_id ne ""} {
      # Remove the block_id part from the leaf instance name
      set leaf_name [string range $leaf_name \
         [string length $leaf_block_id] end]
   }
   set insertion_parent_leaf_instance_array($key) $leaf_name

   if {$insertion_type == "sink"} {
      no_transcript {
         move_connections -from $node_object \
            -to [get_pins ${input0} -of_instances  $cell_inst]
         create_connections [get_pins ${output} -of_instances \
            $cell_inst] $node_object
      }
   } else {
      no_transcript {
         move_connections -from $node_object -to [get_pins ${output} \
            -of_instances $cell_inst]
         create_connections $node_object [get_pins ${input0} \
            -of_instances  $cell_inst]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
            }
         }
      }

      if {$connect_in2} {
         no_transcript {
            create_connections $in2_object [get_pins $input1 -of_instances \
               $cell_inst]
         }
      }

      if {$connect_sel} {
         no_transcript {
            create_connections $sel_object [get_pins $select -of_instances \
               $cell_inst]
         }
      }
      return $cell_inst
   }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

open_layout

Context: patterns -scan, patterns -scan_diagnosis

Mode: analysis

Opens an existing Tessent Diagnosis tool-compatible Layout Database directory.

Usage

Layout-Aware Diagnosis Flow

open_layout ***layout_database_name*** [-VERIfy [-EXTRA_LAYout_hierarchy *extra_levels_of_hierarchy_in_layout* | -EXTRA_DESign_hierarchy *extra_levels_of_hierarchy_in_design*]] [-THreshold *percentage_match*] [-chip_design_name *top_design_name*]

Description

Layout-Aware Diagnosis Flow

Opens an existing Tessent Diagnosis tool-compatible Layout Database directory.

Any subsequent usage of this command first closes any existing layout before Tessent Diagnosis opens a new Layout Database. See "Layout-Aware Diagnosis" in the *Tessent Diagnosis User's Manual* for complete information on this flow.

In hierarchical layout-aware diagnosis, you must use the -chip_design_name option when you have a core that is instantiated in more than one design. See "Diagnosis for Hierarchical Design" in the *Tessent Diagnosis User's Manual* for more information.

Layout-Aware Diagnosis Flow with DFM

Opens an existing Tessent Diagnosis tool-compatible Layout Database and if this database contains imported DFM information, the layout-aware diagnosis results are annotated with DFM hit results.

For more information, see "Diagnosis for Design for Manufacturability Analysis" in the *Tessent Diagnosis User's Manual*.

Tessent CellModelGen Bridge and Open, and Cell Neighborhood Defect Extraction Flow

You use Tessent Shell in conjunction with Tessent CellModelGen to extract interconnect bridges and opens as well as cell neighborhood defects, and create UDFMs. These UDFMs are subsequently used in ATPG for critical area based bridge and open, and cell neighborhood pattern generation. This process uses a LDB, which is a prerequisite to this flow. For complete information, refer to "Bridge and Open, and Cell Neighborhood Defects UDFM Creation" in the *Tessent Scan and ATPG User's Manual*, and "Merge Task" in the *Tessent CellModelGen Tool Reference* manual.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***layout_database_name***

  A required string that specifies the name of the existing Layout Database directory created with the create_layout command. For hierarchical layout-aware diagnosis, the LDB must be an instance-aware core-level LDB.

  You can use the tab completion method to list possible files for ***layout_database_name*** that exist in the current directory. For example, specifying "open_layout mem" and pressing the Tab key lists the files with names starting with mem in the current directory.

Layout-Aware Diagnosis Flow

- -VERIfy

  An optional switch that specifies to perform verification if the current design has not been validated against the current Layout Database. Using this option ensures that in subsequent runs the layout database can be opened for the current design without having to re-run verification.

  At the time of opening the layout, Tessent Diagnosis checks whether the current design has been verified against the current Layout Database. If the current design has not been validated, it performs verification. During verification, layout rule violation information is added to the layout database.

  For each flat model used during diagnosis, you must first perform the verification step. You can do this with the open_layout -verify command. With subsequent diagnosis runs using a verified flat model/LDB combination, you can skip the verification step by just specifying open_layout. If you need to verify many flat models against the same LDB, you can use open_layout -verify to verify them in parallel in separate diagnosis sessions.

  Layout verification rules can be divided into two broad categories: those that check the consistency of LEF/DEF files themselves, and those that match design information with the layout. Only the second type of layout verification rules are re-evaluated when a layout database is verified against a new design. Because the LEF/DEF layout information has not changed, the first type of rules do not need to be re-run.

- -EXTRA_LAYout_hierarchy *extra_levels_of_hierarchy_in_layout*

  An optional switch and string pair that removes extra levels of layout hierarchy such that the design hierarchy and layout hierarchy match. You can only specify this switch once, otherwise the tool issues an error and exits.

  You use this switch if your design is a subset of a larger layout hierarchy. You can specify multiple levels of hierarchy in one string with no spaces.

  This switch can only be used in conjunction with the -verify switch.

- -EXTRA_DESign_hierarchy *extra_levels_of_hierarchy_in_design*

  An optional switch and string pair removes extra levels of design hierarchy such that the layout hierarchy and design hierarchy match. You can only specify this switch once, otherwise the tool issues an error and exits.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You use this switch if your layout is a subset of a larger design hierarchy. You can specify multiple levels of hierarchy in one string with no spaces.

This switch can only be used in conjunction with the -verify switch.

- -THreshold *percentage_match*

  An optional switch and integer pair specifies the minimum percentage match threshold required to successfully complete a layout verification of the LEF/DEF files, and subsequent verification of the layout database. The default threshold value is 85 percent.

- -chip_design_name *top_design_name*

  An optional switch that specifies a top-level design whose core instance information is included in the specified instance-aware core-level LDB. This option only pertains to hierarchical layout-aware diagnosis and is required when you have a core that is instantiated in more than one design.

  A chip-mapped core-level LDB can store information on instances of this core in multiple designs. This switch specifies which top-level design in which the core level LDB is included when performing hierarchical layout-aware diagnosis. This enables Tessent Diagnosis to report the correct chip-level coordinates for suspects reported in core-level diagnosis. If this switch is not used, the tool only reports core-level coordinates for suspects reported in the core-level diagnosis.

  The tool issues an error if the hierarchical database and the layout database use different units.

Examples

Example 1

In the following example, Tessent Diagnosis loads the existing Layout Database (*design.layout*) into memory:

```
open_layout design.layout
```

Example 2

The following example illustrates using the -VERIfy switch with the open_layout command for two designs: DesignA.flat and DesignB.flat. Except for different pin constraints, the two designs are essentially identical. Because a high percentage of the logic overlaps, both designs use the same Layout Database for layout-aware diagnosis.

To perform layout-aware diagnosis on these two designs using the same Layout Database, you do the following:

1. Create and verify the Layout Database, in this example DesignA_layout_database, with *DesignA.flat* using the create_layout command.
2. Invoke Tessent Diagnosis in scan diagnosis mode and read in the *DesignB.flat* flat model with the read_flat_model command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

3. Using the open_layout command and -VERIFy switch, open the DesignA_layout_database with the *DesignB.flat model* as in the following:

   open_layout -verify DesignA_layout_database

   This verifies *DesignB.flat* against the newly-created DesignA_layout_database.

4. Proceed with layout-aware diagnosis with *DesignB.flat*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

open_pattern_set

Context: patterns -ijtag

Mode: analysis

Opens an empty named pattern set and makes it ready to be populated with the specified PDL commands.

Usage

open_pattern_set ***pattern_set_name*** [-timeplate *timeplate_name* | -tester_period *tester_period*]
[-tck_ratio { 1 | 2 | 4 | 8 | 16 | … | 1024 }]
[-no_initial_ireset | -explicit_ireset | -ireset_in_test_setup_proc]
[-scan_out_strobe_point {before_rising_edge_of_tck | before_falling_edge_of_tck | auto}]
[-ignore_compliance_values] [-replace] [-usage unspecified | ssn]

Description

Opens an empty named pattern set and makes it ready to be populated with the specified PDL commands.

A populated pattern set is closed using the close_pattern_set command. An existing pattern set cannot be overwritten unless it is first deleted or the -replace option is used.

PDL commands can only be specified when a pattern set is open with the exception of the iReset and iCall commands, which can also be specified within the test_setup and test_end procedures in the patterns -scan context.

You use the -timeplate switch to specify the timeplate for this pattern set. Specifying a different timeplate is rarely needed as the default timeplate is adequate for standard IJTAG operations. Tessent Shell automatically defines the appropriate timeplate based on the specified off value for TCK and the TCK ratio, TCK period and tester period. The only time you need a different timeplate is in the rare situation you want to define clock waveforms with more than two edges.

By default, the tool automatically chooses tck_ratio such that the product of tester period and TCK ratio is at least as long as the TCK period specified with the "set_ijtag_retargeting_options -tck_period" command (the default TCK period is 100 ns). If you have synchronous clocks (add_clocks commands without the -period option), specify tester_period to match your system clock requirements and specify the TCK period with the IJTAG retargeting option such that TCK toggles at a rate that is not higher than it was designed for.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note: When you do not specify a timeplate, the tck_period, tck_ratio and tester_period are calculated as follows:

Table 5-28. Calculating tck_period, tck_ratio and tester_period

tester_period	tck_ratio	Behavior
Not specified	Not specified	tester_period = tck_period tck_period = value of set_ijtag_retargeting_options -tck_period (default 100 ns) tck_ratio = 1
Not specified	Specified	tester_period = tck_period / tck_ratio tck_period = value of set_ijtag_retargeting_options -tck_period (default 100 ns) tck_ratio = as specified

Specified	Not specified	tester_period = as specified
		if tester_period > value of set_ijtag_retargeting_options -tck_period (default 100 ns):
		tck_period = tester_period
		tck_ratio = 1
		else:
		tck_period = tester_period * tck_ratio
		tck_ratio = value of set_ijtag_retargeting_options -tck_period (default 100 ns) / tester_period (rounded up to the next valid power of 2)

Specified	Specified	tester_period = as specified tck_period = tester_period * tck_ratio tck_ratio = as specified

If all your system clocks are asynchronous (add_clocks commands with the -period option), set the -tck_ratio to 1 and specify -tester_period to match your TCK period requirements.

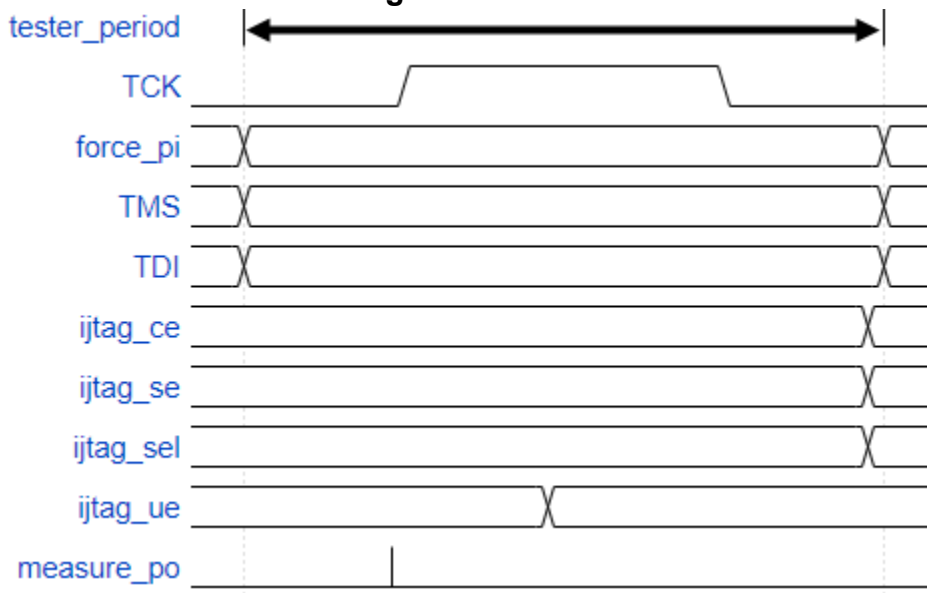
With a custom timeplate, you can specify different timing for the measurements and forces of ScanIn, ScanOut and TCK; however, the measurement time for ScanOut must be before the rising of TCK and after the falling edge of TCK when the off value for TCK is 1. For more information about timeplates, refer to "Timeplate Definition" in the *Tessent Shell User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 5-12. Timeplate for TCK Off State = 0 and TCK Ratio = 1

```
timing_variables =
   tester_period = 10;
   strobe_1 = (0.24 * tester_period);
   t_time = (0.25 * tester_period);
   t_width = (0.5 * tester_period);
   f_time_1 = (0.49 * tester_period);
   f_time_2 = (0.99 * tester_period);
end;

timeplate tessent_ijtag =
   force_pi 0;
   measure_po strobe_1;
   pulse_clock t_time t_width;
   force ijtag_ue f_time_1;
   force ijtag_ce f_time_2;
   force ijtag_se f_time_2;
   force ijtag_sel f_time_2;
   period tester_period;
end;
```

Figure 5-13. Scan Frame Timing for TCK Off State = 0 and TCK Ratio = 1

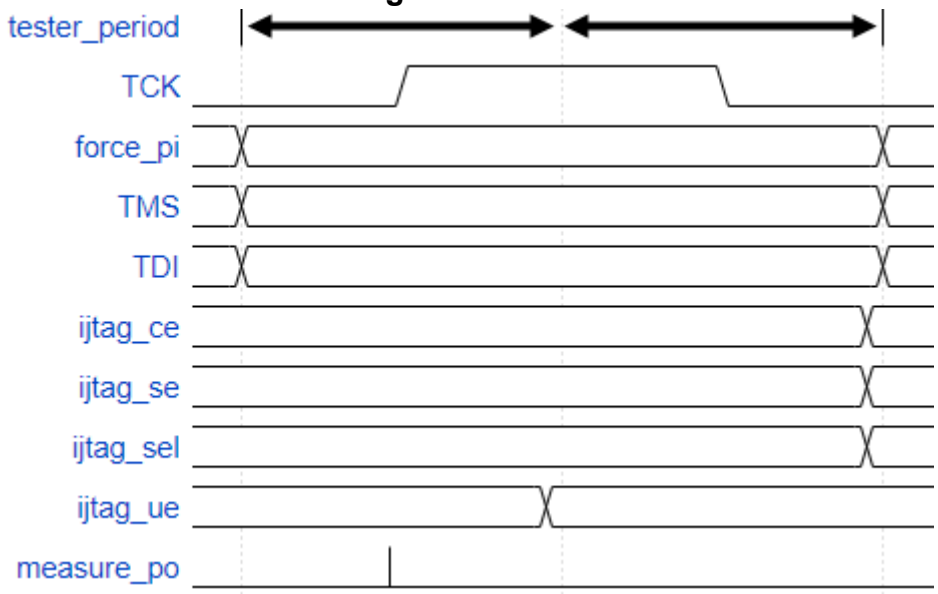


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 5-14. Timeplate for TCK Off State = 0 and TCK Ratio = 2

```
timing_variables =
   tester_period = 10;
   strobe_1 = (0.48 * tester_period);
   f_time = (0.5 * tester_period)
   t_time = (0.25 * tester_period);
   t_width = (0.5 * tester_period);
   f_time_2 = (0.99 * tester_period);
end;

timeplate tessent_ijtag =
   force_pi 0;
   measure_po strobe_1;
   force TCK f_time;
   pulse_clock t_time t_width;
   force ijtag_ce f_time_2;
   force ijtag_se f_time_2;
   force ijtag_sel f_time_2;
   force ijtag_ue f_time_2;
   period tester_period;
end;
```

Figure 5-15. Scan Frame Timing for TCK Off State = 0 and TCK Ratio = 2

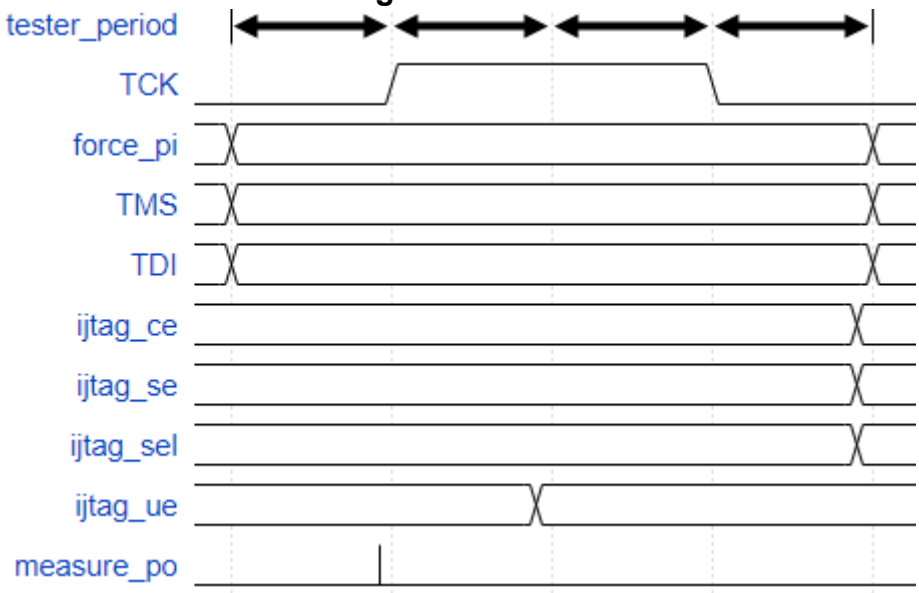


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 5-16. Timeplate for TCK Off State = 0 and TCK Ratio = 4

```
timing_variables =
   tester_period = 10;
   strobe_1 = (0.96 * tester_period);
   t_time = (0.25 * tester_period);
   t_width = (0.5 * tester_period);
   f_time = (0.99 * tester_period);
end;

timeplate tessent_ijtag =
   force_pi 0;
   measure_po strobe_1;
   force TCK 0;
   pulse_clock t_time t_width;
   force ijtag_ce f_time;
   force ijtag_se f_time;
   force ijtag_sel f_time;
   force ijtag_ue f_time;
   period tester_period;
end;
```

Figure 5-17. Scan Frame Timing for TCK Off State = 0 and TCK Ratio = 4

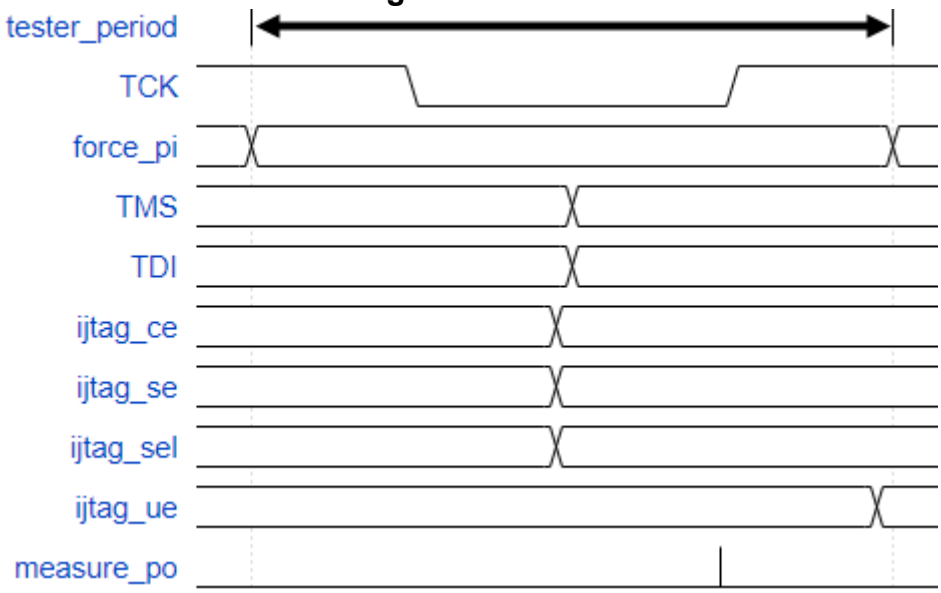


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 5-18. Timeplate for TCK Off State = 1 and TCK Ratio = 1

```
timing_variables =
   tester_period = 10;
   strobe_1 = (0.74 * tester_period);
   t_time = (0.25 * tester_period);
   t_width = (0.5 * tester_period);
   f_time_1 = (0.49 * tester_period);
   f_time_2 = (0.5  * tester_period);
   f_time_3 = (0.99 * tester_period);
end;

timeplate tessent_ijtag =
   force_pi 0;
   measure_po strobe_1;
   pulse_clock t_time t_width;
   force ijtag_ce f_time_1;
   force ijtag_se f_time_1;
   force ijtag_sel f_time_1;
   force TDI f_time_2;
   force TMS f_time_2;
   force ijtag_ue f_time_3;
   period tester_period;
end;
```

Figure 5-19. Scan Frame Timing for TCK Off State = 1 and TCK Ratio = 1

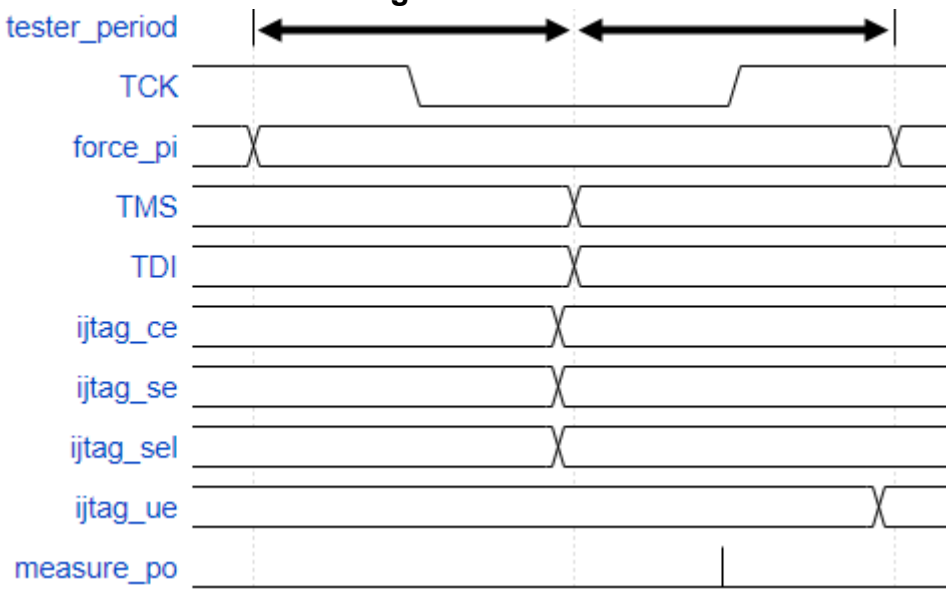


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 5-20. Timeplate for TCK Off State = 1 and TCK Ratio = 2

```
timing_variables =
   tester_period = 10;
   strobe_1 = (0.48 * tester_period);
   f_time = (0.5 * tester_period);
   t_time = (0.25 * tester_period);
   t_width = (0.5 * tester_period);
   f_time_2 = (0.99 * tester_period);
end;

timeplate tessent_ijtag =
   force_pi 0 ;
   measure_po strobe_1;
   force TCK f_time;
   pulse_clock t_time t_width;
   force ijtag_ce f_time_2;
   force ijtag_se f_time_2;
   force ijtag_sel f_time_2;
   force ijtag_ue f_time_2;
   period tester_period;
end;
```

Figure 5-21. Scan Frame Timing for TCK Off State = 1 and TCK Ratio = 2

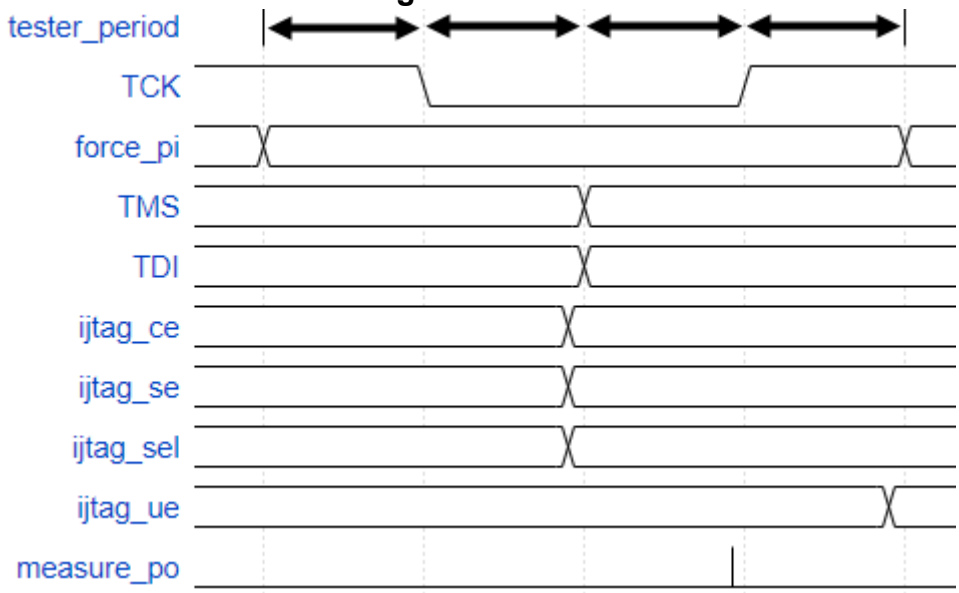


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 5-22. Timeplate for TCK Off State = 1 and TCK Ratio = 4

```
timing_variables =
   tester_period = 10;
   strobe_1 = (0.96 * tester_period);
   t_time = (0.25 * tester_period);
   t_width = (0.5 * tester_period);
   f_time = (0.99 * tester_period);
end;

timeplate tessent_ijtag =
   force_pi 0;
   measure_po strobe_1;
   force TCK 0;
   pulse_clock t_time t_width;
   force ijtag_ce f_time;
   force ijtag_se f_time;
   force ijtag_sel f_time;
   force ijtag_ue f_time;
   period tester_period;
end;
```

Figure 5-23. Scan Frame Timing for TCK Off State = 1 and TCK Ratio = 4



Arguments

- ***pattern_set_name***

  A required string that defines the name of the pattern set. This name uniquely identifies the pattern set for later use with write_patterns. This name also specifies the name of the iProc, if the pattern set is written out in -pdl format.

  Note

  The pattern set names “test_setup” and “test_end” are not permitted.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -timeplate *timeplate_name*

  An optional switch and string pair that specifies the name of a valid timeplate, defined in a procedure file reference with the set_procfile command. This option is mutually exclusive with the -tester_period option.

  Note

  When the opened pattern set has a TCK ratio of one, the tool checks the provided timeplate for IJTAG compatibility. It does not perform this check when the TCK ratio is greater than one. Instead, the tool automatically ensures that the generated TCK cycle is valid by placing the force and strobe events at valid positions.

- -tester_period *tester_period*

  An optional switch and value pair that specifies the tester period when the built-in timeplate, shown in Figure 5-12 through Figure 5-23, is used. This option is mutually exclusive with the -timeplate option. If you specify the tester period, Tessent Shell creates a new timeplate based on the TCK ratio, TCK period (defined with the "set_ijtag_retargeting_options -tck_period" command) and tester period.

  If you do not specify the -tester_period or -timeplate option, Tessent Shell creates a new timeplate based on the TCK period and TCK ratio. The tester period is chosen such that the TCK period equals the product of tester period and TCK ratio.

- -no_initial_ireset

  An optional switch to create a pattern set to run after another one while keeping power and clocks active between them. You may need to create a pattern set of this type if you need to insert some ATE actions in the middle of a test, such as changing a voltage. The tool exports these types of patterns together when writing out the Verilog testbench and as separate STIL or WGL files when exporting the pattern files to use on the ATE. Use special care when running these patterns on the ATE because the lack of a reset at the beginning of the pattern makes it impossible to quit in the middle of the pattern and resume from the start when the capture memory is full. You can only do data logging on such patterns in one pass and to the extent of the size of the capture memory on the ATE equipment.

  The -no_initial_ireset option cannot be used on the first open_pattern_set command of a session. When used on a subsequent open_pattern_set command, the end ICL state of the previous pattern_set is used as the beginning ICL state for the current pattern set.

- -explicit_ireset

  An optional switch that enables delay of the first iReset of a pattern set until the first action that requires the IJTAG network to be in a defined state. In some situations, it is desirable to apply certain actions to the IJTAG network before the first iReset happens, for example, by means of the commands iForcePort and iComparePort. The -explicit_ireset switch suppresses the implied iReset that otherwise would be added to the beginning of the pattern set by the open_pattern_set command. Prior to the first explicit usage of the command iReset in the pattern set, the -explicit_ireset switch prevents the usage of all commands that require the IJTAG network to be in a defined state, which are iRead, iWrite, iRunLoop,

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iClock, iClockOverride, iMerge, iTake, iRelease, create_icl_verification_patterns and import_patterns_from_svf. Other commands, like iForcePort and iComparePort, can be used before the first iReset.

- -ireset_in_test_setup_proc

  An optional switch that instructs the IJTAG retargeter to assume that the test_setup procedure contains the appropriate commands to reset the IJTAG network. When this switch is used, subsequent PDL retargeting assumes that the registers with a ResetValue are in their reset state, the SIBs are in their initial state and so on. You are responsible for the reset of the IJTAG network in the test_setup procedure. If there is a TAP controller, you are also responsible for the required transition from the RESET state to the IDLE state in the test_setup procedure. The -ireset_in_test_setup_proc switch can only be used if there is a test_setup procedure in the procfile. The -ireset_in_test_setup_proc and -no_initial_ireset switches are mutually exclusive.

- -tck_ratio { 1 | 2 | 4 | 8 | 16 | … | 1024 }

  An optional switch and literal pair that specifies the ratio between the required TCK period and the tester period this pattern set uses. The tester period is the value you specify with the -tester_period option when using the built-in timeplate or the period entry of the timeplate that the -timeplate option references. Specify TCK period with the "set_ijtag_retargeting_options -tck_period" command.

  If the TCK ratio switch is not specified and the tester period switch (see above) is specified, the TCK ratio is computed such that the product of tester period and TCK ratio is at least as long as the TCK period. If both the TCK ratio switch and the tester period switch are not specified, the TCK ratio of "set_ijtag_retargeting_options -tck_ratio" is used and the tester period is computed such that the product of tester period and TCK ratio is at least as long as the TCK period.

  The value of tck_ratio might depend on the maximum tck clock frequency that can be applied to the circuit. See the "IJTAG Network Performance Optimization" section in the *Tessent IJTAG User's Manual* showing how to maximize the frequency of the IJTAG network test clock.

- -usage <u>unspecified</u> | ssn

  An optional switch and literal that specifies whether the pattern set is for use with SSN continuity patterns. Using this optional switch with the value "ssn" is a prerequisite for creating SSN continuity test patterns (using the command create_ssn_continuity_patterns). It triggers the creation of the appropriate timeplates for the SSN continuity patterns and adjusts parameters such as the TCK ratio and the tester period.

- -ignore_compliance_values

  An optional switch that instructs the tool to ignore the compliance_value attributes of the ICL ports of the top module. When this switch is omitted, the retargeter applies the compliance values to the ICL ports right at the beginning of the pattern set.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -scan_out_strobe_point before_rising_edge_of_tck | before_falling_edge_of_tck | auto

  An optional switch and literal pair that specifies the primary output strobe point for the IJTAG pattern set. The setting of “auto” resolves to the “before_falling_edge_of_tck” setting for the scan-out strobe point when the design meets all the following conditions:

  - The design level is chip
  - Only TAP client ScanInterfaces are active at the top level

  Otherwise, the “auto” setting resolves to “before_rising_edge_of_tck”. You can override the “auto” setting behavior if you specify either the “before_rising_edge_of tck” or “before_falling_edge_of_tck” settings for the scan-out strobe point.

  If you do not specify the -scan_out_strobe_point switch, the scan-out strobe point defaults to the value set by set_ijtag_retargeting_options -default_scan_out_strobe_point.

  Specify the "before_rising_edge_of_tck" setting if your design level is chip, but you are using a TAP controller that is not compliant and updates the TDO on the rising edge of TCK. Specify the "before_falling_edge_of_tck" setting if your design level is sub_block or physical_block, and your network is not fully compliant and updates the data of the ScanOutPort on the rising edge of TCK.

- -replace

  An optional switch that specifies to replace an existing pattern set with the same name.

Return Values

None

Examples

The following example opens a pattern set called pat1 using the built-in timeplate described in Figure 5-12. The tester_period is over-written to 50 ns. The same pattern set is later replaced by one where the timeplate TP1 is used. TP1 is a timeplate defined inside a procfile and referenced by the set_procfile command in setup mode.

```
set_ijtag_retargeting_options -tck_period 50ns
open_pattern_set pat1 -tester_period 50ns
iCall myProc
close_pattern_set

open_pattern_set pat1 -timeplate TP1 -replace
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

open_tsdb

Context: all contexts

Mode: all modes

Makes the contents of the TSDB directory visible to the tool. The specified TSDB output directory is automatically opened.

Usage

open_tsdb ***directory_path_list*** [-silent]

Description

Makes the contents of the TSDB directory visible to the tool. The specified TSDB output directory is automatically opened.

Many actions are taken on the opened TSDBs at various steps in the design flow. Some of those actions are described here:

- When you run the set_current_design command, the tool automatically reads in the most recent *module_name.format*_interface file using the read_verilog or read_vhdl -interface_only option if the corresponding *module_name* does not already exist in memory. This enables you to only load the modules associated with the current design and have the interface view of all child physical or sub-blocks loaded automatically.

  Note

  The set_current_design command does not automatically load a child block that has the interface_has_external_dependencies property set to 1 in the *.tsdb_info* file of the child block. The interface_has_external_dependencies property is set to "1" if the port list of the interface has external dependencies such as typedef packages or System Verilog interfaces. If you want to load that child block, use the "read_design *child_block* -view interface" command.

- All dft_inserted_designs and instrument subdirectories are automatically searched, and ICL files corresponding to the design modules are automatically loaded after design elaboration.
- The is_physical_module attribute (see the "Built-In Attributes" table in "Module" on page 3654) is automatically set on all modules that are found in a TSDB and that were declared as physical blocks using the set_design_level command when they were processed by the tool.

Arguments

- ***directory_path_list***

  A required value containing an absolute or relative path to a TSDB directory, or a Tcl list containing one or many absolute or relative paths to TSDB directories. The tool verifies that each specified directory path is an existing TSDB directory and that only one copy of an

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

instrument and dft_inserted_designs directories exists for any given *module/design_id* combination in all opened TSDBs; if this is not the case and you get this error, you must close the redundant TSDB directories using the close_tsdb command. See also "Tessent Shell Data Base (TSDB)" on page 4807.

- -silent

  An optional switch that suppresses the warning that is normally generated if the specified *directory_path* is already opened.

Examples

The following example opens three TSDB directories with one command invocation.

```
setenv tsdb_home '/home/projects/chipa/tsdbs'
open_tsdb [list ${tsdb_home}/corea.tsdb \
            ${tsdb_home}/coreb.tsdb \
            ${tsdb_home}/corec.tsdb ]
get_tsdb_list
```

```
/home/projects/chipa/tsdbs/corea.tsdb /home/projects/chipa/tsdbs/coreb.t
sdb /home/projects/chipa/tsdbs/corec.tsdb
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

open_visualizer

Context: unspecified, all contexts

Mode: all modes

Opens the Tessent Visualizer window.

Usage

open_visualizer
[-connection_status |
[-tcp_port *port*]
[-authorization_code [none]]
[-server_only |
[-import_configuration *config_file*]
[-label *session_label*]
[-display *window*...]]]

Description

Starts the Tessent Visualizer client application and server, if one is not already running. This command can also be used to open the server only, to enable remote client connections.

You can also use the following commands to open Tessent Visualizer implicitly:

- add_schematic_objects
- add_schematic_path
- add_schematic_connections
- add_schematic_callout
- analyze_drc_violation
- display_diagnosis_report

Displayed data persists until the Tessent Visualizer server is closed. Close the server by exiting the Tessent Shell or issuing the “close_visualizer -server” command on the host machine.

Arguments

- -connection_status

  An optional switch that prints the Tessent Visualizer connection status to the standard output. Tessent Visualizer is not started. See Example 2 for more information.

- -tcp_port *port*

  An optional switch and integer that sets the TCP port number for connecting to the Tessent Visualizer session from a remote machine.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -authorization_code [none]

  An optional switch that sets a random authorization code for the Tessent Visualizer session. This enables you to connect to your private session remotely, using **$TESSENT_HOME/bin/tessent -visualizer -authorization_code** on the Linux command line.

  Provide the optional literal "none" to disable the authorization mechanism and enable remote connections without any authorization code.

  Caution

  You should only disable the authorization mechanism in secure environments.

- -server_only

  An optional switch that opens Tessent Visualizer as a server only. You can then connect to the session from the local machine or a remote machine.

- -import_configuration *config_file*

  An optional switch and string that specifies a configuration file to be read for Tessent Visualizer. If the configuration file cannot be read, is invalid, or is from an incompatible version of Tessent Visualizer, an error message is displayed.

  This switch cannot be used with -server_only. See "Saving and Restoring the Session State" in the for more information on using configuration files.

- -label *session_label*

  An optional switch and string that sets a prefix to be added to Visualizer window titles.

- -display *window...*

  An optional switch and repeatable literal that specifies which windows display when Tessent Visualizer opens. The Instance Browser, DRC Browser, and Transcript windows open by default if no previous user configuration has been restored and the -display argument is not specified.

  If the specified window is already open, it is activated.

  Valid literals include the following:

  hierarchical_schematic — opens the Hierarchical Schematic window.

  instance_browser — opens the Instance Browser window.

  library_browser— opens the Cell Library Browser window.

  flat_schematic — opens the Flat Schematic window.

  wave_generator — opens the Wave Generator window.

  pin_data — opens the Pin Data window.

  icl_schematic — opens the ICL Schematic window.

  icl_instance_browser — opens the ICL Instance Browser window.

  ijtag_network — opens the IJTAG Network window.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iproc_viewer — opens the iProc Viewer window.

drc_browser — opens the DRC Browser window.

hdl_viewer — opens the Text/HDL Viewer window.

diagnosis_reports — opens the Diagnosis Report Viewer window.

transcript — opens the Transcript window.

search_instances — opens the Search - Instances window.

search_pins — opens the Search - Pins window.

search_gate_pins — opens the Search - Gate Pins window.

search_icl_instances — opens the Search - ICL Instances window.

search_icl_pins — opens the Search - ICL Pins window.

Note

Windows related to ICL and IJTAG require the “Tessent IJTAG” license.

Examples

Example 1

This example opens Tessent Visualizer in server mode on port 59000:

open_visualizer -tcp_port 59000 -server_only

```
//  command: open_visualizer -tcp_port 59000 -server_only
//   Server: started
//   - Hostname:            ies-dft-plp-01
//   - TCP/IP port:         59000
//   - Authentication code: 068434
//   Client: not connected
//  Note: To connect Tessent Visualizer client to the server, issue the
following command in a Linux console:
//  ----------------------------------------------------------
    /<path>/tessent -visualizer -server ies-dft-plp-01 \
    -tcp_port 59000 -authorization_code 068434
//  ----------------------------------------------------------
```

Example 2

This example prints a message to the standard output describing the Tessent Visualizer connection status:

open_visualizer -connection_status

The following information is printed if the Tessent Visualizer server has not yet been started:

```
//  command: open_visualizer -connection_status
//   Server: not started
//   Client: not connected
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following information is printed if the Tessent Visualizer server has been started but no client has connected:

```
//  command: open_visualizer -connection_status
//   Server: started
//   - Hostname:            ies-dft-plp-01
//   - TCP/IP port:         59000
//   - Authentication code: 068434
//   Client: not connected
//  Note: To connect Tessent Visualizer client to the server, issue the
following command in a Linux console:
//  ----------------------------------------------------------
    /<path>/tessent -visualizer -server ies-dft-plp-01 \
    -tcp_port 59000 -authorization_code 068434
//  ----------------------------------------------------------
```

The following Linux shell command opens a Tessent Visualizer client on this server:

$TESSENT_HOME/bin/tessent -visualizer -server ies-dft-plp-01 -tcp_port 59000 -authorization_code 068434

The following information is printed if the **open_visualizer -connection_status** command is invoked again after the client is connected:

```
//  command: open_visualizer -connection_status
//   Server: started
//   - Hostname:            ies-dft-plp-01
//   - TCP/IP port:         59000
//   - Authentication code: -
//   Client: connected
```

Note

The authentication code is reported as a hyphen (-), indicating that the code has been consumed and no more clients may connect to the server. Only one client may connect to a server at a time.

More information on the invocation and usage of Tessent Visualizer can be found in the *Tessent Shell User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

order_patterns

Context: dft -edt, patterns -scan

Mode: analysis

Reorders the internal test pattern set as specified.

Usage

order_patterns [*number_of_passes* | Auto *percentage*]
[-Init_order Normal | REverse | RAndom] [-Simulate [-PATterns_per_pass *integer*]]
[-REMOVE_ineffective_patterns ON | OFF]
[{-CRITICAL_AREA |
{{-CAPture_power [ST_Peak | ST_Average | WSA_Peak | WSA_Average] |
-SHIft_power {LOAD | RESPonse}}
[-INSTance *instance_name* | -MODUle *module_name*]
[-ASCending | -DEScending]}}]

Description

Reorders the internal test pattern set as specified.

Test patterns can be rearranged to detect the most faults first, which is useful for:

- Truncating the test pattern set in order to fit it in the ATE memory.
- Detecting failing chips as early as possible during testing.

Multiple load patterns are ranked according to the average number of faults detected per scan load, ensuring that on average the patterns that detect the most faults per scan load are first.

The order_patterns command is not supported when multiple detection is enabled. This command also does not support MacroTest or parametric patterns.

This is a multiprocessing command. For more information about multiprocessing, refer to "Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual.*

Note

Using the order_patterns command turns off pattern classification until you issue a "set_pattern_classification On" command.

Arguments

- *number_of_passes*

  An optional integer that specifies the number of passes to make through the data when reordering the pattern set. By default, one pass is made.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Auto *percentage*

  An optional switch and integer pair that reorders the pattern set iteratively until further reordering produces negligible improvement in the number of patterns needed to achieve a given fault coverage benchmark. The tool determines the given fault coverage benchmark by subtracting percentage from the total fault coverage (FC) achieved using all the patterns in the internal pattern set. The percentage argument must be a non-negative number less than or equal to FC.

  Reordering stops when either of the following conditions exist:

  - The tool completes a maximum of 100 iterations.
  - The number of patterns that is needed to achieve the given fault coverage benchmark decreases by less than 1 percent from one iteration to the next. This stop point can be stated mathematically as:

  (PFCb)previous_pass - (PFCb)current_pass < (PFCb)previous_pass x .01

  where: PFCb = number of patterns needed to achieve the fault coverage benchmark

  FCb = FC - percentage = fault coverage benchmark

  FC = total fault coverage achieved using all the patterns in the pattern set

- -Init_order Normal| REverse | Random

  An optional switch and literal pair that specifies the initial pattern ordering to use as a starting point for pattern reordering. The literal choices are as follows:

  Normal — A literal that specifies the order of the current internal pattern set as the initial pattern order.

  REverse — A literal that specifies to use the reverse order of the current internal pattern set as the initial pattern order.

  Random — A literal that specifies to randomize the order of the current internal pattern set, and use the random order as the initial pattern order.

- -Simulate

  An optional switch that fault simulates the test patterns after reordering.

- -PATterns_per_pass *integer*

  An optional switch and positive integer pair, used with the -simulate option, that specifies how many patterns to create and simulate in each pass. The integer value must be between 1 and 64. By default, the tool creates and simulates 64 patterns in each pass. When the value is 1, the tool generates and simulates every pattern, and so on.

  A smaller value is appropriate if the expected pattern count is small. A typical scenario for using a small integer value is IDDQ pattern ordering.

- -REMOVE_ineffective_patterns ON | OFF

  Optional switch that enables or disables the removal of ineffective patterns, which do not detect additional faults during the order pattern analysis. If you set this switch to ON, you

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

combine the benefits of the order_patterns and compress_patterns commands. By default, the switch is set to OFF so that no patterns are removed after the order_patterns command runs.

- -CRITICAL_AREA

  Optional switch that orders test patterns based on total critical area (TCA) covered by each pattern, from highest to lowest coverage. It is strongly recommended to first enable TCA coverage reporting with the "set_critical_area_options -reporting on" command so you can see TCA coverage values during fault simulation. When you order external pattern sets, they must first be simulated using a UDFM fault model.

  Note

  Use of this switch requires a Tessent Automotive-Grade ATPG license. If no Automotive-Grade ATPG license is available, the run terminates with an error message.

- -CAPture_power [ST_Peak | ST_Average | WSA_Peak | WSA_Average]

  Optional switch and literal pair that orders test patterns based on switching activity during capture. Literal options include:

  ST_Peak — Peak of state transitions. Default.

  ST_Average — Average of state transitions.

  WSA_Peak — Peak of weighted switching activity.

  WSA_Average — Average of weighted switching activity.

  Weighted Switching Activity is defined in the *Tessent Scan and ATPG User's Manual*.

- -SHIft_power {LOAD | RESPonse}

  Optional switch and literal that orders test patterns based on switching activity during shift. Literal options include:

  LOAD — based test pattern loading.

  RESPonse — based on circuit response.

- -INSTance *instance_pathname*

  An optional switch and string that orders test patterns based on switching activity associated with a specified instance.

  The -instance switch can only be used when -capture_power or -shift_power is specified. The instance_pathname can only specify a single instance.

- -MODule *module_name*

  An optional switch and string that orders test patterns based on switching activity associated with a specified module. The module_pathname may include any number of asterisk (*) and question mark (?) wildcard characters in a name.

  The -module switch can only be used when -capture_power or -shift_power is specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -ASCending | -DEScending

  Optional switch that orders test patterns with respect to the amount of switching activity. By default test patterns are ordered in an ascending order.

Examples

Example 1

Total fault coverage is 98 percent, and there are 2,500 patterns in memory. The following example reorders patterns in the internal pattern set until the difference between the number of patterns that is needed to achieve 93 percent (98 - 5) fault coverage in the current pass and the previous pass is less than 1 percent of the number of patterns in the previous pass:

order_patterns auto 5

Example 2

The following example creates test patterns, and then reorders them based on the average weighted switching activity in capture cycles:

create_patterns
order_patterns -capture_power wsa_avg ascending

```
//  Warning: Pattern classification is disabled.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

printenv

Context: unspecified, all contexts

Mode: all modes

Prints out the values of the Linux environment variables.

Usage

printenv

Description

Prints out the values of the Linux environment variables.

The printenv command enables the Linux printenv command to be available as a common DFT command, for convenience in displaying Linux environment variables. Linux environment variables are automatically available as variable references within the tool.

Arguments

None

Examples

The following example prints out the values of the Linux variables in the environment:

```
printenv
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

process_atpg_failures

Context: patterns -silicon_insight

Mode: setup, analysis

Interprets ATPG failure log files generated on ATE. Produces flop failure files for a given characterization and debug package (CDP) and test.

Usage

process_atpg_failures ***cdp***
test_name[{-interpret_fails *pattern_flog_pairs_list* | -flog_directory *flog_directory*}
[-results_directory *results_directory*] [-diagnosis_result_type *diagnosis_result_type*]
[-learn_uncovered_1hot_patterns
[-maximum_1hot_patterns_to_learn *maximum_1hot_patterns_to_learn*]] |
[-report_ate_data_collection_flow]

Arguments

- ***cdp***

  A required string that specifies the path to the CDP.

- ***test_name***

  A required string that specifies the name of the test in the CDP.

- -interpret_fails *pattern_flog_pairs_list*

  An optional switch and list of string pairs that specifies pattern file names and corresponding failure log pairs to be interpreted.

- -flog_directory *flog_directory*

  An optional switch and string that specifies a directory that contains flog directories generated by the collect_cdp_test_failure_logs command to be interpreted. By default, this command reads these files from the current working directory.

- -results_directory *results_directory*

  An optional switch and string that specifies a root directory for the hierarchy in which to write failing flops files and failing nets files for CDP tests. This hierarchy parallels the directory hierarchy used in the failure log (flog) directory. By default, this command uses the directory specified by the -flog_directory argument as the root directory for the hierarchy in which it writes these files. If that directory is not specified, it defaults to the current working directory.

- -diagnosis_result_type *diagnosis_result_type*

  An optional switch and literal pair that specifies the type of diagnosis to perform on the failure files. The following are valid values for <*diagnosis_result_type*>:

  - failing_pseudo_flops (default)
  - failing_flops

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

  - failing_nets
  - failing_nets_iterative_diagnosis (not supported in hybrid mode)

  Refer to diagnosis_result (a characterization debug option) in the *Tessent SiliconInsight User's Manual for Tessent Shell* for more details about these diagnosis result types.

- -learn_uncovered_1hot_patterns

  An optional switch that automatically learns uncovered patterns and generates additional one-hot pattern files for them, saving the generated files in the CDP. These patterns can then be used by subsequent collections of failures on the tester. You can specify this switch only when you have specified the -interpret_fails argument.

- -maximum_1hot_patterns_to_learn *maximum_1hot_patterns_to_learn*

  An optional switch and integer that specifies the maximum number of one-hot patterns to generate during learn mode.

- -report_ate_data_collection_flow

  An optional switch that reports the exact patterns for a given CDP test that must be run on the tester while saving the failure logs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

process_dft_specification

Context: dft (with no sub context)

Mode: setup, analysis

Validates and processes the content contained in a DftSpecification wrapper.

Usage

process_dft_specification [*id*] [-no_insertion [-design_name *design_name*]] [-validate_only] [-transcript_insertion_commands]

Description

Validates and processes the content contained in a DftSpecification wrapper.

If you specify the -validate_only option, this command only validates the DftSpecification. If you specify the -no_insertion option, the command validates and generates the DFT components. When used without either of these two options, the command validates the specification, generates the DFT components such as the RTL and the ICL description, and inserts them into the design. The insertion can be done at the netlist or the RTL level depending on the set_context -no_rtl versus -rtl option that was specified when setting the context.

If the DftSpecification being processed has a HostScanInterface/Interface consisting of only ports on the current design and the -no_insertion option is not specified, the add_icl_scan_interfaces and the set_icl_scan_interface_ports commands are issued during the insertion stage such that if you perform ICL extraction after the insertion, the ScanInterface is named using the *id* of the HostScanInterface contained in the DftSpecification.

The data created by this command is stored inside the directory returned by the get_tsdb_output_directory command, which is *./tsdb_outdir* (default value) unless you have changed it with the set_tsdb_output_directory command.

See the "Tessent Shell Data Base (TSDB)" section for more information about the output directory structure.

Note

For compatibility with the Verilog and VHDL languages, specified escaped identifiers in the DftSpecification can be terminated with either a whitespace or a backslash. This implies that backslashes are not permitted within Verilog escaped identifiers and whitespaces are not permitted within VHDL escaped identifiers even though the languages support it.

If tcd_scan is available for certain Tessent IP containing scan segments, the tool generates and writes CTL scan information in the TCD. See "Default Generation of CTL Files" in the "Scan" on page 4794 section.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

process_dft_specification.post_insertion Procedure

When performing the insertion (specifically when neither the -validate_only nor -no_insertion switch is specified), the modified design is written out inside the "tsdb_*output_directory/dft_inserted_designs*" directory. If you need to perform custom editing between the insertion of the DFT components and the saving of the design, you can define a Tcl proc with the name "process_dft_specification.post_insertion" and it is automatically called after all DFT components have been inserted and before the write_design command is invoked.

To develop this type of procedure, run all of your custom editing commands interactively in Tessent Shell in the insertion mode. When you are finished, create the Tcl proc with those commands in it; the Tcl proc is then called when rerunning process_dft_specification. If you need to instantiate new modules inside the design as part of this Tcl proc, you must read them in using read_verilog or read_vhdl in setup mode prior to invoking process_dft_specification.

The process_dft_specification.post_insertion procedure must have two arguments as shown in the following example. The first argument holds the configuration element object that is currently being processed. The second argument holds a list of option value pairs corresponding to the options that were used when the process_dft_specification command was called.

```
proc process_dft_specification.post_insertion { root_wrapper args } {
  # "$root_wrapper" is the config_element object being processed.
  # "get_config_value $root_wrapper -id <0>" returns the design_name
  # "get_config_value $root_wrapper -id <1>" returns the id
  # "$args" is a list of option value pairs that reflects the options used
  # when the process_dft_specification command was called.
  array set extra_args $args

  # $extra_args(-instrument_prefix) returns the value of that option
}
```

Licensing

Table 8-4 lists the available contexts and sub-contexts supported by each license. The -license argument of the set_context command can be used to specify a particular license to use. By default, Tessent Shell acquires the least expensive licenses available that provide the requested functionalities. For example, if you insert multiple Tessent instruments, the tool requests all respective product licenses. If you insert only an IJTAG network with your current DFT Specification, the tool will select any one of the following licenses: IJTAG, MemoryBIST-LV, MemoryBIST-LVNS, MemoryBIST-TS, LogicBIST, TestKompress, or MissionMode, unless you had set one of these product licenses prior to process_dft_specification with "set_context dft -license" or as implied by your settings of set_dft_specification_requirements.

Note

License names for MemoryBIST and BoundaryScan are accepted without the -LV or -TS suffix and default to the LV version of the license.

The TESSENT_LICENSE_ORDER environment variable modifies the priority and order of the available licenses. Any available licenses not explicitly listed in the value of the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TESSENT_LICENSE_ORDER environment variable are appended to the list in their original order.

You can refer to the set_license_queue_timeout and get_license_queue_timeout commands for more information about managing the license queuing feature.

Arguments

- *id*

  An optional string that specifies the DftSpecification wrapper that is to be processed. As described in the DftSpecification wrapper syntax, the wrapper is identified by the two strings design_name and id. If you only have one DftSpecification wrapper present in memory that matches DftSpecification(design_name,*), you can omit the id from the process_dft_specification command. However, if several DftSpecification wrappers matching DftSpecification(design_name,*) exist in memory, you must specify the appropriate id when running the process_dft_specification command. Refer to the following -design_name option to understand how the design_name value is obtained.

- -no_insertion

  An optional switch that specifies to not perform the insertion of the DFT components into the design. When this option is specified, no design is needed because no insertion is performed. Some validations, such as verifying that the specified parent_instance properties in the Sib, ScanMux and Tdr wrappers actually exist in the design, are only performed when this switch is omitted. If you run the process_dft_specification command without this option and the set_current_design command has not yet been issued, an error is generated.

- -design_name *design_name*

  An optional switch and string pair that specifies the design_name value used to identify the DftSpecification wrapper when the -no_insertion option is specified. The -design_name value defaults to the current design if it has been set with the set_current_design command. If the current design has not been set, it is extracted from the DftSpecification wrapper existing in memory.

  When the id is specified, if exactly one wrapper matches “DftSpecification (*,<id>)” in memory, the design_name is extracted from the design_name parameter of the DftSpecification wrapper; if more than one wrapper matches, the -design_name must be specified. When the id is not specified, if exactly one wrapper matches “DftSpecification (*,*)” in memory, the design_name is extracted from the design_name parameter of the DftSpecification wrapper; if more than one wrapper matches, the -design_name must be specified.

- -validate_only

  An optional switch that specifies to only perform the validation of the DftSpecification and to skip DFT element generation and insertion. Using the -no_insertion switch with the -validate_only option disables a few checks such as verifying that the specified parent_instance properties in the Sib, ScanMux and Tdr wrappers actually exist in the design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -transcript_insertion_commands

  An optional switch that specifies to enable the transcription of the insertion commands as they are being performed. This option can generate a large number of transcripted commands but you may find it useful to see the instantiation and connection commands created by the process_dft_specification command.

Examples

The following example reads a design and a DftSpecification file, and then processes the DftSpecification. The context is changed to the patterns -ijtag context to extract the ICL of the design and generate simulation patterns. This example uses the DftSpecification shown in the example of the ScanMux wrapper documentation. Notice the use of the iNote "tessent_pragma verilog_insert" to inject native Verilog commands into the testbench at specific times to directly monitor the states of internal signals.

```
#-------------- Insert Ijtag Network --------------------
set_context dft -no_rtl
set_tsdb_output_directory scanMux_ex1_outdir
read_verilog ../data/MyCore.v
set_current_design MyCore
read_config_data ../data/ScanMux_ex1.dft_spec
process_dft_specification

#---------------- Extract ICL ---------------------------
set_system_mode setup
set_context patterns -ijtag
set_system_mode analysis
write_icl -out ScanMux_ex1.icl

#---------------- create Simulation Patterns  -----------# This method is left here as a way to
perform manual testbench generation but the new
# recommended method is shown below and makes use of the create_dft_specification and
# process_patterns_specification commands.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set dut "MyCore_ScanMux_ex1_v_ctl.MyCore_inst"
open_pattern_set p1
iNote "iWrite tessent_rtl_tdr_orange_inst.tdr 0b0 and end_in_pause"
iWrite tessent_rtl_tdr_orange_inst.tdr 0b0
iApply -end_in_pause
iNote "iRead tessent_rtl_tdr_orange_inst.tdr 0b0"
iNote "iWrite tessent_rtl_tdr_orange_inst.tdr 0b1 and end_in_pause"
iRead tessent_rtl_tdr_orange_inst.tdr 0b0
iWrite tessent_rtl_tdr_orange_inst.tdr 0b1
iApply -end_in_pause
iNote "tessent_pragma verilog_insert \$display(\$time, \"ns:\
  measured tessent_rtl_tdr_orange_inst.ijtag_sel = %b\",\
 ${dut}.tessent_rtl_tdr_orange_inst.ijtag_sel);"
iNote "tessent_pragma verilog_insert \$display(\$time, \"ns:\
 measured tessent_rtl_tdr_green_inst.ijtag_sel = %b\",\
 ${dut}.tessent_rtl_tdr_green_inst.ijtag_sel);"
iNote "tessent_pragma verilog_insert \$display(\$time, \"ns:\
 measured tessent_rtl_tdr_red_inst.ijtag_sel = %b\",\
 ${dut}.tessent_rtl_tdr_red_inst.ijtag_sel);"
close_pattern_set

write_patterns ScanMux_ex1.v -ver
```

The manual method for creating simulation shown above is no longer needed nor recommended because you can now run the create_patterns_specification and process_patterns_specification commands to automate the process. Immediately after running extract_icl, you can issue the commands below and the signoff simulation patterns are created including the ICLNetworkVerify wrapper to automatically verify the entire IJTAG network. The run_testbench_simulations and the check_testbench_simulations command automatically manage the invocation of the external simulator.

```
set spec [create_patterns_specification]
process_patterns_specification
run_testbench_simulations
check_testbench_simulations
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

process_patterns_specification

Context: patterns

Mode: all modes

Validates and processes the content of the PatternsSpecification(*design_name*, *design_id*, *pattern_id*) wrapper.

Usage

process_patterns_specification [*pattern_id*] [-validate_only] [-unprocessed_only]
[-config_objects *config_object*]
[-import_command_options | -ignore_command_options_inconsistencies]

Description

Validates and processes the content of the PatternsSpecification(*design_name*, *design_id*, *pattern_id*) wrapper.

You typically need an elaborated ICL design with the Verilog interface view to run this command; the only exception is when you are using the "BSDL-Only Flow," which is described in this section. The *design_name* identifier is extracted from the top ICL module. The *design_id* is extracted from the tessent_design_id attribute, described in icl_module, which is stored in the top ICL module when the top ICL module is created using ICL extraction.

You use the Verilog interface view to make sure the pattern has access to all the ports of the design. If you only have the ICL view the patterns are limited to only the port defined in the ICL. For boundary Scan patterns or if you have AdvancedOptions/ConstantPortSettings to define, you must read in the top level Verilog module such that all ports of the top module are available. You can use read_verilog or read_vhdl -interface_only to load the interfaced top module. With the Tessent Shell Data Base (TSDB) open, a more automated and easier way to do that is to use the read_design -view interface command. It loads the interface and the ICL view directly from the opened TSDB.

If the top ICL module was not generated with ICL extraction, it may be missing the tessent_design_id attribute. In this case, the *design_id* is taken from the one specified with the set_context -design_id option; the *pattern_id* can be specified as the first argument to the command. When unspecified, it defaults to the *pattern_id* of the PatternsSpecification wrapper found in memory when there is only one match for PatternsSpecification(*design_name*, *design_id*, *).

If the PatternsSpecification wrapper was created manually and only includes Patterns wrappers containing only TestStep/BoundaryScan wrappers with the bsdl_file property pointing to a BSDL file, you are using what is referred to as the "BSDL only flow". In this case, having an elaborated ICL module is not needed. If you only have one PatternsSpecification wrapper in memory, it is automatically used. If you have many PatternsSpecification wrappers in memory, you must point to one of them using the -config_objects option.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

By default and when generating patterns, any signal that is classified as unused for the pattern being written is removed from the pattern file, reducing the size of the pattern file or of the Verilog testbench. To disable this behavior, use the include_all_power_pins and include_only_used_pins properties in the Patterns/AdvancedOptions wrapper of the PatternsSpecification.

The simulation of ICL network patterns supports the graybox for the sub-physical blocks listed in the LowerPhysicalBlocksInstances wrapper within the SimulationOptions wrapper.

If you specify the -validate_only option, the command only validates the Patterns specification and does not generate any patterns. Without the -validate_only option, the validation is done for each Patterns wrapper and only the error-free ones are processed.

The data created by this command is stored inside the patterns directory located in the directory returned by the get_tsdb_output_directory command, which is *./tsdb_outdir* (default value) unless you have changed it with the set_tsdb_output_directory command. See the "Tessent Shell Data Base (TSDB)" section for more information about the TSDB output directory structure. The patterns files are the Verilog testbenches when the usage property inside the PatternsSpecification is "signoff". It is the patterns in the selected format specified by the manufacturing_patterns_formats property when the usage is set to manufacturing_test. In either case, the *.pdl* file is always generated, which shows the retargeted PDL that was used to create the patterns file.

The process_patterns_specification command is destructive as follows:

- All input constraints are deleted.
- The value of attribute loadboard_pull_resistor is reset on all ports.
- All loadboard loopback pairs are deleted.
- All added clocks are deleted.

The tool performs this destruction as follows:

- At the very start of process_patterns_specification so that only the settings in the PatternsSpecification affect pattern generation. For example, if your dofile defined loopback pairs with the add_loadboard_loopback_pairs command, the tool deletes these; however, the tool uses loopback pairs if you have defined such in the Loopbacks wrapper.
- At the very end of process_patterns_specification so that the settings in the PatternsSpecification do not affect subsequent calls to the write_patterns command.

Controlling Ports Used in Pattern and Testbench Generation

You can control the full list of ports that the tool uses to generate the patterns and simulation testbench by using the "set_write_patterns_options -existing_used_ports" command and switch prior to issuing the process_patterns_specification command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The syntax is as follows:

set_write_patterns_options -existing_used_ports *ports*

Use this command with caution as the list of *ports* is considered an absolute collection. If the list of *ports* is missing ports that should be toggling or sampled by the pattern, the ports should be part of the *ports* collection. As a best practice, you should remove the unwanted ports instead of trying to select those ports that should appear in the pattern/testbench. The following example demonstrates how to do this in a safe manner:

```
set allExistingPorts [get_write_patterns_options -existing_used_ports]
set ports2Remove [get_ports [list sel* vdd*]]
set portsNeeded [remove_from_collection $allExistingPorts $ports2Remove]
set_write_patterns_options -existing_used_ports $portsNeeded
process_patterns_specification
```

In the example, the tool removes the ports with names beginning with “sel” and “vdd” from the default list of *ports* seen after design elaboration.

If the tool removes ports needed for the generated pattern/testbench to be effective, the result can be deceiving. For example, removing the “TDI” and “TDO” port on a chip with a TAP ends up in a pattern/testbench that still has the annotations but no actual data is scanned in/out, and there is no comparison to expected values.

When you specify a collection of *ports*, the process_patterns_specification command issues the following note:

```
//  Note: The following ports were selected using
//  'set_write_patterns_options -existing_used_ports <ports>':
//          { <port_name>... }.
//       Only those selected ports appear in the generated pattern files,
//       which may render them unusable if the above list is lacking
//       ports that are needed.
//       The preceding selection does not have an effect on boundary scan
//       patterns that use LoadBoardInfo.
```

Licensing

By default, Tessent Shell acquires the least expensive licenses available that provide the requested functionalities. For example, if you generate patterns for multiple Tessent instruments, the tool requests product licenses for the instruments tested.

Arguments

- *pattern_id*

  Specifies the string corresponding to the *pattern_id* of the PatternsSpecification(*design_name*,*design_id*,*pattern_id*) wrapper to process.

- -validate_only

  An optional switch that specifies to only validate the patterns specification and to not generate the patterns.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -unprocessed_only

  An optional switch that instructs the process_patterns_specification command to consider only those Patterns wrappers that have not been processed by a previous call to process_patterns_specification in the same Tessent Shell session. Once all the Patterns have been processed, the next call to process_patterns_specification with the -unprocessed_only switch does not process anything. It is possible to reset this behavior by doing the following:

  - By reading in a new PatternsSpecification or re-creating one.
  - By resetting the attribute is_unprocessed on selected Patterns wrappers. Refer to the reset_attribute_value command.

- -config_objects *config_objects*

  An optional switch and value pair that specifies a collection or list of config element names found inside one or many PatternsSpecification(*design_name*,*design_id*,*pattern_id*) wrappers. When the config objects are elements below a Patterns wrapper, the parent Patterns wrapper is used instead. When a config element is a PatternsSpecification or wrapper, all Patterns wrappers below it are used. The tool issues an error when the collection or list includes an InSystemTest wrapper.

  This option is used in the BSDL only flow to reference a PatternsSpecification wrapper. It can also be used to replace the -select option. You can use the full power of the get_config_elements command to find the Patterns wrapper you want to process and pass the collection to the process_patterns_specification command with this option.

- -import_command_options

  An optional switch, mutually exclusive with -ignore_command_options_inconsistencies, that instructs the process_patterns_specification command to delete the contents of the PatternsSpecification/IjtagRetargetingOptions prior to running, and populate the wrapper with the current retargeting options set by prior set_ijtag_retargeting_options commands.

- -ignore_command_options_inconsistencies

  An optional switch, mutually exclusive with -import_command_options, that instructs the process_patterns_specification command to ignore any retargeting options set by a set_ijtag_retargeting_options command prior to running, and only process the retargeting options specified in the PatternsSpecification/IjtagRetargetingOptions wrapper.

Examples

The following example uses the process_patterns_specification command to validate only the patterns with the name “MemoryBist_*” in the PatternsSpecification that were automatically generated by the create_patterns_specification command and modified using the configuration data editing and introspection commands listed in Table 13-1.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set spec [create_patterns_specification]
set mbist_pats [get_config_el Patterns(MemoryBist_*) -in $spec -silent]
foreach_in_col mbist_pat $mbist_pats {
  set first_test_step [get_config_value TestStep<0> -in $mbist_pat -obj]
  read_config_data -before $first_test_step -from_string {
    SimulationOptions {
      SimulationMacros {
        DEBUG;
      }
    }
  }
}
process_patterns_specification -validate_only -config_objects $mbist_pats
set spec [create_patterns_specification]
set mbist_pats [get_config_el Patterns(MemoryBist_*) -in $spec -silent]
foreach_in_col mbist_pat $mbist_pats {
   set first_test_step [get_config_value TestStep<0> -in $mbist_pat -obj]
   read_config_data -before $first_test_step -from_string {
      SimulationOptions {
        SimulationMacros {
          DEBUG;
        }
      }
    }
}
process_patterns_specification -validate_only -config_objects $mbist_pats
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

process_top_module_connections

Context: dft

Mode: setup, analysis

Validates and processes the content contained in a TopModuleConnections wrapper to automatically create the DFT ports on tiled modules and connect them at the chip level.

Usage

process_top_module_connections
[-validate_only]
[-transcript_insertion_commands]
[-extension_suffix *extension_suffix*]

Description

Use this command for your tiled design flow. In a tiled design, all ports are pre-created on each tile and pre-connected in the chip-level netlist. This enables you to have a detailed floorplan with pin placement before implementing the tiles. You can insert the DFT ports using the same process that you use for the functional ports, but if this process is manual, you may want to automate the DFT port creation using the process_top_module_connections command.

See the TopModuleConnections wrapper for details on how to describe the ports you want to create on each tile and specify the connections between them.

Arguments

- -validate_only

  An optional switch to specify that you only want to validate the TopModuleConnections connection, not process it, even if there are no errors.

- -transcript_insertion_commands

  An optional switch to specify that you want to record all insertion commands used to create and connect the ports to the log file.

- -extension_suffix *extension_suffix*

  An optional switch and string pair to specify the extension suffix the tool uses when writing out the modified design files. The default extension suffix is "_with_dft_ports". Each modified file is written to the same directory as the source file, with the extension suffix appended to the original file extension.

Examples

Example 1

The following dofile creates the DFT ports on a set of tile modules and then makes the connections between them at the chip level. Initially, the tool only reads the chip-level module. The configuration file processed by the process_top_module_connections command includes the filepaths to the tile modules.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_context dft -rtl -design_id rtl0

read_verilog ../../design/chip/chip.v
set_current_design chip

read_config_data top.top_level_ports_and_connections

process_top_module_connections
```

Below is an example that shows the TopModuleConnections specification used to create the DFT ports in the chip module used in Example 4 of the OCC reference design.

The top-level wrapper is called TopModuleConnections, and its ID must match the name of the current_design. The referenced instance name supplied with the Instance() wrappers must match instances below the current design, and their module name must match the module name specified with the ChildBlock wrapper.

Note

There are some abbreviations (using ...), but there is enough to illustrate the concept. See TopModuleConnections for more details on the syntax.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
TopModuleConnections(chip) {
  ChildBlock(c1) {
    file_name            : ../../design/c1/c1.v;
    language_format      : 2001;
    input_ports          : bscan_select,
                           bscan_force_disable,
                           bscan_select_jtag_input,
                           bscan_select_jtag_output,
                           bscan_clock,
                           bscan_capture_en,
                           bscan_shift_en,
                           bscan_update_en,
                           bscan_scan_in,

                           ijtag_tck,
                           ijtag_reset,
                           ijtag_ce,
                           ijtag_se,
                           ijtag_ue,
                           ijtag_sel,
                           ijtag_si,

                           pdgB_bisr_mem_chain_select,
                           pdgB_bisr_shift_en,
                           pdgB_bisr_clk,
                           pdgB_bisr_mem_disable,
                           pdgB_bisr_reset,
                           pdgB_bisr_si,

                           ssn_bus_clock,
                           ssn_bus_data_in[3:0];

    output_ports         : bscan_scan_out,

                           ijtag_so,

                           pdgB_bisr_so,

                           ssn_bus_data_out[3:0];
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
    Instance(c1_i1) {
      input_port_sources : c2_i1/r1_bscan_to_select,
                           c2_i1/r1_bscan_to_force_disable,
                           c2_i1/r1_bscan_to_select_jtag_input,
                           c2_i1/r1_bscan_to_select_jtag_output,
                           c2_i1/r1_bscan_to_clock,
                           c2_i1/r1_bscan_to_capture_en,
                           c2_i1/r1_bscan_to_shift_en,
                           c2_i1/r1_bscan_to_update_en,
                           c2_i1/r1_bscan_to_scan_in,

                           c2_i1/r1_ijtag_to_tck,
                           c2_i1/r1_ijtag_to_reset,
                           c2_i1/r1_ijtag_to_ce,
                           c2_i1/r1_ijtag_to_se,
                           c2_i1/r1_ijtag_to_ue,
                           c2_i1/r1_ijtag_to_sel,
                           c2_i1/r1_ijtag_to_si,
                           c2_i1/pdgB_r1_bisr_to_mem_chain_select,
                           c2_i1/pdgB_r1_bisr_to_shift_en,
                           c2_i1/pdgB_r1_bisr_to_clk,
                           c2_i1/pdgB_r1_bisr_to_mem_disable,
                           c2_i1/pdgB_r1_bisr_to_reset,
                           c2_i1/pdgB_r1_bisr_to_si,

                           c2_i1/r1_ssn_to_bus_clock,
                           c2_i1/r1_ssn_to_bus_data_in[3:0];
    }
    Instance(c1_i2) {
      input_port_sources : c2_i1/r2_bscan_to_select,
                           ...;
    }
    Instance(c1_i3) {
      input_port_sources : c2_i2/r1_bscan_to_select,
                           ...;
    }
    Instance(c1_i4) {
      input_port_sources : c2_i2/r2_bscan_to_select,
                           ...;
    }
  }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
ChildBlock(c2) {
  file_name            : ../../design/c2/c2.v;
  language_format      : 2001;
  input_ports          : bscan_select,
                         ...;
  output_ports         : bscan_scan_out,

                         r1_bscan_to_select,
                         r1_bscan_to_force_disable,
                         r1_bscan_to_select_jtag_input,
                         r1_bscan_to_select_jtag_output,
                         r1_bscan_to_clock,
                         r1_bscan_to_capture_en,
                         r1_bscan_to_shift_en,
                         r1_bscan_to_update_en,
                         r1_bscan_to_scan_in,

                         r2_bscan_to_select,
                         r2_bscan_to_force_disable,
                         r2_bscan_to_select_jtag_input,
                         r2_bscan_to_select_jtag_output,
                         r2_bscan_to_clock,
                         r2_bscan_to_capture_en,
                         r2_bscan_to_shift_en,
                         r2_bscan_to_update_en,
                         r2_bscan_to_scan_in,

                         ijtag_so,

                         r1_ijtag_to_tck,
                         r1_ijtag_to_reset,
                         r1_ijtag_to_ce,
                         r1_ijtag_to_se,
                         r1_ijtag_to_ue,
                         r1_ijtag_to_sel,
                         r1_ijtag_to_si,

                         r2_ijtag_to_tck,
                         r2_ijtag_to_reset,
                         r2_ijtag_to_ce,
                         r2_ijtag_to_se,
                         r2_ijtag_to_ue,
                         r2_ijtag_to_sel,
                         r2_ijtag_to_si,

                         pdgA_bisr_so,

                         pdgB_bisr_so,

                         pdgB_r1_bisr_to_mem_chain_select,
                         pdgB_r1_bisr_to_shift_en,
                         pdgB_r1_bisr_to_clk,
                         pdgB_r1_bisr_to_mem_disable,
                         pdgB_r1_bisr_to_reset,
                         pdgB_r1_bisr_to_si,
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
                              pdgB_r2_bisr_to_mem_chain_select,
                              pdgB_r2_bisr_to_shift_en,
                              pdgB_r2_bisr_to_clk,
                              pdgB_r2_bisr_to_mem_disable,
                              pdgB_r2_bisr_to_reset,
                              pdgB_r2_bisr_to_si,

                              ssn_bus_data_out[3:0],

                              r1_ssn_to_bus_clock,
                              r1_ssn_to_bus_data_in[3:0],

                              r2_ssn_to_bus_clock,
                              r2_ssn_to_bus_data_in[3:0];

      Instance(c2_i1) {
        input_port_sources : c3_i1/left_bscan_to_select,
                             ...;
      }
      Instance(c2_i2) {
        input_port_sources : c3_i1/right_bscan_to_select,
                             ...;
      }
    }
    ChildBlock(c3) {
      file_name            : ../../design/c3/c3.v;
      ...
    }
  }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 6
Command Dictionary (Ra-Report_i)

The Command Dictionary describes the application commands for Tessent Shell. For quick reference, the commands appear alphabetically with each beginning on a separate page.

The command usage line syntax conventions are common to all products documented in this manual. See "Conventions for Command Line Syntax" on page 97.

Command Descriptions 1470
range_collection 1477
read_bsdl 1479
read_cell_library 1480
read_config_data 1481
read_core_descriptions 1484
read_cpf 1486
read_def 1489
read_design 1492
read_failures 1499
read_fault_sites 1507
read_faults 1519
read_flat_model 1530
read_icl 1534
read_lef 1536
read_liberty 1537
read_modelfile 1539
read_patterns 1543
read_procfile 1555
read_sdc 1559
read_sdf 1566
read_upf 1569
read_verilog 1571
read_vhdl 1578
register_attribute 1582
register_callback 1586
register_drc 1596
register_drc_class 1631
register_static_dft_signal_names 1632
register_tcl_command 1636
remove_from_collection 1650
rename_instance 1652
rename_module 1654

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

rename_net . . . 1655
rename_pin . . . 1657
rename_port . . . 1658
replace_instances . . . 1660
report_aborted_faults . . . 1662
report_atpg_constraints . . . 1665
report_atpg_functions . . . 1666
report_atpg_timing . . . 1667
report_attributes . . . 1672
report_bisr_repair_register_icl_instances . . . 1680
report_bist_capture_ranges . . . 1681
report_black_boxes . . . 1682
report_boundary_scan_port_options . . . 1684
report_bsdl . . . 1686
report_bsdl_extraction_options . . . 1687
report_bsdl_extraction_user_instructions . . . 1688
report_bus_data . . . 1689
report_bypass_chains . . . 1692
report_capture_procedures . . . 1695
report_capture_timing_options . . . 1701
report_cell_constraints . . . 1703
report_cell_models . . . 1706
report_chain_masks . . . 1708
report_clock_controller_pins . . . 1709
report_clock_controls . . . 1710
report_clock_domains . . . 1712
report_clock_gating . . . 1717
report_clocks . . . 1719
report_compactor_connections . . . 1724
report_config_data . . . 1727
report_config_messages . . . 1732
report_config_syntax . . . 1735
report_context . . . 1739
report_control_signals . . . 1741
report_core_descriptions . . . 1743
report_core_instance_parameters . . . 1746
report_core_instances . . . 1748
report_core_parameters . . . 1752
report_current_design . . . 1754
report_default_ssn_datapath_configuration . . . 1756
report_design_partitions . . . 1757
report_design_sources . . . 1758
report_dfm_rules . . . 1760
report_dft_cell_selection_mapping . . . 1762
report_dft_clock_enables . . . 1763

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_dft_clock_muxes 1765
report_dft_control_points 1766
report_dft_modal_connections 1768
report_dft_signal_names 1772
report_dft_signals 1773
report_diagnosis 1775
report_drc_rules 1776
report_edt_abort_analysis 1784
report_edt_blocks 1799
report_edt_configurations 1800
report_edt_finder 1806
report_edt_instances 1811
report_edt_lockup_cells 1814
report_edt_pins 1819
report_environment 1822
report_external_simulator 1824
report_failures 1825
report_false_paths 1831
report_fault_sites 1840
report_faults 1848
report_feedback_paths 1865
report_flattener_rules 1867
report_gates 1869
report_gate_to_rtl_mapping 1894
report_graybox_statistics 1895
report_icl_extraction_options 1897
report_icl_modules 1898
report_icl_network 1902
report_iclock 1904
report_id_stamp 1906
report_iddq_exceptions 1908
report_ijtag_instances 1909
report_ijtag_logical_connections 1911
report_ijtag_retargeting_options 1913
report_input_constraints 1915
report_insert_test_logic_options 1918
report_instance_polygons 1920
report_iprocs 1921

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Command Descriptions

This section describes the Tessent Shell commands alphabetically, Ra through Report_i. The beginning of each command description shows the contexts and modes that support the command and provides the following information:

Context: Lists each context and sub-context that supports the command. “All contexts” means that the command is supported in all available contexts. “Unspecified” means that you do not have to set a context to use the command.

Mode: Lists each mode that supports the command. “All modes” means that the command is supported in all available modes. In some cases, a particular mode supports the command only within certain contexts, and this is noted in parentheses after the mode type.

Table 6-1. Commands Ra Through Report_i

Command	Description
range_collection	Returns one or more adjacent elements from a collection and creates a new collection.
read_bsdl	Reads a BSDL file for one or more die in a 2.5D design.
read_cell_library	Loads one or more cell libraries into the tool.
read_config_data	Reads the content of configuration data files or a string into the Tessent Shell environment.
read_core_descriptions	Reads the specified Tessent core description (TCD) files.
read_cpf	Loads a CPF (Common Power Format) file and removes any previously loaded power data.
read_def	Reads in the DEF file so that the tool can extract the coordinates of instances in the design and the dimensions of the design.
read_design	Reloads a design after DFT insertion to perform analysis or a future DFT insertion pass.
read_failures	Checks the specified failure file for correct syntax and semantics and optionally appends a failure file.
read_fault_sites	Loads the path or bridge definitions or user-defined fault models (UDFM) contained in the specified ASCII file into the current tool session.
read_faults	Updates the current fault list with the faults contained in the specified fault file or from the Tessent Shell Database (TSDB).
read_flat_model	Loads a flat model into the tool from a specified filename or from the Tessent Shell database (TSDB).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 6-1. Commands Ra Through Report_i (cont.)

Command	Description
read_icl	Reads ICL files into the internal ICL database.
read_lef	Reads in the LEF file so that the tool can extract the dimensions of the module.
read_liberty	A command to read a Liberty file to extract attribute information to be populated onto read_cell_library models or read_verilog modules.
read_modelfile	Initializes the specified RAM or ROM gate using the memory states contained in the named modelfile or deletes an earlier modelfile assignment.
read_patterns	Loads the external pattern set from the specified file or from the Tessent Shell database (TSDB).
read_procfile	Reads a test procedure file and merges the timing and named capture procedure data it contains with existing data loaded from previous test procedure files.
read_sdc	Reads false path and multicycle path information from a Synopsys Design Constraint (SDC) file.
read_sdf	Loads timing data from a Standard Delay Format (IEEE SDF 1497) file.
read_upf	Loads a UPF (Unified Power Format) file and removes any previously loaded power data.
read_verilog	Reads one or more Verilog files into the specified or default logical library.
read_vhdl	Reads one or more VHDL files into the specified or default logical library.
register_attribute	Registers a new user-defined attribute.
register_callback	Register a Tcl proc that is to automatically run before or after a built-in feature of Tessent Shell.
register_drc	A command used to register a custom DRC rule, and make it look and behave as a built-in DRC rule fully supported by the analyze_drc_violation command and the Tessent Visualizer GUI.
register_drc_class	Registers a new class of DRC rules such that a DRC of that class can be registered
register_static_dft_signal_names	Registers new static DFT signal names and their attributes.
register_tcl_command	Registers a Tcl command and makes it available in the global name space.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 6-1. Commands Ra Through Report_i (cont.)

Command	Description
remove_from_collection	Removes objects found in obj_spec from a collection, resulting in a new collection.
rename_instance	Renames the leaf name of an instance.
rename_module	Assigns a new name to an existing module.
rename_net	Renames the leaf name of a net.
rename_pin	Assigns a new name to an existing instance pin.
rename_port	Assigns a new name to an existing port.
replace_instances	Replaces the module definition of an instance with a new module definition.
report_aborted_faults	Displays information on some potentially testable faults that remain undetected (UD) after the ATPG process.
report_atpg_constraints	Displays all the current ATPG state restrictions and the pins on which they reside.
report_atpg_functions	Displays all the current ATPG function definitions.
report_atpg_timing	Reports the current ATPG timing setup or statistical analysis of delay data.
report_attributes	Creates a readable report for the set of attributes based on the usage options.
report_bisr_repair_register_icl_instances	Reports a list of BISR repair registers found in the design.
report_bist_capture_ranges	Displays a list of the currently defined BIST capture ranges.
report_black_boxes	Displays information on blackboxes and undefined models.
report_boundary_scan_port_options	Displays a table listing all of the ports with their inferred or specified cell_options.
report_bsdl	Writes a report of the BSDL data stored for the die in a 2.5D design.
report_bsdl_extraction_options	Reports all values for various BSDL attributes previously set with the set_bsdl_extraction_options command.
report_bsdl_extraction_user_instructions	Reports added user instructions.
report_bus_data	Displays the bus data information for either an individual bus gate or for the buses of a specific type.
report_bypass_chains	Displays the defined EDT bypass scan chain connections.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 6-1. Commands Ra Through Report_i (cont.)

Command	Description
report_capture_procedures	Displays the cyclized information for a specified named capture procedure.
report_capture_timing_options	Reports the capture timing options defined for a specified icl_module, module, or the current physical block.
report_cell_constraints	Displays a list of the constrained cells.
report_cell_models	Displays a list of either all cell models or the DFT library models associated with the specified cell type.
report_chain_masks	Reports the list of masked scan chains and their associated unload value.
report_clock_controller_pins	Reports the clock controller pins specified with the set_clock_controller_pins command.
report_clock_controls	Displays current configuration information for clock control definitions.
report_clock_domains	Displays all clocks and their corresponding clock domain numbers.
report_clock_gating	Reports specified clock gating instances or modules in the design.
report_clocks	Displays a list of user-defined clocks and SDC-defined clocks.
report_compactor_connections	Displays defined scan chain connections for each EDT scan channel output.
report_config_data	Reports the content of configuration wrappers.
report_config_messages	Reports messages associated to a configuration element. When the -hierarchical option is used, it also reports the messages associated to elements below the specified wrapper.
report_config_syntax	Reports the legal configuration syntax for a specified configuration object.
report_context	Reports the current context as specified by the set_context command and any inferred subcontexts.
report_control_signals	Displays the rules checking results for control signals consisting of clocks added with the add_clocks command and pins identified for gating scannable memory elements with test logic.
report_core_descriptions	Reports all core description files.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 6-1. Commands Ra Through Report_i (cont.)

Command	Description
report_core_instance_parameters	Reports the instrument configuration parameters and their values that were specified when the core instance was added with the add_core_instances commands -parameter_values switch.
report_core_instances	Reports the instance bindings created with the add_core_instances command, as well as the pattern count for patterns read in for each core.
report_core_parameters	Reports all core parameters and their values.
report_current_design	Generates a report listing design ports that will cause problems during the design flow.
report_default_ssn_datapath_configuration	Reports all override settings of an ICL network's default SSN datapath configuration.
report_design_partitions	Reports the partitions in a design.
report_design_sources	Reports the pathnames or file extensions previously specified with the set_design_sources command.
report_dfm_rules	Returns a tabular report of the DFM rules contained in the layout database.
report_dft_cell_selection_mapping	Returns a tabular report of library cell selections, along with their associated power domains.
report_dft_clock_enables	Reports the currently defined DFT clock enables.
report_dft_clock_muxes	Reports the currently defined DFT clock muxes that were previously added using the add_dft_clock_muxes command.
report_dft_control_points	Reports the currently defined DFT control points that were previously added using the add_dft_control_points command.
report_dft_modal_connections	A command to report the cumulative effect of the previously invoked add_dft_modal_connections and delete_dft_modal_connections commands.
report_dft_signal_names	Reports all registered DFT signal names and identifies their attributes as well as what was the source of registration.
report_dft_signals	Reports all the DFT signals added or to-be-created in the design.
report_diagnosis	Writes the scan diagnosis report to standard I/O.
report_drc_rules	Displays either a summary of DRC violations (fails) or violation occurrence messages.
report_edt_abort_analysis	Reports the results of EDT Aborted fault analysis.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 6-1. Commands Ra Through Report_i (cont.)

Command	Description
report_edt_blocks	Displays a list of the current user-defined EDT block names (also referred to as block tags) and indicates which one, if any, represents the current block.
report_edt_configurations	Displays configuration information for the current compression logic.
report_edt_finder	Reports information about the EDT logic contained in a design.
report_edt_instances	Displays the instance pathnames of the top-level EDT logic, decompressor, and compactor.
report_edt_lockup_cells	Displays a tabular report of the lockup cells in the EDT logic.
report_edt_pins	Displays the currently defined names and inversion status of EDT channel and control pins.
report_environment	Displays the current value(s) of many of the most common “set_...” commands.
report_external_simulator	Displays the shell command the tool uses to invoke an external simulator for verifying patterns.
report_failures	Simulates the current design and test patterns and reports the failures in a pattern-based format.
report_false_paths	Displays false paths previously defined with the add_false_paths command.
report_fault_sites	Depending on the fault model, displays path definitions, bridge entries, or UDFM entries from the internal list.
report_faults	Displays fault information from the current fault list.
report_feedback_paths	Displays a report of currently identified feedback paths.
report_flattener_rules	Displays either a summary of all the flattening rule violations or the data for a specific violation.
report_gates	Displays the netlist information and simulation results for the specified gates and the simulation results for the specified user-defined ATPG functions. This command requires a flattened netlist before use.
report_gate_to_rtl_mapping	Report rules for post-synthesis/post-layout to RTL instance name-mapping for DEF files.
report_graybox_statistics	Reports the statistics gathered by graybox analysis.
report_icl_extraction_options	Provides a human readable report of the ICL extraction options.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 6-1. Commands Ra Through Report_i (cont.)

Command	Description
report_icl_modules	Reports loaded and extracted ICL modules in either ICL syntax or human readable form. The output can be redirected to a file.
report_icl_network	Context: patterns -silicon_insight -ijtag
report_iclock	Reports the ICL ClockPort specified by the iClock command as well as their extracted sources and cumulative freqMultiplier and freqDivider values.
report_id_stamp	Displays the unique identifier that the tool assigns each internal pattern set.
report_iddq_exceptions	Displays current exceptions to Iddq restrictions.
report_ijtag_instances	Reports options of instances for which there is a matched ICL module.
report_ijtag_logical_connections	Reports the logical paths that exist within the current design between the specified source and destination pins/ports, as well as all connections from or to the specified pins/ports.
report_ijtag_retargeting_options	Reports the type and value of each option of the previous set_ijtag_retargeting_options command or their default values if the set_ijtag_retargeting_options command was not issued.
report_input_constraints	Displays the present constraint status of primary input pins.
report_insert_test_logic_options	Reports the prefix/infix values that are used when logic is created by the insert_test_logic command.
report_instance_polygons	Context: patterns -scan_diagnosis
report_iprocs	Displays a table listing of all iProcs and their related namespace iProcNameSpace for each ICL module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

range_collection

Context: unspecified, all contexts

Mode: all modes

Returns one or more adjacent elements from a collection and creates a new collection.

Usage

range_collection ***collection first last***

Description

Returns one or more adjacent elements from a collection and creates a new collection.

The returned collection consists of elements first through last, inclusive. If first is greater than last, this command returns an empty collection.

The base collection remains unchanged.

Arguments

- ***collection***

  A required string that specifies the collection to be searched.

- ***first***

  A required integer that specifies the first index into the collection. Permitted values are from 0 to *sizeof_collection* - 1.

- ***last***

  A required integer that specifies the ending index into the collection. Permitted values are from 0 to *sizeof_collection* - 1. You can also specify the value of the last argument relative to the end of the collection by using “end-<*integer*>”.

Examples

The following example returns a collection with the last two elements from $mycoll.

```
range_collection $mycoll end-1 end
```

Related Topics

add_to_collection

append_to_collection

compare_collections

copy_collection

filter_collection

foreach_in_collection

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_collection

remove_from_collection

sizeof_collection

sort_collection

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_bsdl

Reads a BSDL file for one or more die in a 2.5D design.

Usage

read_bsdl ***bsdl_file_path*** [-package_file_directories *package_file_directories*]
[-for_instances *for_instances*]

Description

Reads a BSDL file, checks it for correctness, and stores it in memory. The BSDL file is used by the tool for BSDL extraction and for multi-die interconnect pattern generation. If you read the BSDL for the current design, it is used by the create_patterns_specification command for BoundaryScan pattern generation.

Arguments

- ***bsdl_file_path***

  Required. The pathname of the BSDL file.

- -package_file_directories *package_file_directories*

  An optional switch and Tcl list of directories containing the package files you provide. The tool uses the descriptions of custom cells from the package files for the BoundaryScan extraction of the package. If a BSDL file uses package files, the tool must also use them to build a complete data set for BSDL extraction. By default, the tool searches the parent directory of the BSDL file for the package files.

- -for_instances *for_instances*

  An optional switch and Tcl collection or list of instance names. If it is not specified, the tool derives the module name from the BSDL entity and assumes that the BSDL description describes all instances of that module. If you load BSDL descriptions for the same module with and without the -for_instances switch, the tool reports an error.

Examples

The following example reads the BSDL data for one instance of a die in a 2.5D design.

```
read_bsdl tsdb_outdir/inst/diee.inst/
diee_base.bsdl -for_instances i1
report_bsdl
//
// // ------ -------- ----------------------------------------
// // Entity Instance BSDL File
// // ------ -------- ----------------------------------------
// // diee   i1       tsdb_outdir/inst/diee.inst/diee_base.bsdl
// //
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_cell_library

Context: unspecified, all contexts

Mode: setup

Loads one or more cell libraries into the tool.

Usage

read_cell_library ***lib_path*** ... [-force]

Description

Loads one or more cell libraries into the tool.

This command provides functionality equivalent to the -Library tool invocation switch.

You can issue this command multiple times. In case of parsing errors, the tool clears all the library data and returns to Setup mode. After the erroneous file has been corrected and reloaded, the tool reparses all files that were successfully parsed previously.

Arguments

- ***lib_path***

  A required and repeatable string that specifies the names of the files containing the Tessent Cell library descriptions for cell models in your design. You can use wildcards to specify multiple filenames.

- -force

  An optional switch to suppress the error message that normally prevents the accidental deletion of previously defined setup information. After you read in a library using this command, you can incrementally issue the command again to read more files. However, once the tool builds the design using set_current_design and you add any configuration information to it (for example, add_clocks or add_scan_chains), running the read_cell_library command causes the setup information to be discarded. In this case, to prevent configuration information from being discarded unintentionally, the tool requires that you run the command with the -force switch to indicate that it is acceptable to delete the setup data.

Examples

The following example loads two cell libraries:

```
read_cell_library lib1 lib2
```

Related Topics

delete_cell_library

set_cell_library_options

write_cell_library

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_config_data

Context: unspecified, dft, patterns

Mode: all

Reads the content of configuration data files or a string into the Tessent Shell environment.

Usage

read_config_data {*filename_list* | -from_string *string*}
[-in_wrapper *parent_wrapper*]
[-before *before* | -after *after* | -first | -last]
[-replace]

Description

Reads the contents of configuration data files or a string into the Tessent Shell environment.

The configuration data file format is currently used for two features but may be used to describe additional specifications later.

The first usage is to load a dft specification and a defaults specification containing the syntax described in the DftSpecification, and DefaultsSpecification/DftSpecification sections, which are used by the get_defaults_value and process_dft_specification commands.

The second usage is to load the test setup specifications described in "Prepare the Design Under Test" in the *Tessent SiliconInsight User's Manual for Tessent Shell* to enable Tessent SiliconInsight to communicate via Desktop, SimDUT, or offline ATE modes with the design under test (DUT). In this application, Tessent Shell uses the settings in the Tessent SiliconInsight setup specification to control bring-up and usage of the Tessent SiliconInsight session.

Arguments

- *filename_list*

  An optional string that specifies the name of a file or a Tcl list of filenames. The names can include wildcards (the * character) to permit matching multiple filenames with a single pattern.

- -from_string *string*

  An optional value pair that specifies a string to parse. The string can be contained in a Tcl variable or typed between braces ({}).

  You can also put the string between quotation marks ("”). For example:

  ```
  read_config_data -from_string " EDT {interface: $sib;} "
  ```

  This is useful when variables within the string must be evaluated.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -in_wrapper *parent_wrapper*

  An option and value pair that specifies that the read in configuration elements are to be loaded inside a specific wrapper. The *parent_wrapper* can be the complete name of a wrapper relative to the root of the partition or a collection containing a single wrapper element as returned by or "-object". When the -in_wrapper is not specified, the destination parent wrapper is assumed to be the root of the partition unless the -before or -after option is specified.

  If the specified root wrapper is a wrapper of type CSV, you can load a CSV file generated out of Excel™ where the line terminator is the carriage return instead of a semicolon.

- -before *before*

  An optional value pair that specifies a configuration element before which the new configuration elements are to be loaded. When the -before option is used with a collection containing a configuration element object, the -in_wrapper option cannot be used as the parent wrapper is extracted from the object.

  When before is a name, it must a name relative to the wrapper specified with the -in_wrapper option. It is a name relative to the root of the partition when the -in_wrapper option is not specified.

  The -before option is mutually exclusive to the -after, -first and -last options. When none of those four options is specified, -last is assumed.

- -after *after*

  An optional value pair that specifies a configuration element after which the new configuration elements are to be loaded. When the -after option is used with a collection containing a configuration element object, the -in_wrapper option cannot be used as the parent wrapper is extracted from the object.

  When after is a name, it must a name relative to the wrapper specified with the -in_wrapper option. It is a name relative to the root of the partition when the -in_wrapper option is not specified.

  The -after option is mutually exclusive to the -before, -first and -last options. When none of those four options is specified, -last is assumed.

- -first

  An optional that specifies the new configuration elements are to be loaded before any other element in the parent wrapper.

  The -first option is mutually exclusive to the -before, -after and -last options. When none of those four options is specified, -last is assumed.

- -last

  An optional switch that specifies the new configuration elements are to be loaded after all other elements in the parent wrapper.

  The -last option is mutually exclusive to the -before, -after and -first options. When none of those four options is specified, -last is assumed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -replace

  An optional switch that suppresses the error that is normally generated when loading configuration elements having the same name as an existing element in the destination wrapper. The error only occurs if the name is defined as not repeatable in the metadata. Instead of an error, the existing element is first deleted.

Examples

Example 1

The following example loads the content of the *mydesign.dft_spec_rtl* file into the Tessent Shell environment.

```
read_config_data mydesign.dft_spec_rtl
```

Example 2

The following example reads a CSV file into a wrapper of type CSV.

```
add_config_element csv(1)
read_config_data data.csv -in_wrapper csv(1)
report_config_data csv(1)
```

Related Topics

Configuration-Based Specification

display_specification

process_dft_specification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_core_descriptions

Context: all contexts

Mode: setup

Reads the specified Tessent core description (TCD) files.

Usage

read_core_descriptions *filenames*

Description

Reads the specified Tessent Core Description (TCD) files.

The Tessent Core Description file is used to map scan information from one level to the next for running ATPG at the higher level, and to retarget patterns for instances of the associated core. Additionally, the TCD is used to describe EDT IP instances for ATPG.

The Tessent Core Description format is also used to document the behavior of Memory modules, the embedded BoundaryScan content of a module, or the behavior of a FuseBoxInteface module. See "Tessent Core Description" on page 4793 for descriptions.

To view the contents of a Core wrapper loaded with this command, use the "report_config_data Core(module_name) -partition tcd -level 1" command to see the first level of wrappers. Then, use the "report_config_data Core(module_name)/XXX -partition tcd" command to see the content of wrapper XXX.

The TCD format contains public wrappers such as Memory, BoundaryScan, and FuseBoxInterface, and private wrappers such are AtpgMode(mode_name) and LogicBistMode(mode_name). The public wrapper formats are documented in the Tessent Core Description chapter.

Arguments

- ***filenames***

  A required, repeatable string that specifies the core description files to be read. The names of the files can contain wildcards. You can define the names of files to read into a Tcl list, and use the name of the Tcl list as the ***filenames*** argument.

Examples

The "Retargeting Example" in the*Tessent Scan and ATPG User's Manual* demonstrates the use of this command in a design context.

Related Topics

add_core_instances

delete_core_descriptions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_core_instances

report_core_descriptions

report_core_instances

write_core_description

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_cpf

Context: all contexts

Mode: setup

Loads a CPF (Common Power Format) file and removes any previously loaded power data.

Usage

read_cpf ***filename*** [{> | >>} *file_pathname*]

Description

Loads a CPF (Common Power Format) file and removes any previously loaded power data. You can use multiple CPF files by using the native CPF "include" command.

The tools support CPF version 1.0 and 1.1.

Effects of Reading Power Data

Loading power data from a CPF or UPF file has the following effects:

- Simulation of test procedures and scan patterns take into account whether a gate is powered on during simulation.
- The tool runs the Power-Aware Rules (V Rules) when transitioning out of Setup mode.
- Scan insertion restricts scan stitching such that each scan chain includes only scan cells from one power domain.
- This command loads power data into the tool and the report_gates command enables power data reporting to show the power on (PON) or power off (POFF) as well as the power domain of the gate.
- Test point sharing is restricted based on power information.
- The power attributes provide access to the power information. The power attributes are:

Data Model	Power Attribute
Instance, Module	is_power_always_on_cell is_power_isolation_cell is_power_level_shifter_cell is_power_retention_cell
Instance, Module, Net, Pin, and Port	power_domain_name

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

As described in the create_dft_specification command description, the DftSpecification is affected by the power domain information to make sure that groups of DesignInstances associated to different power domains are isolated by a SIB. This is done to avoid daisy chaining them with elements from a different power domain and running into issues when accessing the elements in a given power domain while the other power domains are off.

Arguments

- ***filename***

  A required string that specifies the name of the CPF file.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

This example loads a CPF file:

read_cpf cpf_file

Example 2

In this example, read_cpf loads power data into the tool and the report_gates command reports the power on (PON) status as well as the power domain of the gate.

read_cpf *cpf_file* /lp_case_si_rst_sms/ati_rst_sync/sync_r/U3/Udff

report_gates /lp_case_si_rst_sms/ati_rst_sync/sync_r/U3/Udff

```
//  /lp_case_si_rst_sms/ati_rst_sync/sync_r  sync3msfqxss1ul
//      SDI    I(PON)  /vl_sms_lp_case_si_sms_proc_sms_1_stp/
U_lp_case_si_sms_proc_bist/uu5/Z
//      D      I(PON)  /lp_case_si_rst_sms/ati_rst_sync/sync_buf/Z
//      SEN    I(PON)  /se
//      CLK    I(PON)  /lp_case_si_rst_sms/ati_rst_sync/uu1/Z
//      Q      O(PON)  /lp_case_si_rst_sms/ati_rst_sync/uu2/A  /
lp_case_si_rst_sms/and_r/B
//      in power domain PD_P2
```

Related Topics

add_faults

delete_faults

read_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_upf

report_faults

report_power_data

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_def

Context: dft, patterns

Mode: setup

Prerequisites: The current design must be set with the set_current_design command.

Reads in the DEF file so that the tool can extract the coordinates of instances in the design and the dimensions of the design.

Usage

read_def ***filename*** [-instance_of_current_design *instance_name*]
[-sequential_cells ON | OFF] [-default_udm *default_udm*]

Description

Reads in the DEF file so that the tool can extract the coordinates of sequential cells, ports of the current design, instances reported by get_memory_instances, and lower-level physical blocks. A lower-level physical block is a module of type design that is placed or fixed in the DEF file. The read_def command also extracts (x,y) coordinates for modules with a type attribute of "cell" that contain RAM or ROM primitives, as well as for memory wrapper modules with a type attribute of "design" that contain a single memory cell instance.

When using this command, the physical coordinates of the memories and sequential elements are used for grouping memories during MemoryBIST insertion and for placement of Boundary Scan cells. Note that Tessent Scan does not use the DEF file for scan insertion or stitching.

The tool stores the (x,y) coordinates of top-level ports and all instances in the following attributes:

- def_x_coordinate
- def_y_coordinate

All coordinates are floating point values in microns (um).

The tools also stores the orientation of the instances in the following attribute:

- def_orientation

Refer to "Instance" on page 3668.

In addition, the read_def command also extracts the lower left and upper right coordinates of the die. The lower left coordinates (xl, yl) and the upper right coordinates (xh, yh) of the die are stored in the following attributes for the top module:

- def_xl_coordinate
- def_yl_coordinate

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- def_xh_coordinate
- def_yh_coordinate

The read_def command also extracts lists of the x coordinates and corresponding y coordinates for the top module of the current design. This information provides the physical shape, size, and location of the die. The tool stores the lists in these attributes:

- def_x_coordinate_list

  This list contains the x coordinates for the top module of the current design.

- def_y_coordinate_list

  This list contains the y coordinates for the top module of the current design.

For example:

```
blockA  def_x_coordinate_list=='{0 0 8040 8040 13080 13080 10120 10120}'
def_y_coordinate_list=='{0 2347400 2347400 2352640 2363120 2363120 0}'
```

An x coordinate and corresponding y coordinate form a vertex of a polygon. The polygon describes the physical shape, size, and location of the top module of the current design.

Refer to "Module" on page 3654.

Arguments

- ***filename***

  A required string that specifies the name of the DEF file.

- -instance_of_current_design *instance_name*

  An optional switch and value pair that is used when the current design in Tessent Shell is a sub-module of the design that existed in the synthesis or layout tool when creating the DEF file. This situation occurs when you want to perform DFT insertion on a sub-block module and the sub-block module was synthesized as part of its parent physical block, so therefore it is not possible to obtain a DEF file for the sub-block alone. In this case, the DEF file created for the parent physical block is used and you specify, with the -instance_of_current_design switch, the instance inside the physical block DEF file that matches the sub-block module that is set as the current design in Tessent Shell. The coordinates of the instances found below that instance in the DEF file are attached to the corresponding instances found below the current design inside Tessent Shell. This enables the tool to create clusters of memories and do all other physically aware placement supported by Tessent Shell. For more information about the differences between a sub-block and physical block, refer to the set_design_level command description.

- -sequential_cells ON | OFF

  An optional switch and literal pair that specifies whether to extract the coordinates for sequential cells. The default is ON.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -default_udm *default_udm*

  An optional switch and integer pair that overrides the default dbu-per-micron value of 100 from the DEF file.

Related Topics

read_lef

add_gate_to_rtl_mapping

delete_gate_to_rtl_mapping

report_gate_to_rtl_mapping

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_design

Context: dft, patterns

Mode: setup

Reloads a design after DFT insertion to perform analysis or a future DFT insertion pass.

Usage

read_design [[*design_name*] [-design_identifier *design_identifier*] [-view *view*]
[-no_hdl] | -from_dictionary *from_dictionary*] [-verbose]
[-variant_icl_id *variant_id*] [-variant_scan_graybox_id *variant_scan_graybox_id*]
[-force] [-skip_existence_checks] [-include_child_blocks_icl_and_pdl]

Description

You must use the read_verilog and read_vhdl commands to initially read your design into the tool. Once you have performed a DFT insertion pass on the design, the list of read_verilog and read_vhdl commands required to reload the modified design into the tool is stored in the *design_name.design_source_dictionary* file located in the dft_inserted_designs directory. (See "Tessent Shell Data Base (TSDB)" on page 4807.)

For all subsequent reloads of the design, use the read_design command to automate reading the current view of the design. Additionally, the ICL view automatically loads all contexts. If you use the extract_icl command, the top-level ICL module automatically deletes to enable re-extracting it.

If the location of the chip repository is different from one site to another, use the "set_tsdb_options -directory_path_map" switch to specify a mapping. See "How to Relocate the TSDB Directories" on page 4808 for a detailed discussion on how the design source directory includes absolute, relative, and logical file path names to fully support TSDB relocation.

If read_design encounters ports that can cause issues in simulation or in the TSDB flow, a warning is issued:

```
// Warning: The design 'sv_mod1' just read has ports with 'unpacked
struct'/'enumerate'
// datatypes declared in Global ($unit) or local scope ('sv_mod1' scope).
// If you try to elaborate this design within its parent, you will get an
error.
// It will only work if this design is your current design.
```

These issues typically arise when port declarations use data types that are not visible to the tool. The warning indicates that a call to set_current_design of the parent block will raise additional warnings. Use report_current_design after running set_current_design to obtain more information about these ports.

If the top design is read in using read_design, the *design_level* is also imported from the *.tcd* file found in the dft_inserted_designs directory.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

You must use the read_design command before running any set_design_sources commands to avoid the possibility of overwriting your set_design_sources commands.

Figure 6-1 illustrates an important aspect of design views in a bottom-up flow. The figure shows a parent block instantiating two child blocks. In a bottom-up flow, you start with the golden RTL views of the parent block and the two child blocks. Once you have performed the first DFT insertion pass on the two child blocks, you may want to move up to the parent block and perform the first DFT insertion pass on the parent, and verify the DFT logic within the child blocks from the top level. The first insertion pass is typically memoryBIST and IJTAG. In Figure 6-1, the blue child blocks within the blue parent block (top-right corner of the figure) illustrates the RTL view where memoryBIST and IJTAG were inserted in all three blocks. You can use this view to fully verify the memoryBIST in the parent block and within the child blocks as accessed from the IJTAG scan interface at the boundary of the parent block.

When you want to perform the second DFT insertion pass, which is typically for the logic test insertion pass, you start again with the two child blocks. When you want to do the logic test DFT insertion pass on the parent block, you start with the design view shown in the bottom-right corner of Figure 6-1. Use the read_design command to achieve this task. Load the blue view of the parent block using the following command syntax:

```
read_design parent_module_name -design_id blue
```

When issued, the preceding command only loads the modules below the parent block but above the two child blocks. This is true for the HDL part (Verilog, VHDL, or both), and it is also true for ICL. The ICL file found inside the blue TSDB dft_inserted_designs directory contains the complete hierarchical view of the ICL including the two child blocks. However, there is a pragma in the ICL file that separates the modules belonging to the top module and those belonging to the child blocks. The pragma in this comment is as follows:

```
//tessent_pragma child_block_beginning
```

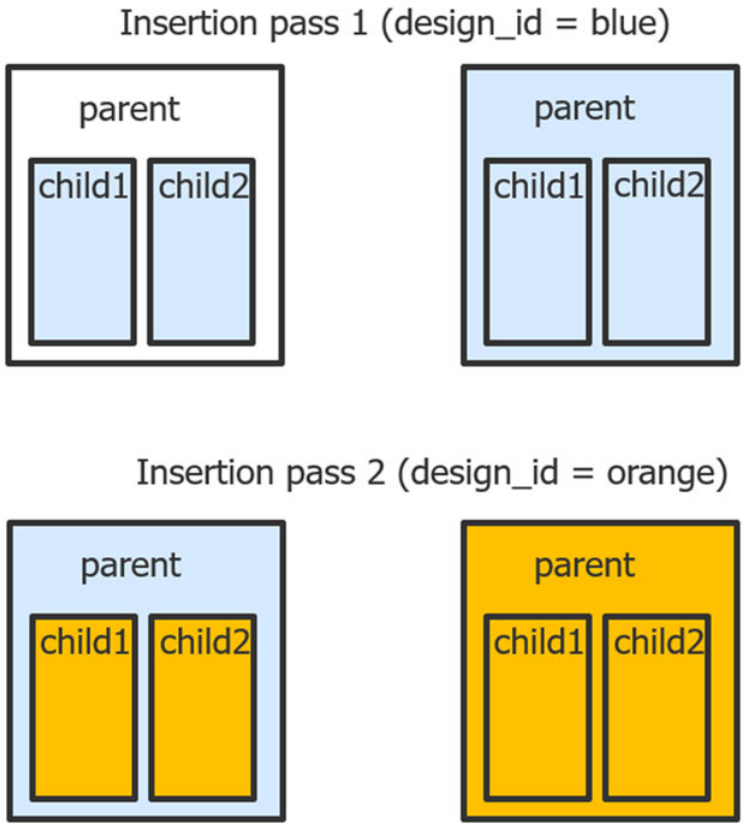
In the dft context, when you issue the read_design command with either the -no_hdl or the -view full option, it invokes the read_icl command with the switch -skip_child_blocks such that only the ICL module found between the top level and the first layer of child blocks is loaded. Figure 6-1 illustrates why this is done. When doing the second DFT insertion pass of the parent, it is the orange view of the child block that must be loaded in memory, not the blue view. If the read_design command loads the full ICL view of the parent module all the way down to the bottom, then the tool would load the blue ICL view of the child blocks.

If you do not read in the child blocks before you issue the set_current_design command, the child block orange interface view for the HDL and their full ICL view automatically loads from their TSDB dft_inserted_designs directory (as long as you have opened their TSDBs). You can force the loading of specific views of the child blocks by using the read_design command and specifying the -design_id and -view you want. If the child blocks have any external dependencies (for example, if top level ports are defined via packages that are in separate files),

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

then you must perform an explicit read_design for each child block. The read_design command gets the library info from the tsdb_info file. The command compiles the child block interface module into the same library previously used when insertion was done on the child. The module is also retained in memory.

Figure 6-1. Design Views in a Bottom-up Flow



The tool issues a warning when the current logical design library list has a different order than the one required by the read_design command because this can lead to a different view of the design in memory. An error is issued when the logical design library list is missing logical libraries required by the read_design command. In both cases, it is recommended to issue delete_design and re-issue the set_logical_design_libraries command to make it complete. The messages also list the libraries that are required.

Arguments

- *design_name*

  An optional string specifying the name of the design to reload. This value is mandatory when the design is not already elaborated and the get_current_design command returns an empty collection. When the design is already elaborated, you can omit the name of the design as it is assumed you are trying to reload the current design. You must, however, specify the -force switch when the design is already elaborated to enable the tool to destroy the current elaborated view and reload the design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -design_identifier *design_identifier*

  An optional switch-value pair used to specify the *design_id* of the design you want to read in. You only need to specify this option when the design is not already elaborated and the Tessent Shell Data Base (TSDB) contains multiple dft_inserted_designs views for the *design_name* you want to read in.

  A *design_id* must only use a combination of the following characters:

  - Alphabetic, both uppercase and lowercase. For example: “abc”, “ABC”, “AbC”, and so on.
  - Numeric. For example: “123”.
  - Underscore. For example: “A_1”.

- -view full | scan_graybox | ijtag_graybox | interface

  An optional switch-value pair used to specify which view of the design to load. When you specify interface, the command reads in the interface view from the TSDB, which includes only the ports and the parameters definitions of the module. When you specify scan_graybox, it loads the graybox view that was created by the analyze_graybox command and stored in the TSDB using the write_design -tsdb -graybox command. When you specify ijtag_graybox, it loads the graybox view that was created by the extract_icl command and stored in the TSDB output directory.When you specify full, the commands reads in the complete RTL or netlist view of the design. The command reads in the RTL view when in the -rtl context and the netlist view when in the -no_rtl context. When the -view switch is not specified, the full view is loaded. Using “read_design -view interface” is the recommended method to load the ICL and the interface view to use with the process_patterns_specification command.

  Note

  If you define the ports of a child block using parameters from a separate package file, to generate a clean interface model of the child block, you must load the package file separately using the “read_verilog <*pkg file*> -format sv2009” command. This package file is then explicitly marked as “needed_before_reading_interface_view on” in the <*child block + design_name*>*.design_source_dictionary* file.

  In higher-level design insertion or integration, the child block’s read interface view only reads the interface model plus the package file, as needed. Failing to satisfy this requirement results in all design files being loaded when you only want to read the child block interface model. The higher level *design_name.design_source_dictionary* file now also includes the lower-level child block’s design.

- -no_hdl

  An optional Boolean switch used to specify that you want to load the ICL and TCD views of the design but skip loading the Verilog or VHDL view. You use this switch when you got a netlist from synthesis or layout and you want to load the rest of the information from its pre-synthesis or pre-layout view as shown in Example 3.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -from_dictionary *from_dictionary*

  An optional switch-value pair used to supply a Tcl dictionary as input to the read_design command. When using the -from_dictionary switch, you must specify a DICTIONARY for from_dictionary, not a file that contains a dictionary, otherwise the tool issues an error similar to the following:

  ```
  //  Error: The supplied dictionary is not a list with an even number
  //  of elements.
  ```

  The format of the Tcl dictionary is documented in the dft_inserted_designs section of the TSDB chapter.

  Note

  You cannot read in the dictionary stored in the TSDB using the -from_dictionary option.

  This switch is useful to read in the design for the first time into the tool. You can have your RTL flow generate the dictionary and use read_design -from_dictionary to load it into the tool using the file dictionary you imported into the tool. You can also create a Tcl Dictionary by reading the design into the tool using read_verilog and read_vhdl and then calling get_design_sources -file_dictionary after having called the set_current_design command.

  See "Example 4" on page 1498.

- -verbose

  An optional Boolean switch used to request that the command transcript all the read commands used to load the design.

- -variant_icl_id *variant_id*

  An optional switch and string pair that specifies an ICL variant identifier causing the command to load and read the ICL variant *"design_name*.icl_*variant_id"* instead of the default *"design_name*.icl". This is useful in two circumstances:

  a. When you need to generate simulation patterns for a particular ICL variant.

  b. When you need to use an ICL variant for a child block in your design and use this variant in the parent ICL extraction to create a variant ICL for the parent itself.

- -variant_scan_graybox_id *variant_scan_graybox_id*

  An optional switch and value pair to read a specific scan graybox variant previously created with the "write_design -graybox -variant_scan_graybox_id *variant_scan_graybox_id*" command. You can only use this switch when you specify the "-view scan_graybox" option. Typically, you need to create several scan grayboxes and differentiate them with a *variant_scan_graybox_id* if there are several external scan modes in a given design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -force

  An optional Boolean switch used to disable the error you would receive if calling the read_design command when a design is already elaborated inside the tool.

  If you run “read_design -force” and the design that was already elaborated has any of clocks, input constraints, and black boxes definitions, the tool restores these after the newly loaded design is re-elaborated using the set_current_design command.

- -skip_existence_checks

  An optional Boolean switch used to suppress the errors that the tool normally issues if the design-source dictionary contains nonexistent file or directory references. Refer to the example in the write_design_source_dictionary command description to understand when you should use this switch.

- -include_child_blocks_icl_and_pdl

  An optional Boolean switch that instructs the read_design command to not skip the child block's ICL and PDL definitions present in the *.icl* and *.pdl* files of the design.

  By default, when in dft context, read_design skips reading the child-block ICL and PDL definitions unless the following condition is met: the child-block has been updated, and only the ICL and PDL of the current design need updating with this information. This switch forces the tool to always read ICL and PDL definitions for child blocks.

  By default, when in “patterns -ijtag” context, the tool always asserts this switch because the tool needs the full ICL and PDL definitions of the current design to create and process a PatternsSpecification.

Examples

Example 1

This example shows how to use the read_design command to reload a design into the tool as it was saved after a DFT insertion pass that used a *design_id* equal to “rtl”:

```
set_context dft -rtl -design_id rtl2
read_design mytop -design_id rtl
set_current_design mytop
```

If you exited Tessent Shell after performing DFT insertion, you can use the same steps listed above to reload the design from a TSDB directly without having to perform DFT insertion again. If the TSDB resides in the same directory where Tessent Shell is run and uses the default name (*tsdb_outdir*) then the TSDB is automatically opened. However if the TSDB has a different name or resides in a different directory then one must use the “open_tsdb *TSDB_path*” command prior to issuing the commands shown above.

Example 2

This example shows how to load the concatenated netlist after some instruments were inserted into the netlist and the instruments were synthesized using the run_synthesis command. The process_dft_specification command creates the *design_name.vg_no_instruments* view in the dft_inserted_designs directory. The run_synthesis commands synthesizes the instruments and

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

concatenates the netlist of the synthesized instruments with the *.vg_no_instruments* netlists into a single .vg netlist. The "read_design -force" command reloads the .vg netlist into the tool.

```
process_dft_specification
set_run_synthesis_options dc_shell -startup ../.synopsys_dc.setup
run_synthesis -wait
read_design -force
set_current_design
```

Example 3

The following example uses the -no_hdl option of the read_design command to load the information about *my_design* from the pre-synthesis TSDB directory and combine it with the post-synthesis netlist to create a complete gate level TSDB directory. This enables using "read_design *my_design* -design gate" to create patterns later.

```
set_context dft -design_id gate -no_rtl
read_cell_library cell_library
read_verilog path/my_design.vg
read_design my_design -design_id rtl -no_hdl -verbose
set_current_design my_design
write_design -tsdb -softlink_netlist -verbose
```

Refer to Example 2 in run_testbench_simulations for a usage example about how to use the write_design -tsdb -softlink_netlist command to create a TSDB dft_inserted_designs container for a netlist generated from synthesis or layout, and on which you want to use the run_testbench_simulations command to simulate the patterns described in your PatternsSpecification.

Example 4

The following example demonstrates reading a Tcl dictionary from a saved dictionary file:

```
INSERTION> write_design_source_dictionary Chip.design_source_dictionary -replace
INSERTION> set_system_mode setup
.....
#Reread the design when you enter the setup mode.
SETUP> delete_design
SETUP> source Chip.design_source_dictionary
SETUP> read_design -from_dictionary $design_source_dictionary
```

Related Topics

get_design_sources

run_synthesis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_failures

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: analysis

Checks the specified failure file for correct syntax and semantics and optionally appends a failure file.

Usage

read_failures ***failure_file*** [-Pattern | -Cycle] [{> | >>} *output_file*] [-verbose { on | off }]

read_failures ***failure_file*** [-Append]

Description

Checks the specified failure file for correct syntax and semantics and optionally appends a failure file.

By default, this command displays the first 20 pattern mismatches. Use the set_design_sources command to change the number of pattern mismatches that display.

You can also use the read_failures command to display additional information about failed cells within the failure file, or use it to convert a failure file to either pattern-based or cycle-based format. This command supports both compressed and uncompressed test patterns.

Test patterns must be WGL or STIL format for cycle-based failure files.

When you use “read_failures -append”, the order in which you supply the failure files must match the order in which the multiple patterns sets have been loaded. In the event that one of the failure files is inconsistent with the corresponding test set, the tool issues an error and does not append that failure file to the previously loaded failure files.

Note

If you omit the -append switch, then the tool overwrites the in-memory failure file.

Tessent Diagnosis performs a basic syntax check during parsing.

Arguments

- ***failure_file***

  A required string that specifies the name of the failure file to read. For failure mapping, this file is a top-level failure file generated as a result of applying top-level patterns created by scan pattern retargeting.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The filename for a core-level failure log generated during reverse mapping (refer to "Reverse Mapping Top-Level Failures to the Core" in the *Tessent Diagnosis User's Manual* for details) contains escaped characters if the core instance name contains escapes. In this case, the *failure_file* string requires additional escape characters. For example, you must specify *failure_file* for an escaped log file named *test.flog__c2_1\[3\]__core2__internal* as:

```
read_failures {test.flog__c2_1\\\[3\\]__core2__internal}
```

- -Pattern

  An optional switch that displays/writes the failure file in the pattern-based format. Use this switch to convert a cycle-based failure file to a pattern based format or use it to add location information to failed cells within a pattern-based failure file.

- -Cycle

  An optional switch that displays/writes the failure file in the cycle-based format. Use this switch to convert a pattern-based failure file to a cycle-based format or use it to add location information to failed cells within a cycle-based failure file.

- -verbose { on | off }

  An optional switch that specifies whether to disable inclusion of comments for the names and chains/offsets for all scan cells. By default, the tool displays all comments. For compressed designs, this could mean up to 500 lines of comments per failing bit. Disabling the comments only applies to the current command execution.

  When not disabled, the default verbosity defers to the "set_diagnosis_options -verify_patterns on { verbose | noverbose }" setting.

- -Append

  An optional switch that appends the *failure_file* to a previously-specified failure file and saves the appended *failure_file* in memory.

  Before appending a failure file, you must issue the read_failures command without the -append switch and, subsequently, issue the read_failures command with the -append switch.

- > *output_file*

  An optional redirection operator and pathname pair used at the end of the argument list to create or replace the contents of *output_file*.

- >> *output_file*

  An optional redirection operator and pathname pair used at the end of the argument list to append output to the contents of *output_file*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example loads a test pattern source (*1pat.wgl*), reads in a pattern-based failure file (*padding_failures.pat*) and writes out a cycle-based failure file (*pat_cycle.fail*).

```
read_patterns 1pat.wgl
read_failures fail_files/padding_failures.pat -cycle > results/pat_cycle.fail
```

Example 2

In the following command sequence example, the tool appends *failure_file2* to *failure_file1*:

```
read_failures failure_file1

read_failures failure_file2 -append
```

Example 3

In the following command sequence example, the tool appends *failure_file2* to *failure_file1*:

```
read_patterns pat1

read_patterns pat2 -append

read_failures failure_file1

read_failures failure_file2 -append
```

Example 4

The following example shows the *padding_failures.pat* file.

```
chain_test
0  chain5     0   H  L
0  chain5     1   H  L
0  chain5     4   H  L
0  chain5     5   H  L
0  chain5     8   H  L
0  chain5     9   H  L
0  chain5     12  H  L
0  chain5     13  H  L
0  chain5     16  H  L
0  chain5     88  H  L
0  chain5     100 H  L
last_pattern_tested  0
scan_test
0   chain5    2   H  L
0   chain5    4   H  L
0   chain5    6   H  L
0   chain5    8   H  L
0   chain5    9   H  L
0   chain5    11  H  L
0   chain5    12  H  L
0   chain5    13  H  L
0   chain5    28  H  L
0   chain5    100 H  L
last_pattern_tested  2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 5

In the following command sequence example, the tool appends *failure_file2* to *failure_file1*:

read_failures failure_file1

read_failures failure_file2 -append

Example 6

The following example shows the cycle-based failure file (*pat_cycle.fail*). The cycle-based data is written in columns from left to right: cycle id, pin name, expected value, simulated value, pattern id, chain/PO, cell number, cell path name.

```
format cycle
failures_begin
// chain test
105 dhiraj_so5 H L  // pat 0  chain chain5  cell 0   uUART/reg_MODE0_CLK
106 dhiraj_so5 H L  // pat 0  chain chain5  cell 1   uCNTR/reg_PORT_3_SAMPLE_P1_2_
109 dhiraj_so5 H L  // pat 0  chain chain5  cell 4   uCNTR/reg_PORT_3_SAMPLE_P1_5_
110 dhiraj_so5 H L  // pat 0  chain chain5  cell 5   uINTR/ix546
113 dhiraj_so5 H L  // pat 0  chain chain5  cell 8   uINTR/ix476
114 dhiraj_so5 H L  // pat 0  chain chain5  cell 9   uINTR/ix446
117 dhiraj_so5 H L  // pat 0  chain chain5  cell 12  uINTR/ix356
129 dhiraj_so5 H L  // pat 0  chain chain5  cell 24  uSTM/ix56/SFFR
 ...
// scan test
314 dhiraj_so5 H L  // pat 0  chain chain5  cell 2   uCNTR/reg_PORT_3_SAMPLE_P1_3_
316 dhiraj_so5 H L  // pat 0  chain chain5  cell 4   uCNTR/reg_PORT_3_SAMPLE_P1_5_
318 dhiraj_so5 H L  // pat 0  chain chain5  cell 6   uINTR/ix106
320 dhiraj_so5 H L  // pat 0  chain chain5  cell 8   uINTR/ix476
321 dhiraj_so5 H L  // pat 0  chain chain5  cell 9   uINTR/ix446
323 dhiraj_so5 H L  // pat 0  chain chain5  cell 11  uINTR/ix386
621 dhiraj_so5 H L  // pat 2  chain chain5  cell 101 padding_cycle
failure_buffer_limit_reached all
failures_end
```

Example 7

The following example loads a test pattern source (*1pat.wgl*), reads in a cycle-based failure file (*pat_cycle.fail*) and writes out a pattern-based failure file (*cyc_pat.fail*).

read_patterns 1pat.wgl
read_failures fail_files/pat_cycle.fail -pattern > results/cyc_pat.fail

Example 8

The following example shows the pattern-based failure file (*cyc_pat.fail*). The pattern-based failure file data is written in columns from left to right: pattern_id, chain/PO_name, cell_number, expected_value, simulated_value, cell_path_name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
chain_test
0 chain5   0   H  L  //  uUART/reg_MODE0_CLK
0 chain5   1   H  L  //  uCNTR/reg_PORT_3_SAMPLE_P1_2_
0 chain5   4   H  L  //  uCNTR/reg_PORT_3_SAMPLE_P1_5_
0 chain5   5   H  L  //  uINTR/ix546
0 chain5   8   H  L  //  uINTR/ix476
0 chain5   9   H  L  //  uINTR/ix446
0 chain5   12  H  L  //  uINTR/ix356
0 chain5   13  H  L  //  uINTR/ix326
0 chain5   16  H  L  //  uINTR/ix236
0 chain5   88  H  L  //  padding_cycle
0 chain5   100 H  L  //  padding_cycle
last_pattern_tested  0
scan_test
0 chain5   2   H  L  //  uCNTR/reg_PORT_3_SAMPLE_P1_3_
0 chain5   4   H  L  //  uCNTR/reg_PORT_3_SAMPLE_P1_5_
0 chain5   6   H  L  //  uINTR/ix106
0 chain5   8   H  L  //  uINTR/ix476
0 chain5   9   H  L  //  uINTR/ix446
0 chain5   11  H  L  //  uINTR/ix386
0 chain5   12  H  L  //  uINTR/ix356
0 chain5   13  H  L  //  uINTR/ix326
0 chain5   28  H  L  //  uSTM/reg_WAIT_SYNC/SFFR
0 chain5   100 H  L  //  padding_cycle
last_pattern_tested  2
```

Example 9

The following example loads a test pattern source(*1pat.wgl*), reads in a compressed pattern-based failure file (*fail_file.pat*) and writes out a compressed pattern-based failure file (*pat_2_pat*) that includes cell path names for internal scan chains corresponding to each failing EDT channel.

```
read_patterns 1pat.wgl
read_failures fail_file.pat -pat > results/pat_2_pat
```

Example 10

The following example shows the compressed pattern-based failure file (*fail_file.pat*) loaded into the tool.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
chain test
   0  edt_channel1  2  H  L
   0  edt_channel1  3  L  H
   0  edt_channel2  0  L  H
   0  edt_channel2  4  H  L
   1  edt_channel1  0  H  L
last_pattern_tested 1
scan test
   0  /x_out[4]  H  L
   0  /x_out[2]  H  L
   0  /y_out[5]  L  H
   0  /y_out[4]  L  H
   0  edt_channel1  0  L  H
   0  edt_channel1  1  H  L
   0  edt_channel1  2  L  H
   1  /x_out[3]  H  L
   1  /x_out[2]  H  L
   1  /x_out[1]  H  L
last_pattern_tested 1
```

Example 11

The following example shows the resulting compressed pattern-based failure file (*pat_2_pat*) output by the tool. Data is written in columns from left to right: pattern_id, channel/PO_name, channel_cycle_id, expected_value, simulated_value, chain---cell_id---cell_path_name. For compressed patterns, the location of every possible failing cell in the internal scan chains corresponding to each failing EDT channel is listed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
chain_test
0  edt_channel1  2  H  L   //  chain1---2---circle_i/reg_x_2_/DF1
                            //  chain2---2---circle_i/reg_f_6_
                            //  chain3---2---circle_i/reg_f_1_
0  edt_channel1  3  L  H   //  chain1---3---circle_i/reg_x_3_/DF1
                            //  chain2---3---circle_i/reg_f_7_
                            //  chain3---3---circle_i/reg_f_2_
0  edt_channel2  0  L  H   //  chain4---0---circle_i/reg_y_2_
                            //  chain5---0---circle_i/reg_x_5_/DF1
0  edt_channel2  4  H  L   //  chain4---4---circle_i/reg_y_6_
                            //  chain5---4---circle_i/reg_y_1_
1  edt_channel1  0  H  L   //  chain1---0---circle_i/reg_x_0_/DF1
                            //  chain2---0---circle_i/reg_f_4_
                            //  chain3---0---circle_i/reg_y_7_
last_pattern_tested  1
scan_test
0  /x_out[4]        H  L   //  PO
0  /x_out[2]        H  L   //  PO
0  /y_out[5]        L  H   //  PO
0  /y_out[4]        L  H   //  PO
0  edt_channel1  0  L  H   //  chain1---0---circle_i/reg_x_0_/DF1
                            //  chain2---0---circle_i/reg_f_4_
                            //  chain3---0---circle_i/reg_y_7_
0  edt_channel1  1  H  L   //  chain1---1---circle_i/reg_x_1_/DF1
                            //  chain2---1---circle_i/reg_f_5_
                            //  chain3---1---circle_i/reg_f_0_
0  edt_channel1  2  L  H   //  chain1---2---circle_i/reg_x_2_/DF1
1  /x_out[3]        H  L   //  PO
1  /x_out[2]        H  L   //  PO
1  /x_out[1]        H  L   //  PO
last_pattern_tested  1
```

Example 12

In the following command sequence example, the tool appends *failure_file2* to *failure_file1*:

read_patterns pat1

read_patterns pat2 -append

read_failures failure_file1

read_failures failure_file2 -append

Example 13

The following example loads the top-level *fail1.top* failure file.

read_failures fail1.top

Related Topics

diagnose_failures

report_failures

set_diagnosis_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_failures

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_fault_sites

Context: dft -edt, patterns -scan

Mode: analysis

Loads the path or bridge definitions or user-defined fault models (UDFM) contained in the specified ASCII file into the current tool session.

Usage

Path Delay Faults Usage:

read_fault_sites ***filename***… [-Force | -Noforce]
[-INSTance *instance_name*... | -MODule *module_name*...] [{> | >>} *output_filename*]

Bridge Faults Usage:

read_fault_sites ***filename***…
[-INSTance *instance_name*... | -MODule *module_name*...]
[-VErbose [All | Modules | Instances]] [-NAmes_ignored]
{[-NOEQ] |
{[-Filter "{DISTance | PARAllel_run | WEIGHT} *operator number"*]…
[-Filter "TYPE {= | !=} *type_identifier*"]…
[-Filter "LAYER {= | !=} *string"*]…
[-SElect *number*[%]]…}} [{> | >>} *output_filename*]

UDFM Faults Usage:

read_fault_sites ***filename*** [-VErbose]
[-INSTance *instance_name*... | -MODule *module_name*...]
[-OBSERVATION {ON | OFf}] [-TWO_WAY_BRIDGE_MODEL {ON | OFF}]
[-Filter "{IMPACT | TCA} *operator number*"]…

Description

Loads the path or bridge definitions or user-defined fault models (UDFM) contained in the specified ASCII file into the current tool session.

Bridge Fault Specifics

A bridge fault definition file or output file from the Calibre query server can be loaded with this command. If a syntax error is encountered, the tool issues an error message and stops loading the bridge entries from the file. If a non-syntax error occurs, an error displays but the file continues loading.

After bridge loading completes, the tool displays a summary of all errors and warnings encountered during the load. The summary lists how many bridges could not be mapped and why. For example, a bridge cannot be mapped if it relates to a location within a library cell.

Note

If you define one or more filters using the -Filter switch, the command loads only those bridges that satisfy all specified filter criteria.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For more information on bridge faults and the bridge fault definition file, refer to "Net Pair Identification with Calibre for Bridge Fault Test Patterns" in the *Tessent Scan and ATPG User's Manual*.

Path Delay Specifics

Before you use the tool to create ATPG patterns to detect path delay faults, first place the paths into a properly formatted file. Then use this command to load the file, and the add_faults or read_faults command to add the faults on the specified paths. You must also specify the path delay fault type with the set_fault_type command.

If you return to Setup mode with faults in the path delay fault list, the tool deletes the faults in that list and issues a warning message.

You can use multiple read_fault_sites commands and the results are additive. However, if you do not use the -Force switch, and one of the following conditions is met when the tool reads in the path data, the tool issues an error message and terminate the command:

- The path name must not be the same as the name of an existing path.
- The first pin for a path must be a valid launch point. A valid launch point is a primary output of a scan cell state element, or a non-scan state element that satisfies the C1 clock rule.

  If the path includes a clock or state element D-input pin, you must include the state element name in the path (or use the -Force switch). If you do not, the tool does not resolve the path, and it reports an error.
- The last pin for a path must be a valid capture point or a clock input of a scan cell. A valid capture point is a primary output or a data input of a cell that can capture data, whether scanned or not. Examples of invalid capture points would be a latch that is transparent or a memory element that cannot capture data due to a DRC violation, pin constraint, or cell constraint.
- Each pin must have unambiguous fan-in connectivity to the preceding pin, which must not tie to a constant logic value. If the pin fails to have a valid connection with the preceding pin, the tool generates an error and terminates the read_fault_sites command. However, if there is ambiguity in the connectivity, the tool selects a path between the pin and the preceding pin and generates a warning message. You can display the gates in the complete path using the Report Path command.
- Paths cannot propagate through RAM gates, ROM gates, or transparent latches.
- Paths cannot have edge ambiguity during any point in the path. An edge that propagates through XOR gates or the select lines of MUX gates can result in either a rising or falling edge at the gate outputs. You can use inversion parity to avoid edge ambiguity. If this check fails, the tool generates a warning, and you can assume that an edge on the pin is not inverted relative to the preceding pin.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The condition statements in the path definition file must occur before the first pin statement and before the tool checks for valid pin names and values. The tool does not use the conditions to resolve edge or path ambiguities.

For more information about path delay faults and the path definition file, refer to "Path Delay Test Set Creation" in the *Tessent Scan and ATPG User's Manual*.

UDFM Specifics

You use this command to load a UDFM definition file. If the tool encounters an error during loading, the tool stops the loading operation and displays a message. When the UDFM file is successfully loaded, the tool displays a summary similar to the following:

```
// UDFM static definitions read: Total=990, Cells=0, Modules=558,
//       Instances=0, No Match=432, Static faults=87485
```

The summary lists how many UDFM definitions were read from the file and how many of them have been added to the list of fault sites. In this example, 990 UDFM definitions are read from the file and 558 cells are added to the list of fault sites. The last value is the number of the UDFM definitions that did not match a cell or module in the design. You can use the -Verbose switch to report more detailed warning messages about the UDFM definitions that were not added to the list of fault sites.

For more information about UDFM, refer to "Overview of Test Types and Fault Models" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- ***filename***

  A required repeatable string that specifies the pathname of the path or bridge definition file or the UDFM file to load. This file is an ASCII file that defines the path delay or bridge fault sites or user-defined fault models for ATPG.

- > | >> *output_filename*

  An optional switch that writes the parsing errors, warnings and summary report to the specified file. Use this switch at the end of the command line. Options include:

  > Creates a new or overwrites an existing file.

  >> Appends the output to an existing file.

- -Filter {"{DISTance | PARAllel_run | WEIGHT} *operator number*"}

  An optional repeatable switch, literal, operator, and number quadruplet that filters which bridge entries are loaded based on a specified physical attribute of the bridge. A single blank space is required between each member of the quadruplet and the literal, operator, and number portion of the quadruplet must be enclosed in quotation marks. The *number* must be a nonnegative number that applies to the specified attribute in the bridge entry. Attributes include the following:

  DISTANCE — Real number that specifies the distance attribute in the bridge entry.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

PARALLEL_run — Real number that specifies the parallel_run attribute in the bridge entry.

WEIGHT — Real number that specifies the weight attribute in the bridge entry.

The *operator* must be one of the following:

< less than

<= less than or equal to

= equal to

!= not equal to

>= greater than or equal to

> greater than

Due to rounding, it is not advisable to use "=" or "!=" for floating point numbers.

Note

The tool uses the information you provide with the -Filter switch only to determine a subset of bridge entries to load; the values are not used in any other way.

- -Filter "{IMPACT | TCA} *operator number*"

An optional repeatable switch, literal, operator, and number quadruplet that filters which fault entries are loaded based on a specified attribute of the fault. A single blank space is required between each member of the quadruplet, and the literal, operator, and number portion of the quadruplet must be enclosed in quotation marks. The *number* must be a nonnegative number that applies to the specified attribute in the fault entry. Attributes include the following:

IMPACT — Integer in the range of 1 to 100 that specifies the impact value of a UDFM test alternative.

TCA — Real number greater than zero that specifies the total critical area attribute in the fault entry.

Note

UDFM interconnect bridge faults are an exception (see Bridge Fault Model in the *Tessent Scan and ATPG User's Manual*). In this case, the tool considers the TCA value of the complete bridge defect.

The *operator* must be one of the following:

< less than

<= less than or equal to

= equal to

!= not equal to

>= greater than or equal to

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

> greater than

Due to rounding, it is not advisable to use “=” or “!=” for floating point numbers.

- -Filter “LAYER {= | !=} *string*”

  An optional repeatable switch, literal, operator, and string quadruplet that filters which bridge entries are loaded based on the layer attribute of the bridge. A single blank space is required between each member of the quadruplet and the literal, operator, and string portion of the quadruplet must be enclosed in quotation marks. The *string* must be the string value that specifies the layer attribute in the bridge entry.

  Note

  The tool uses the information you provide with the -Filter switch only to determine a subset of bridge entries to load; the values are not used in any other way.

- -Filter “TYPE {= | !=} *type_identifier*”

  An optional repeatable switch, literal, operator, and literal quadruplet that filters which bridge entries are loaded based on the bridge type. A single blank space is required between each member of the quadruplet and the literal, operator, and literal portion of the quadruplet must be enclosed in quotation marks. Select one of the following literals as the *type_identifier*:

  S2S — Side-to-side

  SW2S — Side-Wide-to-Side: same as S2S but at least one of the two signal lines is a wide metal line.

  S2SOW — Side-to-Side-Over-Wide: same as S2S, but the bridge is located over a wide piece of metal in a layer below.

  C2C — Corner-to-Corner

  V2V — Via-to-Via: an S2S for vias.

  VC2VC — Via-Corner-to-Via-Corner

  EOL — End-of-Line: the end head of a line faces another metal line.

  Note

  The tool uses the information you provide with the -Filter switch only to determine a subset of bridge entries to load; the values are not used in any other way.

- -Force

  An optional switch that specifies continued path reading after the first occurrence of an invalid path. If you do not specify this switch and the command encounters an invalid path, the command generates an error and terminates.

- -INSTance *instance_name...*

  An optional switch and repeatable string that causes each fault site in *filename* to be prepended by the *instance_name* string for the reuse of a fault file created prior to the instances being embedded in the current larger design. This switch must occur after the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

filename argument. You can use the asterisk (*) and question mark (?) wildcards to specify the *instance_name* argument, and the wildcard is expanded to all legal instance names. The fault file is loaded for each instance.

- -MODule *module_name*...

  An optional switch and repeatable string that causes each fault site in *filename* to be prepended by *module_name* string for reuse of a fault file created prior to the instances being embedded in the current larger design. This switch must occur after the *filename* argument. You can use the asterisk (*) and question mark (?) wildcards to specify the *module_name* argument, and the wildcard is expanded to all legal instance names whose actual module name matches the regular expression. The fault file is loaded for each instance.

- -NAmes_ignored

  An optional switch that directs the tool to ignore all bridge names while loading. Use this option to load files having duplicate bridge names in the bridge definition or Calibre query server file, or to save memory.

- -NOEQ

  An optional switch that prevents the loading of equivalent fault sites. This has the advantage of producing the smallest peak memory demand, as well as the smallest final memory demand, for loading.

- -Noforce

  An optional switch that specifies for the tool to stop path reading after the first occurrence of an invalid path. This is the default.

- -OBSERVATION {ON | OFf}

  An optional switch and literal pair that enables you to override the setting specified with the "observation" keyword in the UDFM file. The UDFM file setting "observation:true" (default) enables fault propagation, and "observation:false" disables it, as required by fault models such as toggle bridge.

  ON — All the UDFM faults loaded by the read_fault_sites command are treated as "observation:true."

  OFf — All the UDFM faults loaded by the read_fault_sites command are treated as "observation:false."

- -SElect *integer*[%]

  An optional repeatable switch and positive integer pair that specifies to keep the *integer* best bridges of each bridge type. The tool determines the best based on the weight of each bridge after loading all fault locations in all files specified in the command line.

  If you include the optional "%" character after the *integer*, it is interpreted as a percentage instead of an absolute number. For example, "-select 20%" directs the tool to retain the best 20% of all bridges after loading, whereas "-select 20" means to keep the twenty best bridges.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

A weight attribute must be defined for each bridge in the file for this option to be meaningful. If a weight is not defined for a bridge, it is loaded regardless of this switch.

- -TWO_WAY_BRIDGE_MODEL {ON | OFF}

  An optional switch and literal pair that specifies, when set to on, for the tool to use the two-way bridge model instead of the default four-way bridge model. The default is off. For more information about the two models, refer to Bridge Fault Model in the *Tessent Scan and ATPG User's Manual*.

- -VErbose

  An optional switch that enables the display of parsing errors and warnings while the path, bridge, or UDFM ASCII file is loaded.

  For bridge entries only, this switch displays a summary of all errors and warnings once bridge entries are loaded. You can specify how these errors and warnings display with one of the following literals:

  All — Displays all errors/warnings during loading and a summary report after loading.

  Modules — Displays the number of bridges along with the names of the related modules that could not be mapped.

  Instances — Displays the number of bridges along with the names of the related modules and instances that could not be mapped.

  By default, only the summary report displays at the conclusion of the load operation.

Examples

Example 1

The following path delay example sets up the tool to perform ATPG for the specified path delay faults in the path definition file*,/user/design/pathfile*.

First, you must set the fault type and read in the paths:

set_fault_type path_delay
read_fault_sites /user/design/pathfile
report_fault_sites

```
PATH "path0" =
    PIN /I$6/Q + ;
    PIN /I$35/B0 + ;
    PIN /I$35/C0 + ;
    PIN /I$1/I$650/IN + ;
    PIN /I$1/I$650/OUT - ;
    PIN /A_EQ_B + ;
END ;
```

report_faults -all

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
type  code  name_of_path
----  ----  --------------------
//  Warning: No faults reported.
```

Next, add the faults on the paths contained in that file:

add_faults -all
report_faults -all

```
type  code  name_of_path
----  ----  --------------------
  0    UC   path0
  1    UC   path0
```

Now you are ready to perform ATPG on the path delay faults:

create_patterns

The following is an example of a path definition file that contains one path:

```
PATH "path0" =
    PIN /I$6/Q + ;
    PIN /I$35/B0 + ;
    PIN /I$35/C0 + ;
    PIN /I$1/I$650/IN + ;
    PIN /I$1/I$650/OUT - ;
    PIN /A_EQ_B + ;
END ;
```

Example 2

The following bridge fault example sets up the tool to perform ATPG for the bridge fault sites specified in the bridge fault definition file,*/user/design/bridgefile*.

First, you must set the fault type and load the bridge sites from the bridge fault definition file:

set_fault_type bridge
read_fault_sites /user/design/bridgefile

```
//
//  Load bridge entries from file "/user/design/bridgefile".
//
//  5 bridge entries were read from file "/user/design/bridgefile".
//  2 bridge entries were skipped due to unknown net name.
//  3 bridge entries were added to the fault site list.
```

report_fault_sites

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
BRIDGE {
  NET1 = "/G5";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  NAME = "U7.BRIDGE";
}
BRIDGE {
  NET1 = "/G4";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  NAME = "U5.BRIDGE";
}
BRIDGE {
  NET1 = "/G11";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  NAME = "U0.BRIDGE";
}
```

Next, add faults on the sites in the fault site list:

add_faults -all
report_fault_sites

```
BRIDGE {
  NET1 = "/G5";
  NET2 = "/G10";
  FAULTS = {UC, UC, UC, UC};
  NAME = "U7.BRIDGE";
}
BRIDGE {
  NET1 = "/G4";
  NET2 = "/G10";
  FAULTS = {UC, UC, UC, UC};
  NAME = "U5.BRIDGE";
}
BRIDGE {
  NET1 = "/G11";
  NET2 = "/G10";
  FAULTS = {UC, UC, UC, UC};
  NAME = "U0.BRIDGE";
}
```

Now you are ready to perform ATPG on the bridge faults:

create_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

The following is an example of a bridge fault definition file that contains one bridge site:

```
VERSION 1.1
BRIDGE {
  NET1="G173";
  NET2="G1296";
  NAME="My_layout.BRIDGE_TYPE1";
  DISTANCE=4.93827e-06 um;
  PARALLEL_run=6.075e-05 um;
  WEIGHT=95
  X_COORDINATE=152500;
  Y_COORDINATE=739500;
  LAYER="metal1";
  TYPE="S2S";
}
```

Example 4

The following example filters bridge entries based on the parallel_run attribute and loads only the bridge entries whose parallel_run attribute is between 7.5e-05 and 8e-05 um into the current fault definition list.

```
delete_fault_sites -all
read_fault_sites /user/design/bridgefile -filter "parallel_run > 7.5e-05" \
    -filter "parallel_run < 8e-05"
```

```
//  5 bridge entries were read from file "/user/design/bridgefile".
//  2 bridge entries were skipped due to unknown net name.
//  2 bridge entries were filtered out.
//  1 bridge entry was added to the fault site list.
```

Notice the tool loads only the bridge entries that meet the specified filter criteria, but does not otherwise utilize the attributes. To see the attributes, include the -Attribute switch when you report the fault sites.

Example 5

The following example loads all bridge entries from the *z_bridge_definitions* file and writes loading warnings and errors to the *loading_errors* file:

```
read_fault_sites z_bridge_definitions > loading_errors
//  ... writing to file loading_errors
```

Example 6

In the following example, remember that the read_fault_sites does not actually add the fault to be targeted by ATPG, but only creates a named fault for referencing later. However, create_patterns automatically adds the loaded faults when issued.

run 1: You create faults via physical extraction for module alu. You complete the run and are satisfied with the block coverage.

```
read_fault_sites alu_bridges_file
create_patterns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

run 2a: You then embed the alu design from run 1 in a large design as instance alu1. You want to reuse the fault file used in run 1 to define the faults within that particular instance of the new larger design.

read_fault_sites alu_bridges_file -instance alu1

You want all bridges loaded to be targeted, except for the one in the fault file named *bridge7*. Bridge fault sites must have names in the file and you cannot have used the -names_ignored switch, which throws away the names to save internal storage.

delete fault site alu1/bridge7 -name
create_patterns # automatically adds all nondeleted, loaded faults ...

run 2b: You embed the alu design 10 times as a different instance and want to reuse the fault file from run 1 to define the faults in each of the 10 instances of module alu.

read_fault_sites alu_bridges_file -module alu

run 2c: You embed the alu design as inst1 and inst2; because no other instances begin with "nest", you can use the * wildcard as shown:

read_fault_sites alu_bridges_file -inst inst*

In all of these runs, you can run create_patterns after loading and deleting fault sites to obtain the set you require. The create_patterns command automatically adds faults at every loaded, undeleted fault site for ATPG to target.

Example 7

The following example loads two UDFM files into the current tool session and enables display of parsing errors and warnings while loading the second file:

read_fault_sites /home/design/c090_std/c090_std.udfm
read_fault_sites /home/design/c090_io/c090_io.udfm -verbose

```
// command: read_fault_sites /home/design/c090_std/c090_std.udfm
// Reading UDFM File: /home/design/c090_std/c090_std.udfm
// File has been successfully parsed
// command: read_fault_sites /home/design/c090_io/c090_io.udfm -verbose
// Reading UDFM File: /home/design/c090_io/c090_io.udfm
// File has been successfully parsed
// Warning: Module c090_io not used in the design. Definition will be
// ignored. Location: File /home/design/c090_io/c090_io.udfm, line 31.
//    UDFM static definitions read: Total=1, Cells=0, Modules=0,
//    Instances=0, No Match=1, Static Faults=0
```

Example 8

The following example identifies and reports only very gross UDFM faults with impact greater than or equal to 99:

read_fault_sites ../data/CMOS65_2020_1.udfm.gz -filter {impact >= 99}
add_faults ip_inst/inst
report_faults -details

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Class  UdfmType            Cell           Fault               TCA    Impact  Instance
//  -----  -----------------  ----------  ---------------------  -----  ------  ------------
//  UC     intra_cell_defects  HS65_XNOR3X12  D12:Bridge:1.0_A:16_C:1     0.87   100
ip_inst/inst
//  EQ     intra_cell_defects  HS65_XNOR3X12  D216:Bridge:1.0_M34:d_M34:g 1.96   100
ip_inst/inst
//  EQ     intra_cell_defects  HS65_XNOR3X12  D462:Open:1.0G@P_B:93_B:96  9.38   100
ip_inst/inst
//  EQ     intra_cell_defects  HS65_XNOR3X12  D84:Bridge:1.0_B:5_VDD:14   25.80  100
ip_inst/inst
//  UC     intra_cell_defects  HS65_XNOR3X12  D19:Bridge:1.0_C:79_B:109   25.83  100
ip_inst/inst
//  UC     intra_cell_defects  HS65_XNOR3X12  D1:PortBridge:1.0_A_GND     0.11   100
ip_inst/inst
```

Example 9

The following example identifies and reports only the UDFM faults that meet the TCA criteria:

read_fault_sites ../data/CMOS65_2020_1.udfm.gz -filter {TCA > 10.0} -filter {TCA < 20.0}
add_faults ip_inst/inst
report_faults -details

```
//  command: read_fault_sites ../data/CMOS65_2020_1.udfm.gz -filter {TCA > 10.0} -filter
{TCA < 20.0}
//  Reading UDFM File: ../data/CMOS65_2020_1.udfm.gz
//     UDFM static definitions read: Total=1940, Cells=0, Modules=69, Instances=0, No
Match=1838, Static faults=7064
//  command: add_faults ip_inst/inst
//  Number added faults=24  instances considered=1
//  command: report_faults -details
//  Class  UdfmType            Cell          Fault             TCA    Impact  Instance
//  -----  -----------------  -------------  --------------  -----  ------  ------------
//  UC     intra_cell_defects  HS65_XNOR3X12  D25:Bridge:1.0_A:105_B:109  14.51  97
ip_inst/inst
//  UC     intra_cell_defects  HS65_XNOR3X12  D26:Bridge:1.0_A:105_15:10  17.99  82
ip_inst/inst
//  UC     intra_cell_defects  HS65_XNOR3X12  D282:Ton:1.0_M9:d_M9:s      14.59  100
ip_inst/inst
//  EQ     intra_cell_defects  HS65_XNOR3X12  D354:Open:1.0G_2:12_2:20    12.49  100
ip_inst/inst
```

Related Topics

add_faults

delete_fault_sites

report_faults

report_fault_sites

set_fault_type

write_fault_sites

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_faults

Context: dft -edt, dft -test_points, patterns -scan

Mode: analysis

Updates the current fault list with the faults contained in the specified fault file or from the Tessent Shell Database (TSDB).

Usage

read_faults [*filename*]
[-INSTance *instance_name*... | -MODule *module_name*...] [-VERBose]
[-GRaybox] [-KEEP_Patterns] [-SHow_warnings]
[-FAULT_type {stuck | iddq | transition | toggle | path_delay | bridge | udfm}]
[-MODE *mode_name*] [-PATtern_id *pattern_id*] [-DESign_id *design_id*] [-SIlent]

Stuck/Transition/Toggle/Iddq/Path_delay Specific Options

[-ALLOW_fault_sampling | -RETain | -Delete | -DELETE_Equivalent |
-Merge [-NO_Warnings]]
[-DETection {Append | Replace}]

UDFM Specific Options

[-RETain | -Delete | -Merge] [-DETection {Append | Replace}]

Bridge Specific Options

[-RETain | -Merge]

Power-Aware Options (Applicable Only After You Have Loaded CPF/UPF Power Data)

[-POWer_check {ON | Off}]

Description

The read_faults command affects the current fault population by either adding or removing faults that you specify in an external fault file or that were written to the TSDB. Because you must identify the faults before performing ATPG or fault simulation, this command is useful when you have a large number of faults to identify. The power-aware option loads faults based on the active power mode.

Provide the faults by specifying the fault filename or, if you used the write_tsdb_data command during pattern generation, specifying one of these switches, instead of the pattern filename, which eliminates the need to remember the location of the pattern file and enables the tool to locate the patterns in the TSDB:

- -mode
- -fault_type
- -pattern_id
- -design_id

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For more information, see logic_test_cores in "Tessent Shell Data Base (TSDB)" on page 4807.

The read_faults command supports the MTFI (Mentor Tessent Fault Information) format. For more information about MTFI, refer to the *Tessent Scan and ATPG User's Manual*.

Path Delay/Stuck/Transition/Toggle/Iddq Faults Specifics

By default, if you do not specify the -Retain switch, the tool initially places all the faults in the fault file into the uncontrolled (UC) fault class. It then analyzes and reclassifies them appropriately.

Note

The filename specified cannot have fault information lines with comments appended to the end of the lines or fault information lines greater than six columns. If it does, the tool does not recognize the line properly and does not add the fault on that line to the fault list.

The format of the fault file data can be either in a three-, four-, or six-column standard format. Regardless of the format, the read_faults command uses only the information in the first, second, and last columns. You may also place comment lines in the file by starting the line with a double slash (//); the tool ignores comment lines. The file follows the format illustrated below:

```
stuck fault_class (cellname) (netname) (ignore) pin_pathname
```

- **stuck** — The first column must be the stuck-at value.
- **fault_class** — The second column must be the fault class value, but only if you use the -Retain option. For detailed information on fault classes, refer to the "Fault Classes" section in the *Tessent Scan and ATPG User's Manual*.
- **cellname** — The third column is the cell name enclosed in parenthesis. When present, this column indicates the type of cell in which the fault resides.
- **netname** — The fourth column is the net name enclosed in parenthesis. When present, this column indicates the net in which the fault resides.
- **ignore** — The fifth column is ignored by both tools.
- **pin_pathname** — The last column must be the pin pathname.

Here is an example of a short fault file that provides just the data the tool uses (as described in the first, second, and last bullets above):

```
// Short fault file example
1 UC /u510/u17/A0
1 UC /u510/u17/Y
0 UU /u510/u17/A1
1 UU /u510/u17/A1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

When black boxes are present in the design, the read_faults command is unable to load the faults that are inside the black boxes. Instead, the tool discards those faults and issues warning messages. The result is fewer faults loaded into the tool.

Arguments

- ***filename***

  An optional string that specifies the name of the ASCII file containing the fault list to load. The command automatically detects when filename is an MTFI file. If the filename ends with *.gz* or *.Z*, the tool automatically uncompresses the file with the GNU gzip utility. (See set_gzip_options for more details.) You can enter only one string.

  This switch is used when you are reading faults from a file rather than the TSDB. If you used the write_tsdb_data command during pattern generation, the patterns are located in the TSDB. Specifying one of these switches eliminates the need to remember the location of the pattern file and enables the tool to locate the patterns in the TSDB: -mode, -fault_type, -design_id, and -pattern_id. The filename cannot be used if you specify one of these switches.

- -INSTance *instance_name...*

  An optional switch and repeatable string that cause each fault site in *filename* to be prepended by the *instance_name* string for the reuse of a fault file created prior to the instances being embedded in the current larger design. This switch must occur after the *filename* argument, if the filename is specified. You can use the asterisk (*) and question mark (?) wildcards to specify the *instance_name* argument, and the wildcard is expanded to all legal instance names. The fault file is loaded for each instance.

  If this switch is specified without a filename, the tool uses the specified instance names to retrieve the module names of those instances to do module matching (see set_module_matching_options) against the *design_name* of the files in the TSDB and only read the fault files that match, applying each file to the corresponding instance of the matching module.

- -MODule *module_name...*

  An optional switch and repeatable string that cause each fault site in *filename* to be prepended by *module_name* string for reuse of a fault file created prior to the instances being embedded in the current larger design. This switch must occur after the *filename* argument, if the filename is specified. You can use the asterisk (*) and question mark (?) wildcards to specify the *module_name* argument, and the wildcard is expanded to all legal instance names whose actual module name matches the regular expression. The fault file is loaded for each instance.

  If this switch is specified without a filename, the tool uses the specified module name to do module matching (see set_module_matching_options) against the *design_name* of the files in the TSDB and only read the fault files that match, applying each file to the corresponding instances of the matching module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -VERBose

  An optional switch that applies only to -Instance and -Module. Prior to being embedded, top-level I/O pins on an instance are faultable sites; after being embedded, they no longer are fault sites. In this case, a summary message states that these faults cannot be loaded due to no longer being a faultable site. If the -Verbose switch is issued, an individual message for each unloadable fault is also issued. By default, this switch is disabled.

- -GRaybox

  An optional switch that causes the instances specified with the -Instance or -Module switch to be treated as gray boxes. When you use the -Graybox switch, you must use either the -Instance or -Module switch.

  Note

  The -graybox switch cannot be combined with the -retain, -delete, or -delete_equivalent switches.

  When the tool can translate the loading faults to an existing net, the -Graybox switch merges the faults to the internal fault list. When the tool cannot translate a loading fault to an existing net, the tool designates that fault as unlisted and retains its detection status.

  Note that this option does not support reading multiple fault lists for the same instance or module. If your intent is to merge multiple fault lists for a given core, then you must perform the merge during the core-level ATPG run using the "read_faults -merge" command. Note also that you can only merge fault lists of a single fault type. You cannot, for example, merge fault lists for stuck and transition at the core level.

  For more information about using the -Graybox switch and hierarchical fault accounting, refer to "Support for Hierarchical Fault Accounting" in the *Tessent Scan and ATPG User's Manual*.

- -DETection {Append | Replace}

  An optional switch and literal operand that specify the operation on the n-detection number of faults in the FaultList data block of the loading MTFI file.

  The Append literal appends the n-detection number stored in the external MTFI file to the n-detection number of the corresponding fault. The append operation caps at the detection limit set by the set_multiple_detection command. When you use either the -Merge or -Graybox switch, the "-Detection Append" switch automatically applies. When you use the "-Detection Append" switch, you must also include either the -Merge or -Graybox switch.

  The Replace literal replaces the n-detection number of the fault in the internal fault list with the n-detection value of the corresponding fault loaded from the MTFI file. When you use the "-Detection Replace" switch, you must also use the -Retain switch.

- -KEEP_Patterns

  An optional switch that specifies that the tool preserves the current internal patterns while loading faults.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

When this switch is set, the new faults loaded are not fault graded by the existing internal patterns, so the new fault status is not reflected by the internal pattern set.

- -ALLOW_fault_sampling

  An optional switch that enables fault sampling when loading faults.

  By default, the read_faults command ignores fault sampling rate data unless you specify this switch. If the fault sampling rate is other than 100 percent, and you do not specify the -ALLOW_fault_sampling switch, then the tool resets the sampling rate; for subsequent commands, you must re-specify the sampling rate.

  Note

  The behavior for the -Delete or -DELETE_equiv switches remain unchanged irrespective of whether the you use this switch or not.

- -RETain

  An optional switch that specifies that the tool retains the fault class of each fault in the fault list.

- -Delete

  An optional switch that specifies that the tool removes all the *filename* faults delete_faults.

- -DELETE_Equivalent

  An optional switch that specifies that the tool removes*filename* faults from the current fault population within the tool's session, and to remove all the faults that are equivalent to those in *filename*.

- -Merge

  An optional switch that merges the loaded external faults with the internal fault list. This enables you to combine the fault lists from different ATPG sessions. For example, if you partition your design and run ATPG for each partition, you can get chip-level fault statistics by merging the partition fault lists. Also, for a single design or design partition, you can run ATPG once, then you can target more undetected faults by incrementally running ATPG using different settings and merging fault lists. For more information, see Example 2.

  If a fault being loaded does not exist in the internal fault list, it is added to the internal fault list.

  If a fault being loaded already exists in the internal fault list, the fault class of the existing fault is updated with the highest priority fault class. For example, if the fault being loaded has a DI class and the matching fault in the internal list has a PT class, the class of the fault in the internal list is updated to DI because DI has a higher priority than PT.

  Fault classes are prioritized as follows:

  DR > DS > DF > DI > TI > BL > PT > PU >OS > HY > UO > UC > AU > RE > UU

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

When performing fault merging, if the faults being merged are for a module that is currently grayboxed, do not run create_patterns or identify_redundant_faults after the fault merging process. If you use create_patterns or identify_redundant_faults, the incorrect RE classification resumes due to incomplete circuit information.

-NO_Warnings

An optional switch that suppresses warnings about conflicting untestable fault classes while merging fault lists. If a fault being loaded has a different untestable fault class than a matching fault in the internal fault list, the fault class defined in the internal list is used.

- -Power_check {ON | Off}

  An optional switch and literal that control the loading of faults in a power-off domain. The default is Off. This switch is applicable only after you have loaded a CPF or UPF file.

  ON — A literal that loads the faults on a power-on domain and discards the faults in a power-off domain. Use this option to calculate the overall test coverage, including the faults that are not active in the current active power mode.

  Off — A literal that enables all the faults to be loaded without checking the power mode.

- -SHow_warnings

  An optional switch that enables the tool to report errors at each line of the file when reading faults. By default, the command ignores errors encountered while reading the fault list and summarizes the errors at the command's completion.

- -FAULT_type {stuck | iddq | transition | toggle | path_delay | bridge | udfm}

  An optional switch and literal string that specify the fault type. The literal string fault type is one of these fault types:

  - stuck
  - iddq
  - toggle
  - transition
  - path_delay
  - bridge
  - udfm

  This switch is required if the *filename* is omitted.

- -MODE *mode_name*

  An optional switch and string that specify the mode to read faults for. The mode name is captured, along with its type, in the Tessent Core Description (TCD) file, which is used for retargeting or core mapping.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the -mode switch is not specified, the tool treats it as unspecified and searches the TSDB for one single file with a mode that matches the specified criteria for the command. The tool issues an error and prompts for additional criteria (-mode) if multiple files with different mode names match the specified criteria.

This switch cannot be used if you specify the filename.

For more information on modes, refer to "Core Mapping for ATPG Process Overview" in the *Tessent Scan and ATPG User's Manual*.

- -PATtern_id *pattern_id*

  An optional switch and string that specify the *pattern_id* used when saving the faults to the TSDB directory. The *pattern_id* differentiates the multiple pattern sets for a given mode name of a given design view.

  If you do not specify this switch, the tool treats it as unspecified and searches the TSDB for one single file that matches the specified criteria for the command.

  If multiple files match the specified criteria, the tool issues an error and prompts for additional criteria to load the pattern file you want. If one of the matching files does not have a pattern ID and you want to select it, specify -pattern_id with the empty ("") string.

  This switch cannot be specified if you use the filename.

- -DESign_id *design_id*

  An optional switch and string that specify the *design_id* used when creating the *logic_test_cores* TSDB sub-directory. When it is used, the tool uses the specified ID. If you do not specify this switch, the tool treats it as unspecified and searches the TSDB for one single file that matches the specified criteria for the command.

  The *design_id* defines the design view used and is typically used to differentiate the pre-layout versus post-layout design views so you can create post-layout patterns without losing pre-layout copies.

  Do not confuse *design_id* with *mode_name* and *pattern_id*. The *design_id* is set by read_design when reading a design. It also is set after elaboration on the current design to the value returned by the"get_context -design_id" command when the value is not null (when the design is loaded with read_verilog command rather than read_design command).

  If multiple files match the specified criteria, the tool issues an error and prompts for additional criteria to load the fault file you want. If one of the matching files does not have a design ID and you want to select it, specify -design_id with the empty ("") string.

  This switch cannot be specified if you use the filename.

- -SIlent

  An optional switch that suppresses the reporting of auto-loaded files. By default, this command reports auto-loaded files as they are read.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example modifies the current fault population in the tool with the contents of an external fault file, *my_detected_faultlist*. Assume this file is the detected fault list from an earlier session in which patterns were generated and saved to disk along with the detected faults.

The example reads in the detected faults from the fault file, restricting the tool to generating patterns for only undetected faults.

```
set_system_mode analysis
add_faults -all
read_faults my_detected_faultlist -merge
report_faults -all
create_patterns
```

Example 2

The following example incrementally runs ATPG in two different modes, combining the results of each session, to target additional undetected faults and increase coverage:

Dofile 1

```
// set up design to run ATPG in mode 1
add_scan_groups grp1 testproc
add_scan_chains chain1 grp1 scan_in1 x_out[0]
add_scan_chains chain2 grp1 scan_in2 scan_out1
add_scan_chains chain3 grp1 scan_in3 scan_out2
add_clocks 0 rst
add_clocks 0 clk
add_input_constraints /G -C1

// generate patterns from mode 1 and write out faults
set_system_mode analysis
create_patterns
write_faults mode1.fault -replace
```

Dofile 2

```
// set up design to run ATPG in mode 2
add_scan_groups grp1 testproc
add_scan_chains chain4 grp1 scan_in4 y_out[2]
add_scan_chains chain5 grp1 scan_in5 x_out[5]
add_clocks 0 rst
add_clocks 0 clk
add_input_constraints /G -C1

// generate patterns from mode 2 and write out faults
set_system_mode analysis
read_faults mode1.fault  -merge
reset_au_faults -reclassify
create_patterns
report_statistics
write_faults modes_1_and_2.fault -replace
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

The following example illustrates the reuse of a faults file in a case that needs the -instance switch.

Tip

As a reminder, the read_faults command automatically adds the faults as well, assuming that they reference faultable sites. The read_faults command should never be followed by the "add_faults -All" command; that sequence of commands is equivalent to only issuing "add_faults -All" and is never what you intend. If you do not issue the add_faults command and only issue the read_faults command, create_patterns targets only the faults loaded, which is the correct result.

run 1: The user has an initial design that is intended to be part of a larger design. This initial design has a special subset of all stuck-at faults that are created via nofaulting, deleting faults, or both. This fault set needs to be reused when this initial design is reused in the larger design. In this run, we assume that the user works and obtains adequate fault coverage. In order to be able to reuse the faults, the faults are written out to a file *stuck_file*.

run 2: The user has embedded the initial design from run 1into a larger design; the instance of the initial design is named alu1. The user now wants to reuse the *stuck_file* fault file (created in run 1) to define the faults within that particular instance (alu1) of the new larger design.

Assume that a PI of the alu is named inA. This is no longer a faultable site once embedded (no longer a top level I/O pin, or on a model boundary).

read_fault_sites stuck_file -instance alu1 -verbose

```
// Loading faults  for instance alu1.
// Note: Original fault /inA is no longer a faultable site.
// Could not load alu1/inA
// Note: 1 fault not loaded because no longer a faultable site.
```

The first note in the output is omitted if -Verbose is not issued. The summary line in the second note is always issued to indicate how many were not loaded and added due to no longer being a faultable site, unless none were rejected for that reason.

Assume that a fault in the file dff_32/Q was loaded and therefore added, but the user wants to delete that fault. Because it is inside alu1 once loaded, that must also be used in the reference to unambiguously identify the fault.

delete_faults alu1/dff_32/Q -s 0

Example 4

The following example modifies the current fault population by merging in the fault classifications of an external fault file, *my_detected_faultlist*. Assume this file is the detected fault list from an earlier session in which patterns were generated and saved to disk along with the detected faults.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_system_mode analysis
add_faults -all
read_faults my_detected_faultlist -merge -power_check on

Example 5

The following example reads the faults for instance ELT_edt_aux_I1from the .flt file, and treats that instance as a graybox:

read_faults det1_all.flt -instance ELT_edt_aux_I1 -graybox -verbose

Example 6

The following example updates the faults in the current fault list with the contents of the external UDFM file *c090_std_io.udfm* and displays parsing errors and warnings:

read_faults /home/design/c090_io/c090_std.io.udfm -verbose

```
//  Warning: UDFM - File c090_io.udfm, line 31 - Cell definition
//  u_and2 does not exist in the circuit.
```

Example 7

This example demonstrates reading faults from the TSDB created with the write_tsdb_data command. The read_faults command specifies the fault type, instance, and mode.

read_faults -fault_type stuck -instance my_instance -mode internal -silent

Example 8

This example demonstrates reading faults from the TSDB created with the write_tsdb_data command. The read_faults command specifies the fault type, instance, mode, and pattern ID.

read_faults -fault_type stuck -instance my_instance -pattern_id pat_id1 -mode internal -silent

In this case, multiple fault files match the specified criteria. The tool reports this message:

```
// Error: There are multiple fault files found in the opened TSDBs for
// instance 'my_instance' (core: 'my_core') with fault type 'stuck' and
// with pattern ID 'pat_id1' and with mode 'internal':
//         Fault file 'my_core_internal_stuck.faults.gz' in
//           TSDB './tsdb_outdir' with design ID ''
//         Fault file 'my_core_internal_stuck.faults.gz' in
//           TSDB './tsdb_outdir2' with design ID 'design_id'
//
//         Please specify the design ID with the -design_id switch.
```

Example 9

This example demonstrates how you resolve the error produced in Example 8. As stated in the error message, the specified criteria matches two fault files with different design IDs, design ID ‘’ and ‘design_id’.

**read_faults -fault_type stuck -instance my_instance -pattern_id pat_id1 **
-design_id "" -mode internal -silent

The design ID with the empty string is specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 10

This example shows the reporting of the auto-loaded fault file when you do not specify the -silent switch.

read_faults -fault_type stuck

```
//  Reading fault file ./tsdb_outdir/logic_test_cores/
small_design.logic_test_core/small_design.atpg_mode_internal/
small_design_internal_stuck.faults.gz
```

Related Topics

add_faults

delete_faults

read_cpf

read_upf

report_faults

report_testbench_simulation_options

set_attribute_value

set_fault_mode

set_fault_sampling

set_fault_type

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_flat_model

Context: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: setup

Loads a flat model into the tool from a specified filename or from the Tessent Shell database (TSDB).

Usage

read_flat_model [*design_path*] [-MODE *mode_name*] [-DESIGN_Id *design_id*]
[-DESIGN_Name *design_name*] [-enable_full_introspection {on | off}]
[-discard_timing_exception_data {on | off}] [-SIlent]

Description

Loads a flat model into the tool. The tool can only read in one flat model at a time.

If an ICL description is required to process the test_setup procedure or the test_end procedure stored in the flat model, then the appropriate ICL files must be read prior to the flat model.

If the flat model filename ends with *.gz* or *.Z*, the tool automatically uncompresses the file with the GNU gzip utility. See set_gzip_options for more details.

Using TSDB to Locate the Flat Model

If you used the write_tsdb_data command, it is not necessary to supply the *design_path* because the information can be located in the TSDB using at least one of these switches:

- -mode
- -design_id
- -design_name

Using the TSDB and specifying one of these switches eliminates the need to remember the location of the flat model file.

Environment and Tcl Variables Stored in the Flat Model

When reading in a flat model, the tool only restores an environment variable if it is not already set in the tool. That is, existing environment variables are not overwritten.

Similarly, when reading in a flat model, the tool only restores a Tcl variable if it is not already set in the tool. That is, existing Tcl variables are not overwritten with the exception of an existing “tessent_user_arg” variable that has no value. For array variables, the tool checks each individual array element.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Reading Password Protected Flat Model Files

When reading in a flat model saved using the "write_flat_model -password *password*" command, the tool requires that you enter the password provided with the switch when the model was saved. For batch runs, you can set the TESSENT_FLAT_MODEL_PASSWORD environment variable to the specified password. The tool obtains the password from the environment variable instead of querying for it. However, if the password setting of this variable is incorrect, the tool reverts to querying you for it.

Arguments

- *design_path*

  A optional string that specifies the pathname of a flat model. If it is not specified, the tool automatically searches for a matching flat model file in the opened TSDB for the current design, if one of the following switches is used: -mode, -design_id, and -design_name.

  The automation for read_flat_model is available for the top-level design only.

- -MODE *mode_name*

  An optional switch and string that specify the mode for which the flat model is read. The mode name is captured, along with its type, in the Tessent Core Description (TCD) file, which is used for retargeting or core mapping. See "Tessent Shell Data Base (TSDB)" on page 4807.

  If the -mode switch is not specified, the tool treats it as unspecified and searches the TSDB for one single file with a mode that matches the specified criteria for the command. The tool issues an error and prompts for additional criteria (-mode) if multiple files with different mode names match the specified criteria.

  This switch cannot be used with *design_path*.

  For more information on modes, refer to "Core Mapping for ATPG Process Overview" in the *Tessent Scan an ATPG User's Manual*.

- -DESIGN_Id *design_id*

  An optional switch and string that specify the *design_id* used when creating the *logic_test_cores* TSDB sub-directory. When -design_id is used, the tool uses the specified ID. If you do not specify this switch, the tool treats it as unspecified and searches the TSDB for one single file that matches the specified criteria for the command.

  The *design_id* defines the design view used and is typically used to differentiate the pre-layout versus post-layout design views so you can create post-layout patterns without losing pre-layout copies.

  Do not confuse *design_id* with *mode_name* and *pattern_id*. The *design_id* is set by read_design when reading a design. It also is set after elaboration on the current design to the value returned by the "get_context -design_id" command when the value is not null (when the design is loaded with read_verilog command rather than read_design command).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If multiple files match the specified criteria, the tool issues an error and prompts for additional criteria to load the flat model you want. If one of the matching files does not have a design ID and you want to select it, specify -design_id with the empty (“”) string.

This switch cannot be used with *design_path*.

- -DESIGN_Name *design_name*

  An optional switch and string that specify the design name for which the flat model should be read. The tool uses the specified design name to do module matching (see set_module_matching_options) against the *design_name* of the file and only reads the flat model file in the TSDB that matches. This can be used when the current design is not set.

  This switch cannot be used with *design_path*.

- -enable_full_introspection {on | off}

  An optional switch and literal that enable full introspection of the flat model. The default is on, which means the introspection commands (for example, get_modules, get_instances and so on) can be used.

  This switch is disabled in the scan_diagnosis context.

  For flat models created with pre-v2015.3 tool versions, full introspection may not be enabled.

- -discard_timing_exception_data {on | off}

  An optional switch and literal that specifies whether the tool should keep the timing exception data when loading the flat model. The default is off. All timing exception data in the flat model is kept by default. When “-discard_timing_exception_data on” is specified, the tool discards the timing exception data to reduce the memory overhead.

  This option reduces the memory footprint of the flat model to improve throughput when running Tessent Diagnosis.

  Because the timing exceptions do not get considered, do not use this option during ATPG when the tool creates new patterns or simulates existing patterns. In this case, simulation mismatches can result.

- -SIlent

  An optional switch that suppresses the reporting of auto-loaded files. By default, this command reports auto-loaded files as they are read.

Examples

Example 1

The following example loads a flat model:

```
read_flat_model /u/my_account/designs/model1
```

Example 2

This example demonstrates reading a flat model from a TSDB created with the write_tsdb_data command. The read_flat_model command specifies the design name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_flat_model -design_name small_design -silent

```
//  Error: There are multiple flat model files found in the opened TSDB
//  for the current design with design name 'small_design':
//           Flat model file 'small_design_internal.flat.gz' in
//             TSDB './tsdb_outdir' with mode 'internal'
//           Flat model file 'small_design_my_mode.flat.gz' in
//             TSDB './tsdb_outdir' with mode 'my_mode'
//
//           Please specify the mode with the -mode switch.
```

In this case, the tool reports that it finds multiple flat model files in the TSDB for the current design with the design name small_design. The files have different mode names: “internal” and “my_mode”

Example 3

This example demonstrates how to resolve the error reported in Example 2. The message asks for the mode to be specified with the -mode switch. In this case, mode “internal” is specified.

read_flat_model -design_name small_design -mode internal -silent

Example 4

This example demonstrates reading the flat model from a TSDB created with the write_tsdb_data command. The read_flat_model command specifies the design ID.

read_flat_model -design_name small_design -design_id my_design_id -silent

Example 5

This example demonstrates reporting the auto-loaded flat model when you do not specify the -silent switch.

read_flat_model -design_name small_design

```
//  Reading flat model file ./tsdb_outdir/logic_test_cores/
small_design.logic_test_core/small_design.atpg_mode_internal/
small_design_internal.flat.gz
```

Related Topics

delete_cell_library

delete_design

read_verilog

set_current_design

set_design_macros

write_flat_model

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_icl

Context: all contexts

Mode: setup

Reads ICL files into the internal ICL database.

Usage

read_icl ***filename_list*** [-bsdl_file *bsdl_file*] [-skip_child_blocks] [-force]

Description

Reads ICL files into the internal ICL database.

Note

If you want to read the ICL file located in one of the dft_inserted_designs directories, use the read_design command, which has a -design_identifier *design_identifier* switch-value pair. You only need to specify this option when the design is not already elaborated and the Tessent Shell Data Base (TSDB) contains multiple dft_inserted_designs views for the design_name you want to read in. Use the -no_hdl switch to only read the ICL view and not overwrite the design view.

Arguments

- ***filename_list***

  A required list of strings that specifies the names of the ICL files to load. The names of the files can contain wildcards. You can define the list of files to read into a Tcl list and use it to load the files.

  If you have used the command, child modules found instantiated into the currently read modules are searched for in the specified -y directories and the -v files.

- -bsdl_file *bsdl_file*

  An optional value pair that defines the name of a BSDL file. The option is only needed and used when reading an ICL module containing an AccessLink of STD_1149_1_2001 element. The BSDL file must define an “Entity entity_name is” clause where entity_name is the one specified in the BSDLEntity property of the AccessLink wrapper. Refer to the Examples section of AccessLinkfor an example of the -bsdl_file usage.

- -skip_child_blocks

  An optional switch, which causes the read_icl command to ignore all ICL modules that are marked as belonging to child sub and physical blocks. Refer to the figure and explanation in the read_design command description for complete usage information.

- -force

  An optional switch that suppresses the normal error message that is generated when the read_icl command is issued after the ICL elaboration has happened. With the -force option,

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

the ICL elaboration tree is discarded instead and must be recreated afterward using the set_current_design command.

Return Values

None

Examples

In the following example, the first read_icl command reads a set of ICL files specified as multiple strings separated by spaces and surrounded by braces. The second read_icl command reads all files matched by the glob command. The third read_icl command reads all filenames contained within the Tcl list icl_list. In all of these cases, undefined ICL modules found within the modules inside the read ICL files are searched for in the specified icl_src directory.

```
set_design_source -format icl -y icl_src -extension icl
read_icl {chip.icl tdr.icl tap.icl block1.icl block2.icl}
read_icl [glob icl/*.icl]
set icl_list [list chip.icl tdr.icl tap.icl block1.icl block2.icl]
read_icl $icl_list
```

Related Topics

delete_icl_modules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_lef

Context: dft, patterns

Mode: all modes

Prerequisites: LEF file

Reads in the LEF file so that the tool can extract the dimensions of the module.

Usage

read_lef ***lef_filename***

Description

This command reads in a LEF file so that the tool can extract the lower left and upper right coordinates of the rectangular boundary of the module and store them as attributes of the module. This information is useful when combined with the information in extracted using the read_def command to precisely localize the module.

The tool stores the (x,y) coordinates of each instance in the following coordinate attributes.

- def_xl_coordinate — Lower left x coordinate
- def_yl_coordinate — Lower left y coordinate
- def_xh_coordinate — Upper right x coordinate
- def_yh_coordinate — Upper right y coordinate

All of the coordinates are floating point values in microns (um).

Arguments

- ***lef_filename***

  A required string that specifies the name of the LEF file.

Related Topics

read_def

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_liberty

Context: unspecified, all contexts

Mode: setup

A command to read a Liberty file to extract attribute information to be populated onto read_cell_library models or read_verilog modules.

Usage

read_liberty ***liberty_filename***
[-populate_power_ground {on | off}]

Description

A command to read a Liberty file to extract attribute information to be populated onto read_cell_library models or read_verilog modules. For more information, see "Create Tessent Insertion Attributes Using Liberty."

The cell attributes gathered from read_liberty are populated onto the read_cell_library model of the same name. This occurs when the populated cell information is needed, which is when the first write_cell_library, get_dft_cell, or set_current_design command is issued. It is advised that you make a separate read_liberty, read_cell_library, write_cell_library run as in the example below, then examine any warnings and other messages produced, and possibly even edit the cell library output by that command (for example, to move a scan_equivalent to the front of the scan_equivalents list because only the front scan cell is used to replace a nonscan cell during test insertion). If you must make edits manually, it is advisable to reiterate read_cell_library and write_cell_library, to ensure the manual edits were done correctly. For iterations, read_liberty should be omitted, because the cell attribute information from that command is already populated onto the models/cells written out by write_cell_library.

Arguments

- ***liberty_filename***

  The name of a Liberty cell attributes file.

- -populate_power_ground {on | off}

  An optional switch and literal pair that specifies whether to include the pg_pin power ports when they are specified in the liberty file with the primary_power or primary_ground pg_type. These port attributes populate onto existing read_cell_library model ports of the same name.

Examples

Example 1

```
read_cell_library tessent_cell.tcellib
read_liberty liberty.lib
write_cell_library tessent_cell.tcellib
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example shows the usage of the "-populate_power_ground" switch. The Liberty cell attributes file "liberty.lib" contains the standard definition format for the power ports VDD and GND:

```
pg_pin (VDD) {
  voltage_name : VDD ;
  pg_type : primary_power ;
}
pg_pin (GND) {
  voltage_name : GND ;
  pg_type : primary_ground ;
}
```

You read the Liberty cell attributes file and include any specified pg_pin power ports with a pg_type of primary_power or primary_ground:

read_liberty liberty.lib -populate_power_ground on

The pg_pin power port attributes populate onto existing read_cell_library model ports of the same name:

```
module SDFCND1T6LVT (SI,D,SE,CP,CDN,Q,QN,GND,VDD);
 input SI ;
 input D ;
 input SE ;
 input CP ;
 input CDN ;
 output Q ;
 output QN ;
 input GND;
 input VDD;
  .
  .
  .
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_modelfile

Context: all contexts

Mode: setup

Initializes the specified RAM or ROM gate using the memory states contained in the named modelfile or deletes an earlier modelfile assignment.

Usage

read_modelfile {***modelfilename*** | -Delete} ***RAM/ROM_instance_name***

Description

The read_modelfile command sets the initial memory states of a RAM or ROM gate using the data that you provide in a modelfile. You can create a modelfile from within the library cell or by using the write_modelfile command. The modelfile must contain initialization data that is in the Tessent modelfile format.

You specify the RAM or ROM gate that you want to initialize by using its instance name. An error condition occurs if the instance contains multiple RAM/ROM gates. When you issue the command, the flat model may not be available, so the tool checks for the correctness of the instance name and the modelfile name during rules checking.

You may also initialize memory states of a RAM or ROM gate by specifying the modelfile from within the RAM or ROM model description. To do so, use the init_file attribute. When using the init_file attribute, you must specify the full pathname to the initialization file if the file is not located in the directory from where you invoked the tool. For more information about modeling RAMs and ROMs and the init_file attribute, refer to the "RAM and ROM" subsection of the *Tessent Cell Library Manual*.

Modelfile Format

A Tessent modelfile contains addresses and data. You must present the addresses in hexadecimal or binary format. You can specify a range of addresses such as 0-1f. An address range may contain an asterisk (*) wildcard character. For example, to specify that you want all addresses set to a hexadecimal F, use "* / f;". You cannot use an X in an address.

To add comments to a RAM or ROM initialization file, precede the comment text with two forward slashes (//). For example:

```
0/3;
// here is a comment
1/2;
```

You can present the data in either binary or hexadecimal format; the default is hexadecimal. To specify data in binary format, you must add a '%' to the beginning of the data values. If you use an X within hexadecimal data, all four bits that it represents are Xs. Therefore, to set a single bit to X, use the binary format.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following two examples are equivalent. The first example shows both an address and its associated data in hexadecimal. The second example shows the same address and data, but the data is now shown in binary.

```
ABCD / 123X;

ABCD /%000100100011XXXX;
```

The following is an example of what an initialization file may look like (range 0-1f):

```
0/ a;
1-f / 5;
10 / 1a;
11-1f / a;
```

Assuming that the ROM has 5 data bits per word, the least that can fit the largest data of “1a”, following this example, the contents of the memory would be:

```
Word (MS-LS)/  Data (MS-LS)
00000       /  01010       // Created by "0/a;"
00001       /  00101       // Created by "1-f/5;"
00010       /  0101
00011       /  00101       ...
...
01111       /  00101       // Created by "1-f/5;"
10000       /  11010       // Created by "10/1a;"
10001       /  01010       // Created by "11-1f/a;"
10010       /  01010       ...
...
11111       /  01010       // Created by "11-1f/a;"
```

You can use an asterisk (*) for an address range. For example, you could rewrite the previous initialization file as shown here. In this example, the first line assigns the data value “a” to the full address range 0-1f. The subsequent lines overwrite the “a” data value with the new data values for the specified addresses.

```
* / a;
1-f / 5;
10 / 1a;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Making the same assumption as the previous example, following this example, the contents of the memory would be:

```
Word (MS-LS)/  Data (MS-LS)
00000       /  01010    // Created by "*/a;"
00001       /  00101    // Created by "1-f/5”; (replaces 'a' from */a)
00010       /  00101
00011       /  00101    ...
...
01111       /  00101    // Created by "1-f/5”; (replaces 'a' rom */a)"
10000       /  11010    // Created by "10/1a”; (replaces ‘a' from */a)"
10001       /  01010    // Created by "*/a;"
10010       /  01010    ...
...
11111       /  01010    // Caused by "*/a;"
```

Arguments

- ***modelfilename***

  A string that specifies the pathname to and filename of the modelfile containing the RAM or ROM initialization data in the Tessent modelfile format.

- **-Delete**

  A switch that deletes the modelfile initialization assignment you made previously for the specified RAM or ROM gate instance. Use this switch to undo an incorrectly specified modelfile or instance name (due, for example, to a misspelling or a non-existent modelfile or instance).

  Note

  If you misspelled the RAM/ROM_instance_name when you first assigned the modelfile, you need to use the same spelling when you delete the assignment. After you remove the incorrect modelfile initialization assignment using the -Delete switch, you can re-issue the read_modelfile command using the corrected modelfile or instance name.

- ***RAM/ROM_instance_name***

  A required string that specifies the instance name of the RAM or ROM gate that you want to initialize, or for which you want to delete a previous modelfile initialization assignment.

Examples

Example 1

The following example initializes the memory states of a RAM gate, then performs an ATPG run:

```
read_modelfile model.ram /p2.ram
set_system_mode analysis
create_patterns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the memory primitive specified in the read_modelfile command did not exist, then the set_system_mode analysis command would result in an error message similar to the following:

```
//  Error : /p2.ram given in an earlier read_modelfile command is NOT a
//  RAM or ROM primitive.
```

Example 2

The next example fixes the error by using the -Delete switch to remove the incorrect entry from the tool's internal list of RAMs and ROMs to be initialized. The example then re-issues the original command using the correct instance name (assumed to be /p1.ram):

```
read_modelfile -delete /p2.ram
read_modelfile model.ram /p1.ram
```

Related Topics

create_initialization_patterns

write_modelfile

set_ram_initialization

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_patterns

Context: dft -edt, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting, patterns -failure_mapping

Mode: analysis

Loads the external pattern set from the specified file or from the Tessent Shell database (TSDB).

Usage

read_patterns [*filename*] [-append]
[-fault_type {stuck | iddq | transition | toggle | path_delay | bridge | udfm}]
[-mode *mode_name*] [-pattern_id *pattern_id*]
[-design_id *design_id*] [-SIlent]

For patterns -scan_retargeting:
[-modules *module_objects* | -instances *instance_objects*]

All other contexts' specific options:
[-mask_file *mask_filename* [-maskfile_pattern_offset *number*]]
[-preserve_masked_observe_points]

Description

For pattern retargeting, this command extracts the core name and mode from the pattern file and automatically associates the patterns with the correct core and mode.

This command appends to or replaces the external pattern set, and optionally provides additional observation masking information.

Provide the patterns by specifying the pattern filename, or if you used the write_tsdb_data command during pattern generation, specifying one of these switches, instead of the pattern filename, eliminates the need to remember the location of the pattern file and enables the tool to locate the patterns in the TSDB:

- -modules
- -instances
- -mode
- -fault_type
- -pattern_id
- -design_id

If read_patterns is issued in the "patterns -scan retargeting" context without a filename, the tool searches for pattern files for all of the instantiated TCD cores (added with add_core_instances), with a *design_name* equal to the core module name of those same cores and with a corresponding mode.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If read_patterns is issued in the "patterns -scan" context without a filename, the tool searches for a pattern file for the current design. The tool infers the core name from the current design's module name by performing module matching (see set_module_matching_options) and treats the mode name as unspecified. The tool issues an error and prompts for additional criteria if multiple files with different mode names match the specified criteria. See Example 5. For more information, refer to logic_test_cores in the Tessent Shell Data Base (TSDB) section.

Error Checking for STIL and WGL Patterns

When reading STIL and WGL patterns back into the Tessent tools for any flow, there are multiple error checks that are performed to see if the pattern file matches the design data in the tool. These error checks include comparing the shift vector to the shift procedure, the load_unload vectors to the load_unload procedure, the test_setup vectors to the test_setup procedure, comparing scan chain information by number of chains/channels and names of chains/channels, comparing signal names in the pattern files to port names on the design, and comparing asynchronous clock information in the pattern file to clocks defined with -period. Mismatches in any of these checks produce an error or a warning, and a statement that this likely indicates an inconsistency between the flat model or design and the pattern file.

The following provides a selection of messages the tool may produce when reading STIL and WGL patterns:

- For a signal defined in the pattern file that is not present in the design, the following message is issued. Note, for signals missing in the pattern file there is no message, as the ALL_EXCLUDE_UNUSED parameter keyword can remove signals from the pattern file that are not needed. Example message:

```
// Error:  The signals defined in the pattern file do not match the
//         ports from the design.
//            First signal not found is '<name>'.
//            This may indicate an inconsistency between the flat
//            model and the pattern file being read.
```

- For asynchronous clocks defined in the pattern file that are not present in the design the following message is issued. Example message:

```
// Error:  The async clocks defined in the pattern file do not match
//         the clocks with -period from the design.
//            First clock not found is '<name>'.
//            This may indicate an inconsistency between the flat
//            model and the pattern file being read.
```

- If the number of asynchronous clocks in the pattern file is less then what is defined in the design, the following message is issued. Example message:

```
// Error:  The number of async clocks (<int>) in pattern file does
//         not match the number of clocks (<int>) defined with
//         -period.
//            This may indicate an inconsistency between the flat
//            model and the pattern file being read.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- If the number of chains or channels in a pattern file does not match those in the design, the following is issued. For reverse fail mapping, this is a warning. Example message:

```
// Error:  Scan chain information in pattern file does not match
//         traced chains.
//           <Int> scan chains in pattern file, <int> traced.
```

- Message given for other attributes of scan chains that do not match.

```
// Error:  Scan chain information in pattern file does not match
//         traced chains.
//           Chain length of <int> does not match traced length
//           <int>.

// Error:  Scan chain information in pattern file does not match
//         traced chains.
//           Chain <name> from pattern file not found.
```

- Message for a mismatch between vector data for the procedures in the pattern file to the design procedures stored in the tool. For test_setup it is just a warning for padding of test_setup or removal of test_setup from pattern data.

```
// Error:  Pattern file data does not match shift procedure at line
//         <int>.

// Error:  Pattern file data does not match load_unload procedure at
//         line <int>.

// Warn:  Pattern file data does not match test_setup procedure.
```

- Message for when a scan chain uses bidirectional signals for both scan in and scan out, and the wrong (swapped) signals are specified for input and output in the STIL file.

```
//  Error: Scan chain information in pattern file does not match
//         traced chains.
//         Chain <name> from pattern file has incorrect scan in
//         signal.
```

Multithreading Support

In the “patterns -scan_retargeting” context, the read_patterns command supports multithreading when reading pattern database (PatDB) files for retargeting. Multithreading is available when you have added processors using the add_processors command.

Arguments

- *filename*

  A string that specifies a file containing the patterns to be read. It is used when you are reading patterns from a file rather than the TSDB. If the filename ends with *.gz* or *.Z*, the tool automatically uncompresses the file with the GNU gzip utility. See set_gzip_options for more details.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This file is typically created with the write_patterns command. The patterns can be in ASCII, binary, pattern database (PatDB), STIL, or WGL test pattern format.

The tool obtains masking information from the patterns read in, applying masks wherever Xs appear in a pattern's unload and measure_po values. Even when known values are determined after simulation, the tool continues to mask those locations.

If you used the write_tsdb_data command during pattern generation, the patterns are located in the TSDB. Specifying one of these switches eliminates the need to remember the location of the pattern file and enables the tool to locate the patterns in the TSDB: -mode, -fault_type, -design_id, -pattern_id, -instances, and -modules.

The filename cannot be used if you specify any of these switches: -mode, -fault_type, -design_id, and -pattern_id, -instances, and -modules,.

- -append

  An optional switch that appends the new patterns on the existing external pattern set. By default, the command deletes the existing external pattern set and replaces with the new pattern set.

- -mask_file *mask_filename*

  An optional switch that masks certain observation points in external patterns added to the internal pattern set after simulation. This switch enables you to salvage patterns that result in unreliable failures on certain cells at the tester, or that fail verification due to simulation mismatches at a few observation points. Rather than discard failing patterns altogether and accept the accompanying significant reduction in test coverage, you can keep the patterns, use this switch to mask the few problem observation points, and typically see only minimal impact on test coverage.

  Note

  The order of the masking information in *mask_filename* is important. To preserve all masking information for a given pattern, specify all the masking information for that pattern together. Otherwise, masking information is overwritten.

  ```
  // Warning: The existing observation masking information for pattern
  0 is replaced by a new one.
  // Warning: The existing observation masking information for pattern
  1 is replaced by a new one.
  ```

  The tool preserves existing masked observation points (scan cell or primary output) in external patterns loaded into the tool when a mask_file is specified.

  This switch affects the capture values of only those patterns from the external pattern set that the tool has added to the internal pattern set after simulation. Therefore, if you write patterns using "write_patterns -external", they do not include the masking information from the mask file in the saved patterns; the saved patterns are the same as the original patterns.

  To save a pattern set that incorporates the masking information, use the simulate_patterns command to perform a good machine simulation of the external pattern set you specified

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

with “read_patterns -preserve_masked_observe_points”. Then use the write_patterns command without the -External switch to write the resulting internal pattern set to a file.

The Mask File

- The *mask_filename* specifies a mask file, a text file you create that lists the patterns and locations you want masked.

  Use the statements and syntax described in Table 6-2 to create a mask file. Use one statement per line. Precede comment text with a pair of forward slashes (//). An example mask file follows the table.

  The mask file must be in a pattern based format. Most ATE’s return accumulated cycle-based failure data. Use the read_failures command to convert cycle-based failure data to pattern based data for the mask file.

Table 6-2. Mask File Statements

Statement	Usage Rules
<location>	Required. Specifies a location to mask or observe. Use as many location statements as you need. List all location statements that specify a pattern index in ascending order. Syntax format depends on the type statement that precedes the location statement, as follows: • If “type mask” or “type observe” precedes the location statement, use one of these two formats: <pattern_index> <primary_output_name> <pattern_index> <chain_name> <cell_ID> • If “type pin_mask” precedes the location statement, use one of the following three formats: <primary_output_name> <chain_name> <chain_output_pin_name>
type <value>	Optional. Indicates if the locations specified by subsequent location statements are to be masked (forced to X) or observed. Specify one of the following values: • mask — specifies to mask the locations, for a specific pattern • observe — specifies to observe the locations and mask all others, for a specific pattern • pin_mask — specifies to mask the locations, for all patterns. Overrides other type statements that cover the same location. Use as many type statements as you like. By default, the tools assume “type mask” for locations listed before the first type statement. A type statement inserted between two location statements for the same pattern does not take effect until the next different pattern.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Following is an example mask file:

```
//file: my_pattern_mask.ascii
//
// The next 3 lines mask 3 locations in pattern 20.
20 primary_output_12
20 chain2 31
20 chain3 101
// The next line masks 1 location in pattern 25.
25 chain1 120
type observe
// The following 2 lines mask all but the 2 specified locations in
//   pattern 40.
40 chain5 3
40 primary_output_1
// The following type statement takes effect with pattern 61.
type mask
// The next line defines a third observe location for pattern 40.
40 primary_output_2
61 chain4 120
// The next line masks chain5 for all patterns, overriding previous
//   "type observe" for cell 3 of chain 5 for pattern 40, and
//   subsequent "type observe" for cell 99 of chain5 for pattern 65.
type pin_mask
chain5
type observe
// The following line masks all cells in chain5 except cell 99 in
//    pattern 65.
65 chain5 99
```

- -maskfile_pattern_offset *number*

  An optional switch and integer that specify the pattern offset number for the mask file. By default (when no offset is specified), the pattern in the mask file starts with pattern 0. This switch is useful for reducing the number of pattern files that must be loaded when only a subset need to be masked. For more information, refer to Example 2.

- -preserve_masked_observe_points

  An optional switch that enables you to preserve existing masked observation points (wherever Xs appear in a pattern's unload and measure_po values) in external patterns loaded into the tool. When you use this switch to preserve existing masked observation points in the external patterns, the internal patterns stored after simulating the external patterns preserve these masked observation points. You can optionally use -mask_file to specify additional observation points to be masked.

- -modules *module_objects*

  An optional switch and module object spec (an object name, a Tcl list of object names, or a collection of objects), only available for use in "patterns -scan_retargeting", that specify modules added as core instances that the tool associates with the patterns. When you use this argument, the tool searches the TSDB for pattern files for the cores of the specified modules. The core and mode names are inferred from the core instantiations (see report_core_instances).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This switch cannot be used if the -instances switch or *filename* is specified.

- -instances *instance_objects*

  An optional switch and instance object spec (an object name, a Tcl list of object names, or a collection of objects), that specify instances added as core instances that the tool associate withs the patterns.When you use this argument, the tool searches the TSDB for pattern files for the cores of the specified instances. The core and mode names are inferred from the core instantiations (see report_core_instances).

  This switch is only available for use in "patterns -scan_retargeting" and cannot be used if the -modules switch or *filename* is specified.

- -fault_type {stuck | iddq | transition | toggle | path_delay | bridge | udfm}

  An optional switch and literal string that specify the fault type. The literal string fault type is one of these fault types:

  - stuck
  - iddq
  - toggle
  - transition
  - path_delay
  - bridge
  - udfm

  This switch is required if *filename* is omitted and cannot be used if the *filename* is specified.

- -mode *mode_name*

  An optional switch and string that specify the mode to read the patterns for. The mode name is captured, along with its type, in the Tessent Core Description (TCD) file, which is used for retargeting or core mapping.

  If -mode is not specified in non-retargeting flows, the tool treats it as unspecified and searches the TSDB for a mode that matches the specified criteria.If the tool finds an instance instantiated with multiple modes and a mode was not specified, it reports an error and prompts for a mode to be specified. See Example 6.

  For retargeting flows, the mode is inferred from the core instances added by the add_core_instances command. If -modules is specified, it uses the instances of the specified modules. If -instances is specified, it uses those instances.

  You cannot use this switch if you specify the filename.

  For more information on modes, refer to "Core Mapping for ATPG Process Overview" in the *Tessent Scan and ATPG User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -pattern_id *pattern_id*

  An optional switch and string that specify the *pattern_id* that was used when saving the patterns to the TSDB directory. The *pattern_id* is used to differentiate the multiple pattern sets for a given mode name of a given design view.

  If you do not specify this switch, the tool treats it as unspecified and searches the TSDB for one single file that matches the specified criteria for the command.

  If multiple files match the specified criteria, the tool issues an error and prompts for additional criteria to load the pattern file you want. If one of the matching files does not have a pattern ID and you want to select it, specify -pattern_id with the empty (“”) string. See Example 7.

  You cannot use this switch if you specify the filename.

- -design_id *design_id*

  An optional switch and string that specify the *design_id* that was used when creating the *logic_test_cores* TSDB sub-directory. When it is used, the tool uses the specified ID. When it is unspecified, the tool treats it as unspecified and searches the TSDB for one single file that matches the specified criteria for the command.

  The *design_id* defines the design view used and is typically used to differentiate the pre-layout versus post-layout design views so you can create post-layout patterns without losing the pre-layout copies.

  Do not confuse *design_id* with *mode_name* and *pattern_id*. The *design_id* is set by read_design when reading a design. It also is set after elaboration on the current design to the value returned by the “get_context -design_id” command when the value is not null (when the design is loaded with read_verilog command rather than read_design command).

  If multiple files match the specified criteria, the tool issues an error and prompts for additional criteria to load the pattern file you want. See Example 4. If one of the matching files does not have a design ID and you want to select it, specify -design_id with the empty (“”) string.

  You cannot use this switch if you specify the filename.

- -SIlent

  An optional switch that suppresses the reporting of auto-loaded files. By default, this command reports auto-loaded files as they are read.

Examples

Example 1

The following example demonstrates how to perform fault grading for stuck-at faults using the transition pattern set created previously. Next, the example shows how to perform top-up ATPG.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set _context patterns -scan
set_system_mode analysis
set_fault_type stuck
add_faults -all
read_patterns pattern_transition
simulate_patterns -store_pattern none
create_patterns
write_patterns pattern_topup.stil -stil

Example 2

The following example demonstrates the use of the -maskfile_pattern_offset switch:

Consider the following mask file, named *mask_data.file*:

```
// The next 3 lines mask 3 locations in pattern 20.
20 primary_output_12
20 chain2 31
20 chain3 101
// The next line masks 1 location in pattern 25.
25 chain1 120
```

The following command reads the patterns from a STIL file, reads the above mask file, applying an offset of 400 in the mask file:

read_patterns pats_400_to_599.stil -mask_file mask_data.file -maskfile_pattern_offset 400
simulate_patterns -store_patterns all
write_patterns pats_400_to_599_masked.stil -stil

The result is that patterns 420 and 425 from the STIL file are masked.

Example 3

The "Retargeting Example" in the*Tessent Scan and ATPG User's Manual* demonstrates the use of this command in a design context.

Example 4

The TSDB *logic_test_cores* directory structure is shown in Figure 6-2.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 6-2. TSDB logic_test_cores Directory Structure

This example and Example 5 have this TSDB *logic_test_cores* directory:

```
tsdb_outdir
    |_logic_test_cores
        |_core1_prelayout.logic_test_core
            |_core1.atpg_mode_internal
                |_core1_internal_stuck_patdId1.patdb
        |_core1_postlayout.logic_test_core
            |_core1.atpg_mode_internal
                |_core1_internal_stuck_patdId1.patdb
            |_core1.atpg_mode_special_mode
                |_core1_special_mode_stuck_patdId1.patdb
```

There are multiple pattern files, with different *design_id*s that match the specified criteria by the read_patterns command. The tool reports an error and prompts for additional criteria:

read_patterns -modules core1 -fault_type stuck -mode internal -silent

```
//  Error: There are multiple pattern files found in the opened TSDBs
//  for module 'core1' with fault type 'stuck' and with mode 'internal':
//            Pattern file 'core1_internal_stuck_patdId1.patdb' in
//              TSDB './tsdb_outdir' with design ID 'prelayout'
//            Pattern file 'core1_internal_stuck_patdId1.patdb' in
//              TSDB './tsdb_outdir' with design ID 'postlayout'
//
//            Please specify the design ID with the -design_id switch.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

To correct this error, specify a *design_id*:

read_patterns -modules core1 -fault_type stuck -mode internal -design_id postlayout -silent

There is a similar error message for multiple pattern files with different pattern IDs.

Example 5

The TSDB directory structure for this example is shown in Example 4. This example demonstrates multiple pattern files with different mode names.

read_patterns -fault_type stuck -design_id postlayout -silent

```
// Error: There are multiple pattern files found in the opened TSDBs for
// the current design with fault type ‘stuck’ and with design ID
// 'postlayout':
//            Pattern file ‘core1_internal_stuck_patdId1.patdb’ in
//              TSDB ‘./tsdb_outdir’ with mode ‘internal’
//            Pattern file ‘core1_special_mode_stuck_patdId1.patdb’ in
//              TSDB ‘./tsdb_outdir’ with mode ‘special_mode’
//
//            Please specify the mode with the -mode switch.
```

To correct this error, specify a mode:

read_patterns -fault_type stuck -design_id postlayout -mode internal -silent

Example 6

In this example, -mode was not specified and the tool finds an instance that has been instantiated with multiple modes:

read_patterns -instances {piccpu_inst} -fault_type stuck -silent

```
//  Error: Instance ‘piccpu_inst’ of module ‘piccpu’ (core: ‘piccpu_core’)
//  has been instantiated with modes ‘internal1’ and ‘internal2’.
//            Please specify the mode with the -mode switch.
```

To correct this error, specify a mode:

read_patterns -instances {piccpu_inst} -fault_type stuck -mode internal1 -silent

Example 7

In this example, there are multiple pattern files in the opened TSDB that match the specified criteria, but with different pattern IDs.

read_patterns -instances [get_instances core_1] -fault_type stuck -silent

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Error: There are multiple pattern files found in the opened TSDB for
//  instance 'core_1' (core: 'piccpu_maxlen16_1', mode: 'internal') with
//  fault type 'stuck':
//         Pattern file 'piccpu_maxlen16_1_internal_stuck.patdb' in
//           TSDB './tsdb_outdir' with pattern ID ''
//         Pattern file 'piccpu_maxlen16_1_internal_stuck_id1.patdb' in
//           TSDB './tsdb_outdir' with pattern ID 'id1'
//
//         Please specify the pattern ID with the -pattern_id switch.
```

To select the pattern file with no pattern ID, use use the -pattern_id with empty string:

read_patterns -instances [get_instances core_1] -fault_type stuck -pattern_id "" -silent

Example 8

This example demonstrates reporting the auto-loaded pattern files when you do not specify the -silent switch:

read_patterns -fault_type stuck

```
//  Reading pattern file ./tsdb_outdir/logic_test_cores/
core1.logic_test_core/core1.atpg_mode_internal/core1_internal_stuck.patdb
//  Reading pattern file ./tsdb_outdir/logic_test_cores/
core2.logic_test_core/core2.atpg_mode_internal/core2_internal_stuck.patdb
//  Reading pattern file ./tsdb_outdir/logic_test_cores/
core3.logic_test_core/core3.atpg_mode_internal/core3_internal_stuck.patdb
//  Reading pattern file ./tsdb_outdir/logic_test_cores/
core4.logic_test_core/core4.atpg_mode_internal/core4_internal_stuck.patdb
```

Related Topics

delete_patterns

simulate_patterns

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_procfile

Context: dft -edt, dft -scan, dft -test_points, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Reads a test procedure file and merges the timing and named capture procedure data it contains with existing data loaded from previous test procedure files.

Usage

read_procfile ***proc_filename***

Description

For complete information on test procedure file creation, syntax, and structure, refer to "Test Procedure File" in the *Tessent Shell User's Manual*.

Note

The procfile can contain Tcl variables that are substituted when the file is read. The referenced Tcl variables must exist in the Tcl global namespace scope. If you set the variables within a proc, use :: as a prefix to the variable name so that it is set in the global namespace. For example, use "set ::my_period 4" so that $my_period exists when processing the proc files. For more info about Tcl variable substitution inside the procfile, refer to "Additional Support for Test Procedure Files" in the *Tessent Shell User's Manual*.

Note

In the patterns -ijtag context, the tool returns an error if the procfile contains a timeplate with the same name as an existing timeplate used in an IJTAG pattern set.

This command does the following:

- Replaces each existing timeplate with the definition in *proc_filename* for that timeplate.
- Adds any new timeplates defined in *proc_filename*.
- Updates the timing (timeplates) used by existing scan procedures (like test_setup, load_unload and shift) to match the definitions of these procedures in *proc_filename*.
- Handles the named capture procedure (NCP) and clock control (CC) data in *proc_filename* according to Table 6-3. An "equivalent" NCP or CC only changes the timing, while a "nonequivalent" one makes changes other than just the timing.

Note

Under certain conditions where this command replaces existing NCPs and CCs with new versions, the tool reports a W43 warning when you run this command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 6-3. Named Capture Procedure and Clock Control Handling Under Different Conditions

Condition	Internal/External Pattern or Patterns	No Internal/External Pattern or Patterns
New NCP	Appends new NCP.	Appends new NCP.
Equivalent NCP	Updates existing NCP's timing only. Does not reactivate a deactivated NCP (for example, from using set_cap_proc off -all).	Updates existing NCP's timing only. Does not reactivate a deactivated NCP (for example, from using set_cap_proc off -all).
Nonequivalent NCP	Produces an error if the NCP is used by any patterns. Otherwise, replaces existing NCP with new NCP and generates a W43 warning.	Replaces existing NCP with new NCP and generates a W43 warning.
New CC	Produces an error.	Appends new CC.
Equivalent CC	Ignores new CC.	Ignores new CC.
Nonequivalent CC	Produces an error.	Replaces existing CC with new CC and generates a W43 warning.

Arguments

- ***proc_filename***

  A required string that specifies the pathname or filename of the test procedure file to read.

Examples

Example 1

Test procedure file *orig.proc* defines a simple timeplate as well as a shift, load_unload, and named capture procedure cap1 and a clock control clk1. The following example uses *orig.proc*, sets the system mode to analysis, then reports the current timeplates, procedures, and clock controls stored in the tool's database (for brevity, only the parts of the timeplates and procedures relevant to this example are shown):

```
add_scan_groups grp1 orig.proc
set_system_mode analysis
report_timeplates

timeplate gen_tp1 =
    ...
end;

report_procedures
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
procedure shift =
   scan_group grp1 ;
   timeplate gen_tp1 ;
   ...
end;

procedure load_unload =
   scan_group grp1 ;
   timeplate gen_tp1 ;
   ...
end;

procedure capture cap1 =
   timeplate gen_tp1 ;
   ...
end;
```

report_clock_controls

```
CLOCK_CONTROL 'clk1' =
   ...
END;
```

Assume a second test procedure file *update.proc* defines a new timeplate in addition to the original one, revises the shift procedure to use the new timeplate, and defines a new named capture procedure cap2 and a new clock control clk2. The following example adds the new timeplate, updates the timing in the scan procedures, and adds the new named capture procedure and clock control.

read_procfile update.proc

```
//  The following occurred at line 46 in file update.proc:
//  Procedure shift updates timing in same procedure from file orig.proc.(W14-1)
//  Loaded new procedure file successfully.
//  1 new capture procedure is added.
//  2 total capture procedures are in the system.
//  2 capture procedures are active for test generation.
```

report_timeplates

```
timeplate gen_tp1 =
    ...
end;
// Timeplate gen_tp1 has an old timeplate
// timeplate gen_tp1 =
//    ...
// end;
 timeplate gen_tp2 =
    ...
 end;
```

report_procedures

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
procedure shift =
   scan_group grp1 ;
   timeplate gen_tp2 ;
   ...
end;
procedure load_unload =
   scan_group grp1 ;
   timeplate gen_tp1 ;
   ...
end;
procedure capture cap1 =
   timeplate gen_tp1 ;
   ...
end;
procedure capture cap2 =
   timeplate gen_tp2 ;
   ...
end;
```

report_clock_controls

```
CLOCK_CONTROL 'clk1' =
   ...
END;
CLOCK_CONTROL "clk2" =
   ...
END;
```

Related Topics

report_procedures

report_timeplates

write_patterns

write_procfile

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_sdc

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis (dft -edt, patterns -scan, patterns -scan_diagnosis and dft -test_points contexts only)

Reads false path and multicycle path information from a Synopsys Design Constraint (SDC) file.

Usage

read_sdc ***sdc_filename***… [{-Instance *ins_pathname*…} | -Module *module_name*] [{> | >>} *file_pathname*]

Description

Typically, you determine false and multicycle paths as part of static timing analysis before running ATPG.

The read_sdc command parses the SDC file, looking for the set_false_path command, the set_multicycle_path command, and other appropriate commands and their arguments. A summary message displays the following information for the relevant commands found in the SDC file:

- Number of false paths — Paths that cannot be activated in the design's functional (at-speed) mode of operation or are untestable due to test constraints you have applied. False paths are derived from the SDC command set_false path, which specifies false path timing (the paths that the STA tool should not evaluate for timing).
- Number of multicycle paths — Paths that have a combined time delay through the path greater than the clock period. False paths are derived from the SDC command set_multicycle_path.
- Number of erroneous paths — False paths that are erroneous (for example, due to misnamed pins).
- Number of case analysis — Conditional false paths, which are false paths along with a Boolean condition. If the associated condition is violated, the effect of the false path is considered. Conditional false paths are derived from the Boolean constraints defined by the SDC command set_case_analysis.
- Number of disable timings — Derived from the SDC command set_disable_timing, which disables the specified timing arcs in a circuit and is used to speed up the runtime of the tool. The set_disable_timing command is virtually equivalent to the set_false_path command.
- Number of clock groups — Clock group count.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Caution

Do not use the read_sdc command to read in SDC files created during the Tessent insertion flow (for example, from MBIST).

The SDC commands set_false_path, set_multicycle_path, set_case_analysis, and set_disable_timing are converted into equivalent add_false_paths commands. For example, the following SDC command set_disable_timing:

```
set_disable_timing [get_pins {U1/buf2/Z}]
```

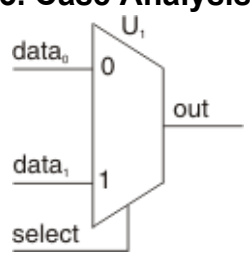
would be translated into the following add_false_paths command:

```
add_false_paths -through U1/buf2/Z
```

The following SDC command set_case_analysis:

```
set_case_analysis 0 [get_pins {U1/select}]
```

Figure 6-3. Case Analysis Example



would be translated into one conditional false path and one unconditional false path:

```
add_false_paths -through U1 if (U1/select !=0)
```

Note

That is, if U_1/select is not 0, the effect of this false path is considered.

```
add_false_paths -through U1/select
```

When test patterns are simulated during ATPG, the related incorrect capture responses are identified and masked (modified to X) in the resultant test patterns, so they are not the source of simulation mismatches when verified in a timing-based simulator.

The read_sdc command supports the following SDC commands:

- create_clock
- create_generated_clock
- set_case_analysis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- set_clock_groups
- set_disable_timing
- set_false_path
- set_hierarchy_separator
- set_multicycle_path

Be aware that the SDC commands create_clock and create_generated_clock are completely independent of the command add_clocks, which you must still use.

Tip

To avoid problems extracting correct paths from the SDC specifications, use the output of the write_sdc command in your synthesis or static timing analysis tool.

Follow the steps outlined in the "Does the SDC File Contain Valid SDC Information?" section of the *Tessent Scan and ATPG User's Manual*.

For more information, see "Pattern Failures Due to Timing Exception Paths" in the *Tessent Scan and ATPG User's Manual*.

Tip

You can use the command to list all of the SDC-defined clocks.

You can use the "set_timing_exceptions_handling -allow_invalid_pin_names on" command to enable the read_sdc command to issue warnings instead of errors when defined objects are not found.

Hybrid TK/LBIST Flow Usage

Reads in the SDC file that describes the false and multicycle paths that should be blocked during logicBIST.

The scan cells that are the destination of false and multicycle paths are inferred from the SDC file. Re-circulating muxes with inversion are inserted on their data input pins to ensure these scan cells do not capture values from timing exception paths. Both false paths and multicycle paths are bound in the same way.

Refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- ***sdc_filename***

  A required, repeatable string that specifies the pathname of one or more SDC files to read.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Instance *ins_pathname*

  An optional switch and repeatable string that specifies the pathname(s) of one or more design instances to which the information in *sdc_filename* should be applied. Use this switch when the SDC information in the file was developed at the module (block) level and you now want to read it in and have it properly applied to the specified instance(s) of that module in the netlist.

- -Module *module_name*

  An optional switch and string pair that specifies a design module or Tessent Cell library model, to all instances of which the information should be applied in the specified SDC file(s). Use this switch to specify when the SDC information in the file(s) was developed at the module (block) or Tessent Cell library model level and that it should be applied to every instance of that module or model in the netlist.

- > *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list for creating or replacing the contents of *file_pathname* with a list of any syntax errors encountered while reading the specified SDC file(s).

- >> *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list for appending to the contents of *file_pathname* a list of any syntax errors encountered while reading the specified SDC file(s).

Examples

Example 1

The following example reads the file *new_design_sdc* containing one false path definition and one multicycle path definition in Synopsys Design Constraint (SDC) format:

read_sdc new_design_sdc

```
// Reading SDC file new_design_sdc ...
// Finished reading SDC file new_design_sdc.
// read_sdc summary: 1 false path, 1 multicycle path, 0 erroneous paths
//    0 disable timings, 0 case analysis, 0 clock groups
```

Tip

If the SDC file contains more than one description of the same path, it is counted only once. Also, the path counts reported in the read_sdc summary apply to the current command only; they are not cumulative counts of the total number of false and multicycle path definitions stored in the internal data base of the tool.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

Continuing with the preceding example, the following example reads another SDC file *latest_design_sdc* containing one false path and three multicycle path definitions, then reports on the false path definitions now stored within the tool:

read_sdc latest_design_sdc

```
// Reading SDC file latest_design_sdc ...
// Finished reading SDC file latest_design_sdc.
// read_sdc summary: 1 false path, 3 multicycle paths, 0 erroneous paths,
//    0 disable timings, 0 case analysis, 0 clock groups
```

report_false_paths

```
// False Path -from_clock clk1 -to_clock clk2
// False Path -from clk2 reset scan_in1 -to y[2] y[1] y[0]
// Total reported false paths = 2
```

Example 3

The SDC files of the two preceding examples could have been read with one command as in the following example:

read_sdc new_design_sdc latest_design_sdc

```
// Reading SDC file new_design_sdc...
// Finished reading SDC file new_design_sdc.
// Reading SDC file latest_design_sdc...
// Finished reading SDC file latest_design_sdc.
// read_sdc summary: 2 false paths, 4 multicycle paths, 0 erroneous paths,
//    0 disable timings, 0 case analyses, 0 clock groups
```

Notice the read_sdc summary lists the combined totals for all the files read with the one command.

Example 4

Suppose the SDC file *new_design_sdc* of the preceding example contained two false path definitions instead of one, but one of the definitions was erroneous due to misnamed pins. The following example shows sample messaging when this error is detected in the SDC file:

read_sdc new_design_sdc latest_design_sdc

```
// Reading SDC file new_design_sdc ...
//  Error: Incorrect pin/instance path name(s) in SDC file new_design_sdc,
//     line 5.
//  Error: No gate found for "all_inputs all_outputs"
// Finished reading SDC file new_design_sdc.
// Reading SDC file latest_design_sdc...
// Finished reading SDC file latest_design_sdc.
// read_sdc summary: 3 false paths, 4 multicycle paths, 1 erroneous path,
//    0 disable timings, 0 case analyses, 0 clock groups
```

report_false_paths

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// False Path -from_clock clk1 -to_clock clk2
// False Path -from clk2 reset scan_in1 -to y[2] y[1] y[0]
// False Path -from B_input -to_clock all_outputs
// Total reported false paths = 3
```

Notice the tool now has the erroneous path definition stored internally. From here, you might want to check for any additional details about this particular definition:

report_false_paths -debug_error -from B_input -to_clock all_outputs

```
// False Path -from B_input -to_clock all_outputs did not find any
//    gate for "B_input all_outputs"
```

Example 5

The following example deletes the erroneous path definition, enters a corrected version of the definition, then confirms there are no remaining errors in any of the false or multicycle path definitions:

delete_false_paths -from B_input -to_clock all_outputs

```
// Deleted 1 false path
```

report_false_paths

```
// False Path -from_clock clk1 -to_clock clk2
// False Path -from clk2 reset scan_in1 -to y[2] y[1] y[0]
// Total reported false paths = 2
```

add_false_paths -from B -to y[2] y[1] y[0]
report_false_paths

```
// False Path -from_clock clk1 -to_clock clk2
// False Path -from clk2 reset scan_in1 -to y[2] y[1] y[0]
// False Path -from B -to y[2] y[1] y[0]
// Total reported false paths = 3
```

report_false_paths -error

```
// Total reported false paths = 0
```

Example 6

The following example reads the SDC file *my_block_sdc* and applies the information it contains to instance U1 of the design module my_block:

read_sdc my_block_sdc -instance U1

```
// Reading SDC file my_block_sdc ...
// Finished reading SDC file my_block_sdc.
// read_sdc summary: 2 false paths, 0 multicycle paths, 0 erroneous paths
//    0 disable timings, 0 case analyses, 0 clock groups
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 7

The following example reads the file of the preceding example and applies the SDC information to instances U2 and U3 of the same design module:

read_sdc my_block_sdc -instance U2 -instance U3

```
// Reading SDC file my_block_sdc ...
// Finished reading SDC file my_block_sdc.
// read_sdc summary: 4 false paths, 0 multicycle paths, 0 erroneous paths
//    0 disable timings, 0 case analyses, 0 clock groups
```

Example 8

The following example reads the SDC file *my_block_sdc* and applies the information to every instance of the design module my_block:

delete_false_paths -all
read_sdc my_block_sdc -module my_block

```
// Reading SDC file my_block_level_sdc ...
// Finished reading SDC file my_block_level_sdc.
// read_sdc summary: 6 false paths, 0 multicycle paths, 0 erroneous paths
//    0 disable timings, 0 case analyses, 0 clock groups
```

Example 9

This example shows that we can read an SDC file in setup mode, transition to analysis mode, and perform X-bounding based on the paths in the SDC file:

read_sdc my_design.sdc

set_system_mode analysis

analyze_xbounding

Related Topics

add_false_paths

report_clocks

report_false_paths

report_multicycle_paths

set_timing_exceptions_handling

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_sdf

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Loads timing data from a Standard Delay Format (IEEE SDF 1497) file.

Usage

read_sdf ***sdf_filename***...
[-MInimum_delay | -TYpical_delay | -MAximum_delay]
[-Replace | -Append | -INCremental]
[-INstance *hierarchical_instance_name*]

Description

Loads timing data from a Standard Delay Format (IEEE SDF 1497) file.

The SDF file is generated by your static timing analysis tool. For complete information, refer to "Timing-Aware ATPG" in the *Tessent Scan and ATPG User's Manual*.

For descriptions of any errors and warnings reported by read_sdf, see "Errors and Warnings While Reading SDF Files" in the *Tessent Scan and ATPG User's Manual.*

The following limitations apply to the SDF file:

- For SDF conditional delays, the tool loads them into an internal database. However, only the maximum delay is used for calculating actual delay. For example:
- (COND b (IOPATH a y (0.21) (0.54)))

  (COND ~b (IOPATH a y (0.27) (0.34)))
- Timing-aware ATPG uses (0.27) (0.54) regardless of the value of b.
- Only the first two delay value tokens are used.
- PATHPULSE, PATHPULSEPERCEN, DEVICE, and RETAIN statements are ignored.
- Delays on Memory and Latch are ignored.
- Hold time information is ignored.
- Delays in clock lines are ignored.
- Latch is assumed to be either a launch or capture point, not as a combinational logic in-between.

Arguments

- ***sdf_filename***

  A required string that specifies the name of the SDF file to read.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -MInimum_delay

  An optional switch that loads only the SDF minimum delay data into the tool.

- -Typical_delay

  An optional switch that loads only the SDF typical delay data into the tool. This is the default.

- -MAximum_delay

  An optional switch that loads only the SDF maximum delay data into the tool.

- -REplace

  An optional switch that removes any previously loaded SDF file information before loading the new SDF file information.

- -APpend

  An optional switch that loads the new SDF file information in addition to the existing information. This is the default. This switch enables you to load multiple SDF files for different modules.

  The tool issues an error message if the new SDF file contains delay data for a flattened pin that was already defined by a previously loaded SDF file. Use the -INCremental switch if you want to merge all delay data for a flattened pin.

  Note

  When using this switch, if the selected delay type (minimum, typical or maximum delay) is different from the previous read_sdf command, all previously loaded SDF timing information is discarded.

- -INCremental

  An optional switch that loads the new SDF file information in addition to the existing information. If the new SDF file contains delay data for a flattened pin that was already defined by a previously loaded SDF file, the new delay data is cumulated with the previous delay data for that pin. Use the -APpend switch if you do not want to merge any delay data for a flattened pin.

- -INstance *hierarchical_instance_name*

  An optional switch and string pair that specifies the hierarchical instance name to which the SDF file is associated. This option is useful when the SDF file is at the module level.

Examples

The following example reads the typical delay data from the *top_worst.sdf* file:

```
read_sdf top_worst.sdf
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following example reads the delay data for multiple modules:

```
read_sdf file1.sdf -maximum_delay -instance /top/module1
read_sdf file2.sdf -maximum_delay -append -instance /top/module2
read_sdf file3.sdf -maximum_delay -append -instance /top/module3
```

Related Topics

report_statistics

set_atpg_timing

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_upf

Context: all contexts

Mode: setup

Prerequisite: You can only run the read_upf command after the design is elaborated.

Loads a UPF (Unified Power Format) file and removes any previously loaded power data.

Usage

read_upf ***filename***

Description

The read_upf command loads power data into the tool, and the report_gates command enables power data reporting and shows the power on (PON) or power off (POFF) as well as the power domain of the gate.

Loading multiple UPF files requires specifying them in the UPF with the load_upf command.

The tools support IEEE 1801-2018 (UPF version 3.1).

For information about what happens when you load power data, refer to "Power-Aware DRC and ATPG" in the *Tessent Scan and ATPG User's Manual*.

Note

As described in the create_dft_specification command description, the power domain information affects the DftSpecification to ensure that a SIB isolates groups of DesignInstances associated to different power domains. This is done to avoid daisy-chaining them with elements from a different power domain and running into issues when accessing the elements in a particular power domain while the other power domains are off.

Arguments

- ***filename***

  A required string that specifies the name of the UPF file.

Examples

Example 1

The following example loads a UPF file:

```
read_upf upf_file
```

Example 2

In this example, read_upf loads power data into the tool and the report_gates command reports the power on (PON) status as well as the power domain of the gate.

```
read_upf upf_file
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_gates /lp_case_si_rst_sms/ati_rst_sync/sync_r/U3/Udff

```
//  /lp_case_si_rst_sms/ati_rst_sync/sync_r  sync3msfqxss1ul
//      SDI    I(PON)  /vl_sms_lp_case_si_sms_proc_sms_1_stp/U_lp_case_si_sms_proc_bist/uu5/Z
//      D      I(PON)  /lp_case_si_rst_sms/ati_rst_sync/sync_buf/Z
//      SEN    I(PON)  /se
//      CLK    I(PON)  /lp_case_si_rst_sms/ati_rst_sync/uu1/Z
//      Q      O(PON)  /lp_case_si_rst_sms/ati_rst_sync/uu2/A  /lp_case_si_rst_sms/and_r/B
//      in power domain PD_P2
```

Related Topics

add_faults

delete_faults

report_faults

read_cpf

report_power_data

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_verilog

Context: dft, patterns (except patterns -scan_diagnosis)

Mode: setup

Reads one or more Verilog files into the specified or default logical library.

Usage

dft -no_rtl and patterns -no_rtl Contexts

read_verilog [*design_path* ...] [-verbose] [-force | -no_overwrite_modules]
[-interface_only] [-exclude_from_file_dictionary]
[-no_lib_rescan] [-format {1995 | 2001 | sv31a | sv2005 | sv2009}]
[-pragma_list *pragma_list*] [-ignore_synthesis_off_sections {On | Off}]
[-f *files*] [-no_duplicate_modules_warnings] [-no_limited_rtl [-report_all_errors]]

dft -rtl and patterns -rtl Contexts

read_verilog [*design_path* ...] [-verbose] [-force | -no_overwrite_modules]
[-interface_only] [-exclude_from_file_dictionary]
[-in_library *logical_lib_name*] [-no_lib_rescan] [-no_mfcu]
[-format {1995 | 2001 | sv31a | sv2005 | sv2009}]
[-pragma_list *pragma_list*] [-ignore_synthesis_off_sections {On | Off}]
[-vcs_compatibility] [-f *files*] [-no_duplicate_modules_warnings]

Description

Define the list of available libraries with the set_logical_design_libraries command in the rtl context. Only one library, named “work”, exists in the no_rtl context.

The TESSENT, MGC_TESSENT, and SYNTHESIS design macros are defined by default. For more information, refer to the set_design_macros command description.

When in -no_rtl context (set_context *context* -no_rtl), the tool reports a warning when parsing RTL constructs. See the -no_limited_rtl switch for additional guidance.

Note

If you define the ports of a child block using parameters from a separate package file, to generate a clean interface model of the child block, you must load the package file separately using the “read_verilog <*pkg file*> -format sv2009” command. This package file is then explicitly marked as “needed_before_reading_interface_view on” in the <*child block + design_name*>*.design_source_dictionary* file.

In higher-level design insertion or integration, the child block’s read interface view only reads the interface model plus the package file, as needed. Failing to satisfy this requirement results in all design files being loaded when you only want to read the child block interface model. The higher level *design_name.design_source_dictionary* file now also includes the lower-level child block’s design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- *design_path*

  One or more optional files containing Verilog elements to load. A well formatted Tcl list of filenames can also be supplied, such as the list returned by the Tcl glob command. The names of the files can contain wildcards. The *design_path* is optional if you use the -f switch to specify a file that contains at least one design_path.

  If you have used the set_design_sources command, child modules found instantiated into the currently read modules are searched for in the specified -y directories and the -v files.

- -verbose

  An optional switch that reports each file read. For every file read, including the ‘include files, the following messages are returned:

  ```
  // Reading Verilog file filename
  // Finished reading file filename
  ```

- -force

  An optional switch that suppresses the error message that normally prevents the accidental deletion of previously defined setup information. After you read in a design using this command, you can incrementally issue the command again to read more files. However, once the tool builds the design using set_current_design and you add any configuration information to it (for example, add_clocks or add_scan_chains), running the read_verilog command causes the setup information to be discarded. In this case, to prevent configuration information from being discarded unintentionally, the tool requires that you run the command with the -force switch to indicate that it is acceptable to delete the setup data.

- -no_overwrite_modules

  An optional switch that instructs the tool not to replace existing modules. If a module is read that has been read before, the tool reports an error. You can use this switch to avoid overwriting modules or accidentally deleting the design.

- -interface_only

  An optional switch that instructs the tool to ignore the internals of all of the modules specified in the filename argument and extract only the module port definitions and parameters.

- -exclude_from_file_dictionary

  An optional switch used when you input a Verilog simulation model into Tessent Shell, and the model does not have a `celldefine; issuing this switch ensures that the read is excluded in the file dictionary and, consequently, excluded in the DC shell load script.

- -in_library *logical_lib_name* (“set_context *context* -rtl” only)

  An optional switch that specifies which logical library the modules are to be read into. The list of possible logical libraries, including the default library, is defined by the set_logical_design_libraries command. If this option is not specified, the modules are read into the default library.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -no_lib_rescan

  An optional switch that specifies to not rescan the -y directories and the -v files from the beginning each time, but to search for the module beginning with the directory or file in which the previous module was found.

- -no_mfcu (“set_context *context* -rtl” only)

  An optional switch that applies to SystemVerilog formats, which specifies the compilation of each file into a separate compile unit. By default, the tool compiles into a multiple-file unit, which makes all `defines, `default_nettype, global typedefs, and any object defined as global visible across the files included with the read_verilog command, including `include files referenced.

  Use this switch to limit the visibility of such constructs/definitions to local for each regular file given to the read_verilog command. In this case, `include files (including recursive `include files) remain part of the regular file that references them.

- -format {1995 | 2001 | sv31a | sv2005 | sv2009}

  An optional switch that specifies the format of the input Verilog files. The default is 2001. In a mixed-format situation, you should be sure to re-define your -y and -v library search path when loading modules in a new format because you cannot mix formats within a single read_verilog command invocation.

  In the -no_rtl context, you must use this switch together with the -interface_only switch to read modules that have interfaces without special System Verilog ports.

  In the -rtl context, you can the -format switch with or without the -interface_only switch.

  The sv31a is not a standard format but it is the default inside Design Compiler when hdlin_sv_packages is not defined.

- -pragma_list *pragma_list*

  An optional switch and string pair that defines the pragmas for the input files. The pragma_list is a Tcl list containing one or more objects. When this option is not specified, the tool uses the following list of strings as default pragmas: {pragma synthesis tessent mgc_tessent synopsys LV_pragma mentor}. You can override the default pragmas by specifying a new pragma list as shown here:

  ```
  read_verilog 1.v -pragma_list {my_pragma1 tessent}
  ```

  You can create an empty list of pragmas by specifying an empty Tcl list as shown here:

  ```
  read_verilog 1.v -pragma_list {}
  ```

  Pragmas are special comments in the source. They control how the tool interprets the source. This example of source code uses a pragma pair to turn translation on or off:

  ```
  module ram1 ( …
    ...
    // synopsys translate_off
    myram u_atpg (  .... );
    // synopsys translate_on
  ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

By default, synopsys is a pragma. The tool ignores all input from when it reads the pragma directive "// synopsys translate_off" until the "// synopsys translate_on" pragma directive. The tool does not include the myram instance.

If you use the command, "read_verilog with -pragma_list {}", you unset the default pragmas and the tool includes the myram instance.

- -ignore_synthesis_off_sections {On| Off}

  An optional switch and literal pair that specifies whether to ignore the synthesis_off sections in the input files. The default is On.

  On — A literal that specifies to ignore synthesis_off sections and to NOT compile the code found within the synthesis_off/on block.

  Off — A literal that specifies to not ignore the synthesis_off sections and to compile the code found within the synthesis_off/on block.

- -vcs_compatibility ("set_context *context* -rtl" only)

  An optional switch that specifies to ignore a series of syntax errors permitted by the Synopsys compilers. By default, the tool reports these non-compliant usages as errors.

  When specified, this setting is in effect until you run the delete_design command. You only specify this switch in the first execution of the read_verilog command after tool invocation, or after the delete_design command is run.

  See "HDL Limitations in the Tessent Shell Flow" on page 5081 for information about how hardware description languages are used in Tessent Shell.

- -f *files*

  An optional switch and Tcl list that specifies the name(s) of one or more ASCII files that contains pointers to design files. The referenced file is usually generated by another tool, so the -f option enables you to use the file rather than to reformat the contents of the file into a read_verilog command line. The file can contain only the following elements, one per line:

  - *design_path* — A string that specifies the name of a Verilog file to load.
  - -v *design_path* — A string that specifies the name of a Verilog file to load.
  - -y *dir_path* — A string that specifies a path to a directory that contains Verilog files to load.
  - +libext+*file_extension* — A string that specifies which files in the "-y dir_path" to load. For example: "+libext+.v" loads all .v files.
  - +define+*XXX[=YYY]* — A string that defines a macro called "XXX" with an optional value of "YYY". The defined macro is used when compiling any files listed in the -f file.
  - +incdir+*dir_path1*+*dir_path2*+... — A string that defines one or more directory paths to directories that contain files that use the 'include directive.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -no_duplicate_modules_warnings

  An optional switch that suppresses overwrite warnings when loading a netlist.

- -no_limited_rtl [-report_all_errors] ("set_context *context* -no_rtl" only)

  An optional switch to flag an error when parsing RTL when the tool is in -no_rtl context. When not specified, RTL Verilog constructs are flagged with a warning.

  The environment variable TESSENT_NO_LIMITED_RTL can be used instead of the -no_limited_rtl switch. Verilog attributes are ignored with a warning by default and with the switch. The optional -report_all_errors switch reports all of the RTL construction errors found by the tool in all modules. Usage is as follows:

  - -no_limited_rtl — When the tool encounters the RTL construct, the tool reports one error per module for all modules in all files.
  - -no_limited_rtl -report_all_error — The tool reports all the RTL construct errors that are encountered in all the modules in all files.
  - TESSENT_NO_LIMITED_RTL = 1 (it can be any value except "report_all_errors") — When the tool encounters the RTL construct, the tool reports one error per module for all modules in all files.
  - TESSENT_NO_LIMITED_RTL = report_all_errors (it can be upper case or lower case) — The tool reports all the RTL construct errors that are encountered in all the modules in all files.

Examples

Example 1

The following example loads all Verilog files matching *.v inside the directory src:

read_verilog src/*.v

Example 2

The following example loads a mixed-language design in the dft context in RTL mode. The results files are written into the *edited_design* directory.

SETUP> set_context dft -rtl

```
# Specify LIBRARY1 to be mapped into <workdir>/lib1
# work is mapped to a scratch dir
```

SETUP> set_logical_design_libraries -logical_library_list { library1 work}

```
# Define macro1 for Verilog compilation
```

SETUP> set_design_macros macro1=5

```
# Search for Verilog modules in /design/abc/*.v
```

SETUP> set_design_sources -y /design/abc -extension v

```
# Read Verilog files
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SETUP> read_verilog top.v core.v -in_library work -format 2001

```
# Read VHDL files
```

SETUP> read_vhdl core1.vhdl -in_library work -format 1993
SETUP> read_vhdl {core2.vhdl core3.vhdl} -in_library library1 -format 1993

```
# Set the top
```

SETUP> set_current_design top

```
# Enter insertion mode
```

SETUP> set_system_mode insertion

```
# Perform design query and manipulation commands
```

INSERTION> …

```
# Write the design
```

INSERTION> write_design -output_directory edited_design -modified

INSERTION> exit

Example 3

The following example uses the -f *files* switch to load in the design file(s) specified within an ASCII file that contains spaces in the filename:

SETUP> read_verilog -f [list "updated cores.f"]

Example 4

The following example uses the -f *files* switch to load in the design files specified within two different ASCII files:

SETUP> read_verilog -f {block_a_cores.f block_b_cores.f}

Example 5

The following example loads the design ports using a System Verilog package and then uses the -f *files* switch to load in the design files:

SETUP> read_verilog *design_pkg.sv* -format sv2009
SETUP> read_verilog -f {block_a_cores.f block_b_cores.f}

Related Topics

delete_design

read_cell_library

read_flat_model

read_vhdl

set_current_design

set_design_include_directories

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_design_macros

set_design_sources

set_logical_design_libraries

HDL Limitations in the Tessent Shell Flow for Verilog and SystemVerilog

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_vhdl

Context: all contexts

Mode: setup

Reads one or more VHDL files into the specified or default logical library.

Usage

dft -no_rtl and patterns -no_rtl Contexts

read_vhdl [*design_path*] [-interface_only] [-exclude_from_file_dictionary]
[-force | -no_overwrite_modules] [-format {1987 | 1993 | 2002 | 2008}]
[-pragma_list *pragma_list*] [-ignore_synthesis_off_sections {on | off}]
[-f *files*] [-no_duplicate_modules_warnings]

dft -rtl and patterns -rtl Contexts

read_vhdl [*design_path*] [-interface_only] [-exclude_from_file_dictionary]
[-force | -no_overwrite_modules] [-in_library *logical_lib_name*]
[-format {1987 | 1993 | 2002 | 2008}] [-pragma_list pragma_list]
[-ignore_synthesis_off_sections {on | off}] [-f *files*]
[-no_duplicate_modules_warnings]

Description

The list of possible logical libraries, including the default library, is defined by the set_logical_design_libraries command.

You can run this command with the -force switch but without a *design_path* argument to clear design setup data only and not read any new files.

Arguments

- *design_path*

  An optional Tcl list that specifies the name(s) of one or more VHDL files to load. The names of the files can contain wildcards. The *design_path* option is optional if you use the -f switch to specify a file that contains at least one *design_path*.

  This command includes an example that shows a suggested way of loading different VHDL files into different libraries.

- -interface_only

  An optional switch that instructs the tool to ignore the internals of all of the modules specified in the filename argument and extract only the module port definitions and parameters.

- -exclude_from_file_dictionary

  An optional switch that is used when you input a VHDL simulation model into Tessent Shell, and the model does not have a `celldefine; issuing this switch ensures that the read is excluded from the file dictionary and, consequently, excluded in the DC shell load script.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -force

  An optional switch that suppresses the error message that normally prevents the accidental deletion of previously defined setup information. After you read in a design using this command, you can incrementally issue the command again to read more files. However, once the tool builds the design using set_current_design and you add any configuration information to it (for example, add_clocks or add_scan_chains), running the read_vhdl command causes the setup information to be discarded. In this case, to prevent configuration information from being discarded unintentionally, the tool requires that you run the command with the -force switch to indicate that it is acceptable to delete the setup data.

- -no_overwrite_modules

  An optional switch that instructs the tool not to replace existing modules. If a module is read that has been read before, the tool reports an error. You can use this switch to avoid overwriting modules or accidentally deleting the design.

- -in_library *logical_lib_name* ("set_context *context* -rtl" only)

  An optional switch that specifies which logical library the modules are to be read into. The list of possible logical libraries, including the default library, is defined by the set_logical_design_libraries command. If this option is not specified, the modules are read into the default library. This switch is only available in -rtl context.

- -format {1987 | 1993 | 2002 | 2008}

  An optional switch that specifies the expected format of the input files and which VHDL language constructs can be understood during parsing. The default is 1993.

- -pragma_list *pragma_list*

  An optional switch and string pair that defines the pragmas for the input files. The pragma_list is a Tcl list containing one or more objects. When this option is not specified, the tool uses the following list of strings as default pragmas: {pragma synthesis tessent mgc_tessent synopsys LV_pragma mentor}. You can overwrite the default pragmas by specifying a new pragma list as shown here:

  ```
  read_vhdl 1.v -pragma_list {my_pragma1 tessent}
  ```

  You can create an empty list of pragmas by specifying an empty Tcl list as shown here:

  ```
  read_vhdl 1.v -pragma_list {}
  ```

  Pragmas are special comments in the source. They are used to control how the tool interprets the source. This example of source code uses a pragma pair to turn translation on and off:

  ```
  begin
  -- synthesis translate_off
  myram: work.comp_decls.topmem port map(...);
  -- synthesis translate_on
  ```

  If synthesis is specified as a pragma, either by using the -pragma_list {synthesis} switch pair for read_vhdl or because it is a default pragma, the tool recognizes the "-- synthesis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

translate_off" directive and ignores all input from the source until it encounters the "-- pragma translate_on" directive. The result, in this case, is that myram is not included.

If you issue a read_vhdl with -pragma_list {}, the default pragmas are unset and the instance myram is included by the tool.

- -ignore_synthesis_off_sections {On | Off}

  An optional switch and literal pair that specifies whether to ignore the synthesis_off sections in the input files. The default is On.

  On — A literal that specifies to ignore synthesis_off sections and to NOT compile the code found within the synthesis_off/on block.

  Off — A literal that specifies to not ignore the synthesis_off sections and to compile the code found within the synthesis_off/on block.

- -f *files*

  An optional switch and Tcl list that specifies the name(s) of one or more ASCII files that contains pointers to design files to load. The names of the files can contain wildcards. The referenced file is usually generated by another tool, so the -f option enables you to use the file rather than to reformat the contents of the file into a read_vhdl command line. The file can contain only the following elements, one per line:

  - *design_path* — A string that specifies the name of a VHDL file to load.

- -no_duplicate_modules_warnings

  An optional switch that suppresses overwrite warnings when loading a netlist.

Examples

Example 1

The following example instructs the tool to read only the interface of the modules/entities and ignore the contents:

```
set_context dft -no_rtl
read_verilog data/pll.v -interface_only
read_vhdl  {data/ram_bist.vhd  data/edt_ip*.vhd} -interface_only
```

Example 2

The following example loads a mixed-language design in the dft context in RTL mode. The results files are written into the *edited_design* directory.

```
SETUP> set_context dft -rtl

# Specify LIBRARY1 to be mapped into <workdir>/lib1
# work is mapped to a scratch dir

SETUP> set_logical_design_libraries -logical_library_list { library1 work}

# Define macro1 for Verilog compilation

SETUP> set_design_macros macro1=5
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# Search for Verilog modules in /design/abc/*.v
```

SETUP> set_design_sources -y /design/abc -extension v

```
# Read Verilog files
```

SETUP> read_verilog top.v core.v -in_library work -format 2001

```
# Read VHDL files
```

SETUP> read_vhdl core1.vhdl -in_library work -format 1993
SETUP> read_vhdl {core2.vhdl core3.vhdl} -in_library library1 -format 93

```
# Set the top
```

SETUP> set_current_design top

```
# Enter insertion mode
```

SETUP> set_system_mode insertion

```
# Perform design query and manipulation commands
```

INSERTION> …

```
# Write the design
```

INSERTION> write_design -output_directory edited_design -modified

INSERTION> exit

Example 3

The following example uses the -f *files* switch to load in the design file(s) specified within an ASCII file that contains spaces in the filename:

SETUP> read_vhdl -f {updated cores.f}

Example 4

The following example uses the -f *files* switch to load in the design files specified within two different ASCII files:

SETUP> read_vhdl -f {block_a_cores.f block_b_cores.f}

Related Topics

delete_design

read_icl

read_verilog

set_current_design

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

register_attribute

Context: unspecified, all contexts

Mode: all modes

Registers a new user-defined attribute.

Usage

register_attribute **-name** ***attribute_name*** **-value_type** ***data_type*** [-default *default_value*] [-min *min_value*] [-max *max_value*] [-enum *enum_values*] **-object_types** ***type_list*** [-dense] [-description *description_text*] [-show_default] [-hidden]

Description

Registers a new user-defined attribute.

This command enables you to define new attributes for object types available in Tessent Shell. After registration, the value of the new attribute can be set and retrieved on objects such as a module or an instance.

If an attribute with the same name already exists for the specified object type, or if an object type in *type_list* does not exist, or if a data type is invalid for one of the options, an error message displays.

If you want an attribute set on a module to be visible (inherited) on the instances of the module, register the attribute only on the *module* object type and not on the instance object type. The same is true for inheritance between ports and pins and between nets and pins. The following exceptions to this inheritance rule exist for Boolean attributes:

- For Boolean attributes registered on module and instance object types — When the value is queried on an instance and it is the default value (usually false), the value from the corresponding module is returned.
- For Boolean attributes registered on port and pin object types — When the value is queried on a pin and it is the default value (usually false), the value from the corresponding port is returned.

Arguments

- **-name** ***attribute_name***

  A required switch and value pair that specifies the name of the attribute to register. An attribute name must begin with an alpha character and must only contains alpha-numeric characters and the underscore (_).

- **-value_type** ***data_type***

  A required switch and value pair that specifies the attribute's data type. The -value_type argument is an enumerated list with the following possible values: Boolean, integer, double, and string.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -default *default_value*

  An optional switch and value pair that specifies the default value for the attribute when the attribute is not explicitly set. However, this pair is required for attributes with non-Boolean data types only; it cannot be set for attributes whose data type is Boolean. Attributes with a data type of Boolean automatically default to false.

- -min *min_value*

  An optional switch and value pair that specifies the minimum value for the attribute. When this argument is specified, the value of the attribute is only accepted if it is greater than or equal to *min_value*. This argument is only valid for attributes whose data type is integer or double. For more information on these data types, see "Attribute Value Types."

- -max *max_value*

  An optional switch and value pair that specifies the maximum value for the attribute. When this argument is specified, the value of the attribute is only accepted if it is less than or equal to *max_value*. This argument is only valid for attributes whose data type is integer or double. For more information on these data types, see "Attribute Value Types."

- -enum *enum_values*

  An optional switch and value pair that specifies a Tcl list of possible values for the attribute. The *enum_values* value can only be set to one value in the list. This argument is only valid for attributes whose data type is string.

- **-object_types *type_list***

  A required switch and value pair that specifies the object type to which the new attribute can be assigned. The *type_list* string is a Tcl list of one or more object types (for example, module, instance, port, pin, net, and so on). If an object type is not included in the *type_list*, the new attribute cannot be assigned to that *object_type*.

- -dense

  An optional Boolean switch that specifies the storage type for the attribute.

  When not specified — Specifies sparse storage, which only stores values for objects that have an attribute value set to something different than the default value. This option is a good choice when many of the attributes on an object are set to their default value because it saves memory. However, indexing and searching can be slower and end up consuming more memory if the number of objects having the attribute specified is more than 10 percent of the object count.

  When specified — Specifies dense storage, which creates an entry for each object and its attribute value. This option is a good choice when many objects with the same attribute have a distributed set of attribute values because it enables faster indexing and searching. However, it has a higher memory usage when less than 10 percent of the objects have the attribute specified.

- -description *description_text*

  An optional switch and value pair that specifies to store a string describing the attribute. The description is displayed when the report_attributes command is run for the attribute. The

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

description_text argument is a quoted string that can be formatted as a Tcl string, which includes support for escaped characters such as "\n" and "\t".

- -show_default

  An optional Boolean switch that specifies how the registered attribute is handled by the get_attribute_list and report_attributes commands.

  When not specified — The "get_attribute_list ***obj_spec***" and "report_attribute ***obj_spec***" command executions list or report attributes found on any object within ***obj_spec*** only if the attribute is set to a non-default value.

  When specified — The attribute is always listed or reported if ***obj_spec*** contains at least one object of the type for which the attribute is registered.

- -hidden

  An optional switch that specifies the attribute should not be listed or reported by the get_attribute_list and report_attributes commands. The attribute can be set and queried using the set_attribute_value and get_attribute_value_list commands. You use this switch when you want to keep an attribute hidden. For example, while you develop a new feature and want to only make the attribute visible when you are ready to release the new feature.

Examples

Example 1

The following example registers the attribute named speed with a range of possible values equal to or greater than 100 and less than or equal to 500; assigns a default value of 200 to it; sets it as an attribute for the pin design object type only. The Tcl append command enables typing the string on multiple lines without inserting extra white spaces into the string.

```
set      desc "This attribute is used to hold the target frequency the pin is \n"
append  desc "permitted to have.\n"
append  desc "The permitted frequency range of 100 to 500 was chosen to match\n"
append  desc "the technology spec."

register_attribute -name speed \
        -value_type integer \
        -default 200 \
        -min 100 \
        -max 500 \
        -object_types pin \
        -description$desc
```

Example 2

The following example defines the attribute named added_by as a string with a value of either "John", "Patrick", or "Susan" with a default of John, and a design object type of module:

```
register_attribute -name added_by \
        -value_type string \
        -default John \
        -enum {John Patrick Susan} \
        -object_types net
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

get_attribute_list

get_attribute_option

get_attribute_value_list

reset_attribute_value

report_attributes

set_attribute_options

set_attribute_value

unregister_attribute

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

register_callback

Context: unspecified, all contexts

Mode: all modes

Register a Tcl proc that is to automatically run before or after a built-in feature of Tessent Shell.

Usage

register_callback ***feature_identifier proc_name***
[-order *integer*] [-static_arguments *list_of_argument_values*]

Description

The register_callback command registers a Tcl proc to automatically run before or after a Tessent Shell builtin feature runs. You use this command in a Tcl file located at *tcl_modules/your_company_name/any_name.tcl* file in one of your plugin directory.

Once you have created and registered a callback procedure to run before or after a specific feature, the proc behaves as if it is part of the builtin behavior of the tool.

You can display the run time for the callbacks you have registered by setting the following environment variable prior to invoking the tool:

```
TESSENT_DISPLAY_CALLBACKS_RUN_TIME 1
```

When this variable has a value of 1, the tool displays the execution time of those callbacks you have registered.

Arguments

- ***feature_identifier***

  A special string that identifies the Tessent Shell builtin feature before or after which you want your proc to run. The following lists all supported feature names and the arguments your proc must have:

  - create_dft_specification.post

    Registers a Tcl proc that automatically runs after the DftSpecification wrapper has been created within the create_dft_specification command.

    If a proc called “create_dft_specification.post” is declared in the global namespace, the proc is automatically run after all other procs registered for that feature. If you explicitly register the proc called “create_dft_specification.post” in the global namespace, the proc runs at the specified order value instead of at the end.

    The proc must be declared with these arguments: {*root_wrapper args*}.

    The *root_wrapper* argument holds the value of the DftSpecification wrapper that was created. The wrapper can then be modified using the configuration editing commands listed in Table 13-1 on page 3843.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This callback is useful to automatically apply custom edits to the created DftSpecification wrapper.

- create_flat_model.post

  Registers a Tcl proc that automatically runs after the flat model is created. The proc must be declared with this argument: {args}.

  This callback is useful to attach attributes on the newly generated gate_pin objects.

  The "get_system_mode -is_in_transition" command returns a "1" when used in the registered callback associated with this feature and the feature is used in a system mode transition. For more information on -is_in_transition, see the get_system_mode command description.

- create_flat_model.pre

  Registers a Tcl proc that automatically runs before the flat model is created. The proc must be declared with this argument: {args}.

  This callback is useful to insert checks that you want to enforce prior to generating the flat model. You can also use it to set the preserve_boundary attribute (see "Built-In Attributes" on page 3655) on hierarchical pins you want preserved in the flat_model. To block the creation of the flat model, issue the error condition with the display_message command and terminate the proc with the "return -code error" command.

  The "get_system_mode -is_in_transition" command returns a "1" when used in the registered callback associated with this feature and the feature is used in a system mode transition. For more information on -is_in_transition, see the get_system_mode command description.

- create_patterns_specification.post

  Registers a Tcl proc that automatically runs after the PatternsSpecification wrapper has been created within the create_patterns_specification command.

  If a proc called "create_patterns_dft_specification.post" is declared in the global namespace, it is automatically run after all other procs registered for that feature. If you explicitly register the proc called "create_patterns_specification.post" in the global namespace, it runs at the specified order value instead of at the end.

  The proc must be declared with these arguments: {*root_wrapper args*}.

  The *root_wrapper* argument holds the value of the PatternsSpecification wrapper that was created. The wrapper can then be modified using the configuration editing commands listed in Table 13-1 on page 3843.

  This callback is useful to automatically apply custom edits to the created PatternsSpecification wrapper.

- delete_design.pre

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Registers a Tcl proc that automatically runs before the design is deleted. The proc must be declared with these arguments: {args}.

This callback is useful to insert checks that you want to enforce prior to deleting the design data. To block the deletion of the design, issue the error condition with the display_message command and terminate the proc with the "return -code error" command.

- elaboration_creation.post

  Registers a Tcl proc that automatically runs after the design is elaborated. The proc must be declared with this argument: {args}.

  This callback is useful to automatically perform actions on the newly-elaborated design such as setting attributes on the design objects. The get_current_design command can be used within this proc to get the module that was elaborated.

  The "get_system_mode -is_in_transition" command returns a "1" when used in the registered callback associated with this feature and the feature is used in a system mode transition. For more information on -is_in_transition, see the get_system_mode command description.

- elaboration_creation.pre

  Registers a Tcl proc that automatically runs before the design is elaborated. The proc must be declared with this argument: {args}.

  The args consist of a dictionary containing two keys: design_name and library_name. The design_name key holds the name of the design that is going to be elaborated. The library_name contains the name of the logical library in which the design being elaborated exists.

  This callback is useful to automatically perform actions before the design is elaborated such as reading some extra design files.

  The "get_system_mode -is_in_transition" command returns a "1" when used in the registered callback associated with this feature and the feature is used in a system mode transition. For more information on -is_in_transition, see the get_system_mode command description.

- exit.pre

  Registers a Tcl proc that automatically runs before the tool exits.

  The proc must be declared with this argument: {args}.

  - Use "array set ARGS $args" to set the arguments into an array.
  - Use "$ARGS(force)" to find out if the -force Boolean option was specified or not. The $ARGS(force) value is 1 if the -force Boolean switch of the exit command was specified and 0 if it was not.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Use "$ARGS(exit_val)" to find the exit code at the time of the exit if you want to perform a specific action after an error caused the tool to exit.

The tool automatically exits if it was started without a STDIN setting.

- It exits with an exit code of 0 when the tool runs out of dofile actions.
- It exits with an exit code of 1 or greater (typically 1) if the tool terminates because of an error.

If you call the exit command in your dofile, you can control the exit code with the exit code argument.

This callback is useful to automatically perform checks that you want to enforce prior to exiting the tool. To block the exit of the tool, issue the error condition with the display_message command and terminate the proc with the "return -code error" command.

- extract_icl.post

  Registers a Tcl proc that automatically runs after ICL extraction has completed. This includes the writing of the ICL to file.

  The proc does not need any argument but it is recommended to use the {args} argument to ensure forward compatibility.

  This callback enables you to post process any files written out during ICL extraction. If you need to add attributes to ICL objects before the ICL is written out to file, use the extract_icl.pre_write callback.

- extract_icl.pre

  Registers a Tcl proc that automatically runs before any check and setup of the design to ensure proper ICL extraction.

  The proc does not need any argument but it is recommended to use the {args} argument to ensure forward compatibility.

  This callback enables you to perform your own check and setup of the design to ensure successful ICL extraction and assuming that you have added some DFT circuitry yourself.

- extract_icl.pre_write

  Registers a Tcl proc that automatically runs after ICL extraction has completed but before the writing of the ICL to a file. The callback is called even if "extract_icl -write_in_tsdb" is set to off.

  The proc does not need any argument but it is recommended to use the {args} argument to ensure forward compatibility.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This callback is useful to add attributes to ICL objects before the ICL is written out to file. To perform actions on the written out file, use the extract_icl.post callback that is called at the very end of the ICL extraction feature.

- insert_test_logic.post

  Registers a Tcl proc that automatically runs after the insert_test_logic has done all its actions including saving the design.

  The proc must be declared with this argument: {args}.

  This callback is useful to do design introspection and generate custom reports.

- insert_test_logic.post_insertion

  Registers a Tcl proc that automatically runs after the insert_test_logic has done all its insertion actions but before it saves the design.

  The proc must be declared with this argument: {args}.

  This callback is useful to do custom editing to the netlist that you want to be part of the saved netlist.

- insert_test_logic.pre

  Registers a Tcl proc that automatically runs after the insert_test_logic has done all its insertion actions but before it saves the design.

  The proc must be declared with this argument: {args}.

  This callback is useful to do custom checks prior to letting the insert_test_logic do its job. If your check fails, then you issue error messages with the "display_message -error" command and return with -code error to cause the insert_test_logic operations to be interrupted. This leaves the tool in analysis mode, so someone can fix the issues reported by your custom checks.

- module_matching.post

  Registers a Tcl proc that automatically runs after the module matching feature was performed. See the set_module_matching_options and the set_design_sources command description for more information about this feature.

  The proc must be declared with this argument: {args}.

  This callback is useful to set attribute on the design objects from the matched ICL or TCD views—see "Tessent Core Description" on page 4793. For example, you can use the get_ijtag_instances or "get_modules -filter instrument_name" commands to get the list of design instances or modules for which there is a matched ICL module then use the ICL introspection commands to transfer information from ICL to the associated design objects.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The “get_system_mode -is_in_transition” command returns a “1” when used in the registered callback associated with this feature and the feature is used in a system mode transition. For more information on -is_in_transition, see the get_system_mode command description.

- module_matching.pre

  Registers a Tcl proc that automatically runs before the module matching feature is performed. See the set_module_matching_options and the set_design_sources command description for more information about this feature.

  The proc must be declared with this argument: {args}.

  This callback is useful to adjust the appropriate set_design_sources and set_module_matching_option settings prior to doing the module matching.

  The “get_system_mode -is_in_transition” command returns a “1” when used in the registered callback associated with this feature and the feature is used in a system mode transition. For more information on -is_in_transition, see the get_system_mode command description.

- process_dft_specification.post_insertion

  Registers a Tcl proc that automatically runs after the insertion of the instruments was performed within the process_dft_specification command.

  If a proc called “process_dft_specification.post_insertion” is declared in the global namespace, it is automatically run after all other procs registered for that feature. If you explicitly register the proc called “process_dft_specification.post_insertion” in the global namespace, it runs at the specified order value instead of at the end.

  The proc must be declared with those arguments: {root_wrapper args}.

  The root_wrapper argument holds the value of the DftSpecification wrapper that was processed.

  This callback is useful to automatically apply custom edits to the design after all instruments were inserted but prior to when the modified design is written out into the dft_inserted_design directory.

- process_patterns_specification.post_open

  Registers a Tcl proc that automatically runs immediately after the open_pattern_set command associated to every TestStep or ProcedureStep wrapper found in the processed PatternsSpecification.

  The proc must be declared with those arguments: {config_path args}. The config_path argument holds the path to the TestStep or ProcedureStep wrapper being processed.

  This callback is useful to add debug code that introspects the initial state of the network prior to running the iCalls associated to the given TestStep or

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ProcedureStep wrapper. Commands such as get_iclock_list, get_iclock_option, and get_pattern_set_option are used to introspect the relevant information to echo to the screen.

- process_patterns_specification.pre_close

  Registers a Tcl proc that automatically runs immediately before the close_pattern_set command associated to every TestStep or ProcedureStep wrapper found in the processed PatternsSpecification.

  The proc must be declared with these arguments: {config_path args}. The config_path argument holds the path to the TestStep or ProcedureStep wrapper being processed.

  This callback is useful to add debug code that introspects the end state of the network after having run the iCalls associated to the given TestStep or ProcedureStep wrapper. Commands such as get_iclock_list, get_iclock_option, and get_pattern_set_option are used to introspect the relevant information to echo to the screen.

- process_patterns_specification.pre_write

  Registers a Tcl proc that automatically runs immediately before the write_patterns command is issued by the process_patterns_specification command for each Patterns wrapper processed of the PatternsSpecification.

  The proc must be declared with these four arguments: {pattern_type config_path command_line args}. The pattern_type is a single word that identifies the pattern file being generated. It can have the value “pdl”, “verilog” or any actual format supported by the manufacturing_patterns_formats property in the PatternsSpecification wrapper.

  The config_path argument holds the configuration data path Patterns wrapper being processed.

  The command_line is a string that contains the full write_patterns command.

  The args argument is a placeholder for backwards compatibility in case more arguments are added in the future.

  This callback can be used to change the set_write_patterns_options for example.

- set_context.post

  Registers a Tcl proc that automatically runs after the context was changed.

  The proc must be declared with these arguments: {context args}. The context argument hold the name of the context that was just entered.

  This callback is useful to automatically set the context upon entering that context.

- set_context.pre

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Registers a Tcl proc that automatically runs before the context is changed.

The proc must be declared with these arguments: {context args}. The context argument holds the name of the context that is to be entered.

This callback is useful to automatically perform checks that you want to enforce prior to changing context. To block the context change, issue the error condition with the display_message command and terminate the proc with the “return -code error” command.

- set_current_mode.post

  Registers a Tcl proc that automatically runs after the current mode was changed.

  The proc must be declared with these arguments: {mode_name args}. The mode_name argument hold the name of the mode that was just set.

  This callback is useful to automatically perform automatic setting when the current mode was changed. See the set_current_mode command for more information about the current mode setting.

- set_current_mode.pre

  Registers a Tcl proc that automatically runs before the current mode is changed.

  The proc must be declared with these arguments: {mode_name args}. The mode_name argument hold the name of the mode that is to be set.

  This callback is useful to automatically perform checks that you want to enforce prior to changing the current the mode name. To block the mode name change, issue the error condition with the display_message command and terminate the proc with the “return -code error” command.

- system_mode_transition.post

  Registers a Tcl proc that automatically runs after the system mode was changed.

  The proc must be declared with these arguments: {from to args}. The “from” argument hold the name of the system mode prior to the system mode transition while the “to” argument holds the name of the system mode after the system mode transition.

  This callback is useful to automatically perform automatic setting automatically after the system mode was changed. Use the get_context command to configure your actions to specific contexts.

- system_mode_transition.pre

  Registers a Tcl proc that automatically runs before the system mode is changed.

  The proc must be declared with these arguments: {from to args}. The “from” argument hold the name of the system mode prior to the system mode transition

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

while the “to” argument holds the name of the system mode after the system mode transition.

This callback is useful to automatically perform checks that you want to enforce prior to changing the system mode. To block the system mode transition, issue the error condition with the display_message command and terminate the proc with the “return -code error” command. Use the get_context command to configure your actions to specific contexts.

- write_edt_files.post_insertion

  Registers a Tcl proc that automatically runs after the insertion of the EDT instruments was performed within the write_edt_files command.

  The proc does not take any arguments.

  This callback is useful to automatically apply custom edits to the design after EDT instruments were inserted but prior to when the modified design is written out into the dft_inserted_designs directory.

- ***proc_name***

  Specifies the name of the proc to call when the specified feature is happening. The proc name can be with a fully specified namespace prefix or not. When the proc name does not start with ::, it is search relative to the current namespace. Using “[namespace current]::proc_name” is equivalent to specifying proc_name.

- -order *integer*

  Specifies the execution order relative to the other callbacks registered for the same feature. When unspecified, the order defaults to 5. You can specify any integer between 1 and 10. A callback with an order value of *N* is guaranteed to run prior to the callbacks having an order value greater than *N*. For callbacks having the same order value, they are run in the order they were registered. It is not possible to control the registration order for two callbacks declared into two separate plugin directories so when the order is important, specify appropriate order values.

- -static_arguments *list_of_argument_values*

  Specifies a list of argument values to pass to the proc in front of the normal arguments being passed.

  This feature is useful to reuse a common proc for two different callback features. You use the values specified in this list to specialize your proc. In the feature_identifier section above, the list of arguments the proc must have for each feature name is documented.

  For example, the argument list of the proc associated to the process_dft_specification.post_insertion feature is {root_wrapper args}. If you specified an static argument list with two values, you would need to declare two new variable in front of the root_wrapper argument as follow {static_arg1 static_arg2 root_wrapper args}. The actual name of the argument is not important but their position and count must match the specified list_of_argument_values.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

This example below registers a proc to run before the system mode transition. When it detects that the context is dft and the transition is from setup to analysis, it deletes the existing Design(*design_name*,*design_id*) wrapper if it already exists and creates a new empty one. With this callbacks, the wrapper is guaranteed to exist and be empty when entering the analysis system mode in dft context. For this code to be available whenever Tessent Shell runs, it is included in the file *tcl_modules/your_company_name/any_name.tcl* file in one of your plugin directory.

```
proc reset_design_wrapper { from_mode to_mode args } {
  if {[get_context] eq "dft" && $from_mode eq "setup" && $to_mode eq "analysis"} {
    set cur_design [get_single_name [get_current_design]]
    set design_id [get_context -design_id]
    if {[get_config_value Design("$cur_design",”$design_id”) -exists]} {
      delete_config_element Design("$cur_design",”$design_id”)
    }
    add_config_element Design("$cur_design",”$design_id”)
  }

register_callback system_mode_transition.pre reset_design_wrapper
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

register_drc

Context: dft, patterns, unspecified

Mode: setup

A command used to register a custom DRC rule, and make it look and behave as a built-in DRC rule fully supported by the analyze_drc_violation command and the Tessent Visualizer GUI.

Usage

register_drc ***name*** **-class** ***class_name*** **-short_help** ***short_help***
-drc_proc ***drc_proc*** **-context_list** ***context_list***
[-summary_text *summary_text*]
[-severity {error | warning | note | ignore}]
[-lowest_downgradable_severity {error | warning | note | ignore}]
[-verbose {auto | on | off}] [-rtl_synthesis_proc *rtl_synthesis_proc*]
[-runs_after *drc_name*] [-allow_auto_fix]

Description

A command used to register a custom DRC rule, and make it look and behave as a built-in DRC rule fully supported by the analyze_drc_violation command and the Tessent Visualizer GUI. Tessent Shell includes several high-level commands such as trace_flat_model and set_attribute_value giving you the ability to code in a few lines of Tcl code very complex DRC rules that run quickly and perform arbitrary complex custom checks. A comprehensive set of examples is provided below. The examples describe the complete implementation of four different DRC rules and provide the complete source code to serve as the basis for your own custom rules.

Arguments

- ***name***

  A required string that is appended to the DRC class name to make up a unique name for the DRC. Each registered DRC must have a unique <*class_name*><*name*> value. You typically use an integer as the name, and a string that identifies your company name and DRC type in the class name.

  For example, the DRC rules presented in the examples below have a class name of “XYZ_S” and a name from 1 to 4. The complete DRC names are therefore “XYZ_S1”, “XYZ_S2”, “XYZ_S3”, and “XYZ_S4”. It is important to use a suffix like “XYZ_” in your class names to make sure you do not collide with built-in DRC names, which do not have such prefixes. For example, even if DRC C27 does not yet exist as a built-in DRC name, it might exist in the future and cause your plugin to start failing when you load it into the newer version of Tessent Shell.

  By default, the order of execution of the user-defined DRC rules is extracted by doing a numerical sort on the DRC name. To precisely control the order of execution, use the -runs_after switch described below.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-class *class_name***

  A required switch-string pair that identifies the class name in which the DRC belongs. You must pre-register the *class_name* using the register_drc_class command.

- **-short_help *short_help***

  A required switch-string pair that provides a one line description of the DRC rule. This string is used to identify the DRCs in the DRC Browser of the GUI and when the report_drc_rules command is called. See the examples below for an illustration of the wording you would typically use.

- **-drc_proc *drc_proc***

  A required switch-string pair defining the name of a proc that is to be called to perform the DRC. As is described in Example 1of the plugin directory section, you use the following:

  ```
  -drc_proc [create_wrapper_proc my_tcl_proc]
  ```

  when you are packaging your custom DRC in a plugin. The created wrapper proc adds lots of functionality to your proc such as proper Ctrl-C handling, showing the Tcl-Stack if you have a bug in your code, and most importantly loading the body of your rule the first time it is used instead of when Tessent Shell starts. The examples below illustrates the use of this switch with the use of the create_wrapper_proc command.

  The supplied proc must accept two arguments: mode and drc_name. The proc is called with the drc_name argument specifying the name of the DRC that called the proc. This argument is useful when you want to share a common proc for more than one DRC name. The drc_name value enables you to customize it for each DRC while keeping the common part common. The mode argument has two values: enabled and run. When it has the value “enabled”, the proc must return 1 if the DRC is enable and you want it to run. Otherwise you return 0. If and only if you returned 1 is it then called with mode equal to “run”. See the examples later for a suggested way to establish if a DRC is enabled or not.

  Another way to control when the DRC runs is to specify a default severity of ignore using the -severity switch described below. You then use set_drc_handling ***drc_name*** {error | warning | note} when you want it to run.

- **-context_list *context_list***

  A required switch-string pair that is used to define in which context the registered DRC is to be run. You typically use “dft -no_sub_context”, which means the DRC is only requested to run in the DFT context when no sub-context (for example, -scan or -test_points) was specified. If you choose to enable the DRC in some other contexts, you may not have the information needed for it. The XYZ_S3 DRC shown in Example 3 below depends on the attributes populated by running the DFT_C6 DRC. DFT_C6 is only permitted in dft context with no sub context specified. It would therefore not be possible to activate XYZ_S3 in any other sub-context as the information computed by DFT_C6 would not be available.

- -summary_text *summary_text*

  An optional switch-string pair that specifies a string that is used when a DRC summary line is echoed to reflect the presence of violations for the given DRC. By default, DRC rules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

with severity warning or below only echo to the summary line if there was at least one violation. If you do not specify this switch, then the ***short_help*** string is also used as the summary text, which is what is done in the examples shown later and in most built-in DRCs.

- -severity error | warning | note | ignore

  An optional switch-value pair specifying the default severity the DRC is to have when enabled. If the -lowest_downgradable_severity is specified with a lower value, then the severity can be downgraded in your dofile using the set_drc_handling command. It is always possible to increase the severity of a DRC. When unspecified, the severity defaults to "error".

  A way to create a DRC that never runs by default but can be activated with the set_drc_handling command is to register the DRC with a default severity of "ignore". When you want to activate the DRC, you use the "set_drc_handling *drc_name severity*" command to bring it up to the required severity.

- -lowest_downgradable_severity error | warning | note | ignore

  An optional switch-value pair specifying the lowest severity the DRC can be downgraded to. When this is unspecified, the lowest downgrade value is the default severity specified with the -severity switch. The lowest downgrade value cannot be higher than the default severity value specified by the -severity switch.

  Note that the severity of a DRC can always be upgraded. For example, even if the default severity is "warning", you can increase the severity to "error" using the set_drc_handling command.

- -verbose auto | on | off

  An optional switch that controls the verbosity of the DRC. When this is set to "auto", the DRC generates one Error message per violation if the severity is "error". Only a summary line showing the number of violations for the given DRC is reported for severity warning or below. Setting -verbose to "on" causes a note or warning to be reported for each violation even if the severity is below "error". Setting -verbose to "off" suppresses the individual error message for each violation and only a summary line with the violation count is reported, even if the severity is "error". To view the individual violations, use the "report_drc_rules ***drc_name***" for the list of violations for the given DRC name.

- -rtl_synthesis_proc *rtl_synthesis_proc*

  An optional switch-value pair that specifies the name of a proc that is called to determine if the DRC requires anything to be synthesized in order for the DRC to check the logic. As with the DRC proc, the RTL synthesis proc has two arguments: mode and drc_name. Just like with the DRC proc, the RTL synthesis proc is called with the drc_name argument specifying the name of the DRC that called the proc. This argument is useful when you want to share a common proc for more than one DRC name. In Example 1 below, you can see that this technique was used and a common proc called *xyz_synth.tcl* was used for all four XYZ_S DRC rules.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

As with the DRC proc, it is best to use the create_wrapper_proc to encapsulate your proc as follows:

```
-rtl_synthesis_proc [create_wrapper_proc my_tcl_proc2]
```

The mode argument has multiple values. The values and their meaning are described in Table 6-4.

Table 6-4. Mode values for the RTL Synthesis Proc

Mode Value	Meaning
enabled	In this mode, the proc returns 1 or 0 to indicate if it requires anything to be synthesized or not.
complete_start_module	In this mode, the proc returns a collection of modules that needs to be synthesized downward. All modules below the returned modules are also synthesized unless the lower module is returned in the specified “complete_stop_module” collection. It must return a null string when nothing needs to be synthesized completely. You typically return the current_design when doing DRC that deals with flip-flops because until a module is synthesized, it is not possible to know if it has flip-flops or not. The DRC XYZ_S3 in Example 3 below is an example of a DRC that requires full synthesis. When you are checking connections between top-level port or pins of instances, you do not need to synthesize everything; instead, you use the selective modes described below to request synthesizing logic in the fanin or fanout of specific nodes.
complete_stop_module	In this mode, the proc returns a collection of modules that stop the downward synthesis from the specified “complete_start_module” collection. For example, if you only want to see flip-flops in the top module, you would specify the current design as the “complete_start_module” and the modules of the instances found in the current_design as the “complete_stop_module”. The code to create the “complete_stop_module” collection would be “get_modules -of_instances [get_instances * -of_type design]”.
selective_fanin	In this mode, the proc returns a collection of pins or ports for which the fanin must be synthesized. For each node in the collection, the fanin is traced until either a net with no source, a top-level PI, or a sequential element is reached. If a net that is part of an RTL expression is reached, its associated module is synthesized, and the preceding tracing process is recursively repeated. The selective_fanin_stop_equation described next is another condition that stops the tracing.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 6-4. Mode values for the RTL Synthesis Proc (cont.)

Mode Value	Meaning
selective_fanin_stop_equation	In this mode, the proc returns a string that represent a filtering equation based on port and pin attributes. When the equation evaluation is true, the fanin traversal is stopped at this point. The usage of such equation is illustrated in the RTL synthesis proc shown in Example 1.
selective_fanout	In this mode, the proc returns a collection of pins or ports for which their fanout must be synthesized. For each node in the collection, the fanout is traced until either a net with no destination, a top-level PO, or a sequential element is reached. If a net that is part of an RTL expression is reached, its associated module is synthesized, and the preceding tracing process recursively repeated.The selective_fanout_stop_equation described next is another condition that stops the tracing.
selective_fanout_stop_equation	In this mode, the proc returns a string that represent a filtering equation based on port and pin attributes. When the equation evaluation is true, the fanout traversal is stopped.

- -runs_after *drc_name*

  An optional switch-value pair that specifies the order a given DRC runs relative to the other DRC rules. You use this to create a sequential order of execution of the DRCs. When unspecified, the DRC runs after the DRC it follows when doing a numerical sort. Remember that each DRC has an independent way of specifying that it is enabled or not. Even if you specify -runs_after DRCX, the DRC runs after DRCX whether DRCX is enabled or not.

- -allow_auto_fix

  An optional Boolean switch that enables auto fix support within this DRC. When you use this switch, the "set_drc_handling ***drc_name***" command supports the -auto_fix on | off switch. The default value is on. Use "get_drc_handling ***drc_name*** -auto_fix" to determine if auto fix is enabled or disabled for a specific DRC. The rest of the work is in the coding of the DRC. Under certain conditions, when auto fix is on, you can choose to implement an automatic fix instead for given a violation. See DFT_C9 for an example of a DRC rule that supports auto fix. Instead of giving a violation, the tool infers a DFT control point to make the set or reset sources controllable during shift.

Examples

The following examples are a set of four custom DRC rules used to provide you with guidance for coding your own rules. The four examples were chosen to cover a wide spectrum of the available infrastructure. Some of the commands and procedures described in Example 1 are re-used in the other three examples.

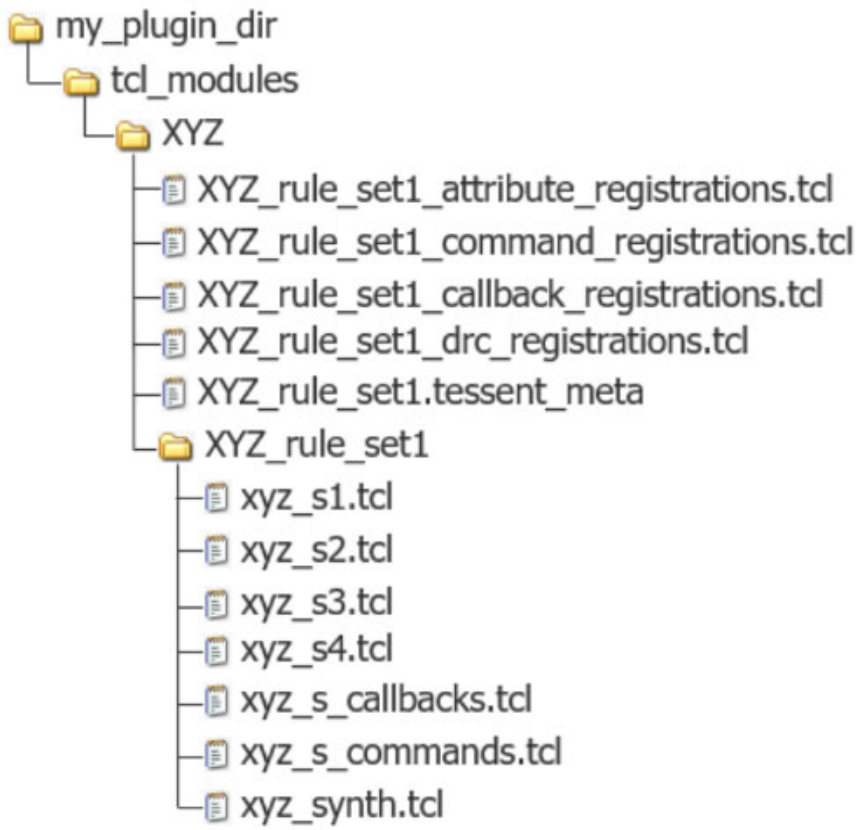
The four DRCs rules are packaged in a *tcl_modules* plugin directory following the same organization described in the plugin directory section. The directory and files described in the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

examples below follow the organization presented in Figure 6-4. The files located in the *XYZ* directory are loaded when Tessent Shell starts (assuming the TESSENT_PLUGIN_PATH environment variable points to the *my_plugin_dir* directory). The files inside the *XYZ_rule_set1* sub-directory are loaded the first time the DRC rules run. As described in the plugin directory section, you must perform the registrations and the metadata declarations in the files that are loaded into Tessent Shell at start time. The body of the procs that implement the commands, the callbacks, and the DRC rules are loaded on demand the first time the command, callback, or DRC, respectively, is run. The four example DRC rules are as follows:

- Example 1 — XYZ_S1: All ports must go through a buffer before any other logic
- Example 2 — XYZ_S2: No combinational path except pure feed-through paths
- Example 3 — XYZ_S3: No inter-domain path not going through sync cells
- Example 4 — XYZ_S4: Input pins must be sourced by output pins with consistent xyz_function

Figure 6-4. Plugin Directory Contains XYZ Custom DRC Rules



Example 1

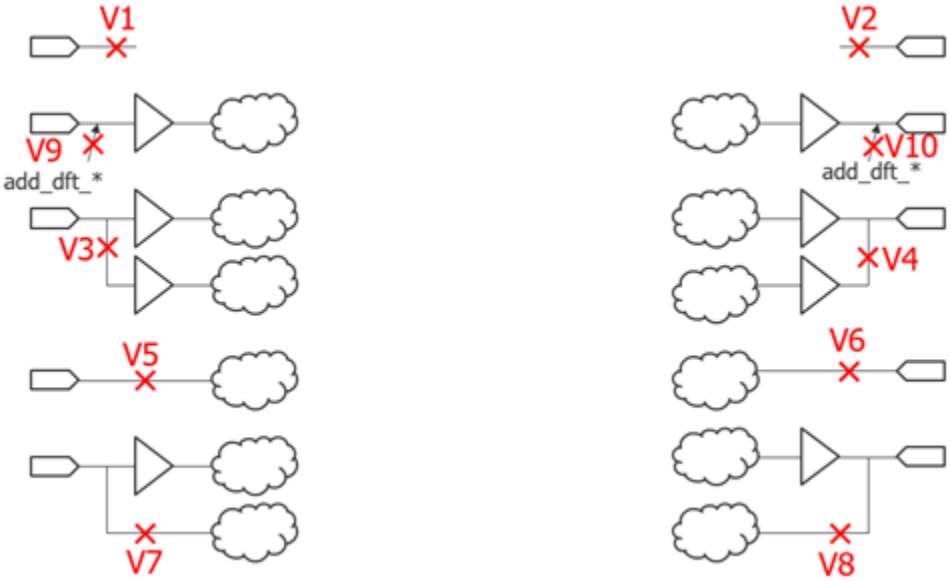
This example implements a DRC rule that verifies that each port of the current design connects to one and only one buffer cell. Some design groups have the practice of having a single buffer connected to each port of a physical layout region such that it can be placed near the physical

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

port of the layout region ensuring that there is no long unbuffered net inside the layout region. Some ports like a clock port are excluded automatically from this check because their buffering is handled during Clock Tree Synthesis, which happens later in the flow during layout.

The types of violation that must be detected by the DRC rule are illustrated in Figure 6-5. The V1 and V2 violations detect unconnected ports. The V3 and V4 violations detect ports connected to more than one buffer cells. The V5 and V6 violations detect ports connected to logic without going through a buffer cell. The V7 and V8 violations detect ports connected to a single buffer cell but also to other logic. The V9 and V10 detects ports properly connected to a single buffer but which were the target of a add_dft_control_points or add_dft_clock_mux command. The logic is not yet inserted but after the process_dft_specification command is run, logic is inserted, and the port is no longer directly connected to a buffer cell.

Figure 6-5. Violation Detected by XYZ_S1 DRC



The following is the content of the *XYZ_rule_set1_command_registrations.tcl* file. It contains the registration of commands used to configure or report the settings of the XYZ_S DRC rules. The file also contains a namespace import of the ::mentor::DrcUtilities::* procedure that is used to facilitate the coding of the custom DRC rules. The imported commands are described later when they are used in the DRC body description. Finally, the file declares a namespace variable that is used to store a Tcl dictionary holding the enable status as well as other options for the DRC rules.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
#### XYZ_rule_set1_command_registrations.tcl ####
namespace eval XYZ_rule_set1 {

  #### Command imports

  # ITCL classes
  #  DrcViolation
  #
  # Procs
  #  get_add_dft_node_filter
  #       returns a string used to filter gate_pin collection to know
  #       if they were the target of an add_dft_* command
  #  get_add_dft_node_message
  #       returns a string explaining which add_dft_* command was
  #       issued on the node
  #  get_synchronous_clock_label_groups

  namespace import ::mentor::DrcUtilities::*

  #### Command registration

  register_tcl_command set_xyz_s_drc_options \
     -context_list {unspecified dft patterns} \
     -system_mode_list {setup} \
     -string_option {{<enable_drc_index_range_list>} {-enable_drcs}\
                     {enable_drc_index_range_list}} \
     -string_option {{<disable_drc_index_range_list>} {-disable_drcs}\
                     {disable_drc_index_range_list}} \
     -string_option {{<excluded_ports_from_xyz_s1>} \
                     {-excluded_ports_from_xyz_s1}\
                     {excluded_ports_from_xyz_s1}} \
     -string_option {{<excluded_clocks_from_xyz_s3>} \
                     {-excluded_clocks_from_xyz_s3} \
                     {excluded_clocks_from_xyz_s3}} \
     -string_option {{<synchronizer_cells>} {-synchronizer_cells}\
                     {synchronizer_cells}} \
     -syntax_bnf {[<enable_drc_index_range_list>] \
                  [<disable_drc_index_range_list>]\
                  [<excluded_ports_from_xyz_s1>]  \
                  [<excluded_clocks_from_xyz_s3>] \
                  [<synchronizer_cells>]} \
     -transcript tool_command \
     -tcl_proc "[create_wrapper_proc set_xyz_s_drc_options]"

  register_tcl_command report_xyz_s_drc_options \
     -context_list {unspecified dft patterns} \
     -system_mode_list {setup analysis insertion} \
     -syntax_bnf {} \
     -transcript tool_command \
     -tcl_proc "[create_wrapper_proc report_xyz_s_drc_options]"
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# Variable definitions
  variable xyz_s_drc_dict {
    drc_indexes {
      1 {
        enable 0
        excluded_ports {}
      }
      2 {
        enable 0
      }
      3 {
        enable 0
        excluded_clocks {}
      }
      4 {
        enable 0
      }
    }
    synchonizer_cells {}
  }
}
```

The next section of code includes the content of the file *xyz_s_commands.tcl*. The file contains the body of the procs used to implement the registered commands. There is nothing specific to DRC rules in this file. and this is simply provided for completeness. You can use this example to see a use of the Tcl “dict” commands and how Tcl dictionaries are perfect to store structured data.

```
###### xyz_s_commands.tcl #####
proc report_xyz_s_drc_options {args} {
  variable xyz_s_drc_dict
  set drc_indexes_dict [dict get $xyz_s_drc_dict drc_indexes]
  puts "------------------"
  puts "DRC XYZ_S* Options"
  puts "------------------"
  foreach drc_index [dict keys $drc_indexes_dict] {
    set enable [dict get $drc_indexes_dict $drc_index enable]
    if {$enable} {
      puts "[string range "XYZ_S${drc_index}    " 0 8] : On"
      report_xyz_s1_excluded_ports $drc_index
      report_xyz_s1_excluded_clocks $drc_index
    } else {
      puts "[string range "XYZ_S${drc_index}    " 0 8] : Off"
    }
  }
  report_xyz_s_synchronizer_cells
  puts "------------------"
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
##################################################################
proc report_xyz_s_synchronizer_cells {} {
  variable xyz_s_drc_dict
  set synchronizer_cells [dict get $xyz_s_drc_dict synchronizer_cells]
  if {[sizeof_collection $synchronizer_cells] > 0} {
    puts "Synchronizer cells : [join [get_name_list $synchronizer_cells]
", "]"
  }
}
##################################################################
proc report_xyz_s1_excluded_ports {drc_index} {
  variable xyz_s_drc_dict
  if {[get_resource design_is_elaborated] && $drc_index == 1} {
    set excluded_ports [dict get $xyz_s_drc_dict \
                                drc_indexes 1 excluded_ports]
    if {[sizeof_collection $excluded_ports] > 0} {
      puts "  Excluded ports:"
      puts "    [join [get_name_list $excluded_ports] "\n    "]"
    }
  }
}
##################################################################
proc report_xyz_s1_excluded_clocks {drc_index} {
  variable xyz_s_drc_dict
  if {[get_resource design_is_elaborated] && $drc_index == 3} {
    set excluded_clocks [dict get $xyz_s_drc_dict drc_indexes 3
excluded_clocks]
    if {[sizeof_collection $excluded_clocks] > 0} {
      puts "  Excluded Clocks:"
      puts "    [join [get_name_list [get_design_objects -from_objects
$excluded_clocks -map_pseudo_ports]] "\n    "]"
    }
  }
}
##################################################################
proc set_xyz_s_drc_options {args} {
  array set ARGS $args
  set error_status 0
  if {[info exists ARGS(enable_drc_index_range_list)]} {
    set_xyz_s_drc_enables 1 $ARGS(enable_drc_index_range_list)
  }
  if {[info exists ARGS(disable_drc_index_range_list)]} {
    set_xyz_s_drc_enables 0 $ARGS(disable_drc_index_range_list)
  }
  if {[info exists ARGS(excluded_ports_from_xyz_s1)]} {
    set_xyz_s1_drc_excluded_ports $ARGS(excluded_ports_from_xyz_s1)
  }
  if {[info exists ARGS(excluded_clocks_from_xyz_s3)]} {
    set_xyz_s3_drc_excluded_clocks $ARGS(excluded_clocks_from_xyz_s3)
  }
  if {[info exists ARGS(synchronizer_cells)]} {
    set_xyz_s_synchronizer_cells $ARGS(synchronizer_cells)
  }
  if {$error_status} {
    return -code error
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
##################################################################
proc set_xyz_s_drc_enables {value drc_index_range_list} {
  variable xyz_s_drc_dict
  upvar error_status error_status
  set drc_indexes_dict [dict get $xyz_s_drc_dict drc_indexes]
  set drc_index_list [dict keys $drc_indexes_dict]
  if {$value} {
    #Remove already enabled ones when enabling new ones.
    foreach drc_index $drc_index_list {
      dict set xyz_s_drc_dict drc_indexes $drc_index enable 0
    }
  }
  foreach drc_index_range $drc_index_range_list {
    if {[regexp {^([0-9])+(?:-([0-9]+))?$} $drc_index_range -> \
                 index0 index1]} {
      if {$index1 eq ""} {
        set index1 $index0
      }
      if {$index1 < $index0} {
        set incr_val -1
        set stop_condition "\$i >= \$index1"
      } else {
        set incr_val 1
        set stop_condition "\$i <= \$index1"
      }
      for {set i $index0} $stop_condition {incr i $incr_val} {
        if {$i in $drc_index_list} {
          dict set xyz_s_drc_dict drc_indexes $i enable $value
        }
      }
    } else {
      display_message -error "DRC index range '$drc_index_range' does not
match # or #-#."
      set error_status 1
    }
  }
}
##################################################################
proc set_xyz_s1_drc_excluded_ports {excluded_ports_from_xyz_s1} {
  variable xyz_s_drc_dict
  upvar error_status error_status
  if {![get_resource design_is_elaborated]} {
    display_message "The -excluded_ports_from_xyz_s1 can only be used
after the design is elaborated."
    set error_status 1
  } else {
    set port_objects [check_current_design_port_spec
$excluded_ports_from_xyz_s1 "<excluded_ports_from_xyz_s1>"]
    dict set xyz_s_drc_dict drc_indexes 1 excluded_ports $port_objects
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
#################################################################
proc check_current_design_port_spec {current_design_port_spec
source_string} {
  upvar error_status error_status
  if {[is_collection $current_design_port_spec]} {
    if {[sizeof_collection $current_design_port_spec] == 0} {
      return {}
    }
    set non_port_objects [filter_collection $current_design_port_spec\
                                            "object_type != port"]
    if {[sizeof_collection $non_port_objects] > 0} {
      display_message -error \
       "The supplied $source_string collection contains elements which are
not port objects:\n  [join [get_name_list $non_port_objects] "\n  "]"
      set error_status 1
      return {}
    }
    return $current_design_port_spec
  } else {
    set returned_collection {}
    foreach name_pattern $current_design_port_spec {
      set port_objects [get_ports [list $name_pattern] -silent]
      if {[sizeof_collection $port_objects] == 0} {
        display_message -error \
         "The name_pattern '$name_pattern' supplied in $source_string does
not match any port on the current design."
        set error_status 1
      } else {
        append_to_collection returned_collection $port_objects
      }
    }
    return $returned_collection
  }
}
#################################################################
proc set_xyz_s3_drc_excluded_clocks {excluded_clocks_from_xyz_s3} {
  variable xyz_s_drc_dict
  upvar error_status error_status
  if {![get_resource design_is_elaborated]} {
    display_message "The -excluded_clocks_from_xyz_s3 can only be used
after the design is elaborated."
    set error_status 1
  } else {
    set clock_objects [check_clock_spec $excluded_clocks_from_xyz_s3\
                                        "<excluded_clocks_from_xyz_s3>"]
    dict set xyz_s_drc_dict drc_indexes 3 excluded_clocks $clock_objects
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
#################################################################
proc check_clock_spec {clock_spec source_string} {
  upvar error_status error_status
  if {[is_collection $clock_spec]} {
    if {[sizeof_collection $clock_spec] == 0} {
      return {}
    }
    set non_clock_objects {}
    foreach_in_collection object $clock_spec {
      if {[get_clock_option $object -type -silent] eq ""} {
        append_to_collection non_clock_objects $object
      }
    }
    if {[sizeof_collection $non_clock_objects] > 0} {
      display_message -error \
       "The supplied $source_string collection contains elements which are
not clock objects:\n  [join [get_name_list $non_clock_objects] "\n  "]"
      set error_status 1
      return {}
    }
    return $clock_spec
  } else {
    set returned_collection {}
    foreach name_pattern $clock_spec {
      set clock_objects [get_clocks [list $name_pattern] -silent]
      if {[sizeof_collection $clock_objects] == 0} {
        display_message -error \
         "The name_pattern '$name_pattern' supplied in $source_string does
not match any clock."
        set error_status 1
      } else {
        append_to_collection returned_collection $clock_objects
      }
    }
    return $returned_collection
  }
}
#################################################################
proc set_xyz_s_synchronizer_cells {synchronizer_cells_spec} {
  variable xyz_s_drc_dict
  upvar error_status error_status
  set synchronizer_cells_objects \
      [check_module_cell_spec $synchronizer_cells_spec \
                              "<synchronizer_cells>"]
  dict set xyz_s_drc_dict synchronizer_cells $synchronizer_cells_objects
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
#####################################################################
proc check_module_cell_spec {module_cell_spec source_string} {
  upvar error_status error_status
  if {[is_collection $module_cell_spec]} {
    if {[sizeof_collection $module_cell_spec] == 0} {
      return {}
    }
    #Check that these are all modules
    set non_module_objects [filter_collection $module_cell_spec
"object_type != module"]
    if {[sizeof_collection $non_module_objects] > 0} {
      display_message -error \
       "The supplied $source_string collection contains elements which are
not module objects:\n  [join [get_name_list $non_module_objects] "\n  "]"
      set error_status 1
      return {}
    }
    #Now that I know they are all modules, check if they are all of
    #type cells.
    set non_cell_modules [filter_collection $module_cell_spec \
                                            "type != cell"]
    if {[sizeof_collection $non_cell_modules] > 0} {
      display_message -error \
       "The supplied $source_string collection contains modules which are
not of type cell:\n  [join [get_name_list $non_cell_modules] "\n  "]"
      set error_status 1
      return {}
    }
    return $module_cell_spec
  } else {
    set returned_collection {}
    foreach name_pattern $module_cell_spec {
      set module_objects [get_modules [list $name_pattern] -silent]
      if {[sizeof_collection $module_objects] == 0} {
        display_message -error \
         "The name_pattern '$name_pattern' supplied in $source_string does
not match any module."
        set error_status 1
      } else {
        set non_cell_modules [filter_collection $module_objects
                                                "type != cell"]
        if {[sizeof_collection $non_cell_modules] > 0} {
          display_message -error \
            "The name_pattern '$name_pattern' supplied in $source_string
matches modules which are not of type cell:\n  [join [get_name_list
$non_cell_modules] "\n  "]"
          set error_status 1
        } else {
          append_to_collection returned_collection $module_objects
        }
      }
    }
    return $returned_collection
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The next code block shows the content of the file *XYZ_rule_set1_drc_registrations.tcl*. This file includes the registration of the four XYZ_S DRC rules. The first line registers the XYZ_S DRC class using the register_drc_class command. Below are the four DRC rule registrations using the register_drc command. Note how the create_wrapper_proc command is used around the referenced proc names. The wrapper proc provides a lot of built-in functionality to your proc such as proper Ctrl-C handling, dumping the stack if you have a Tcl bug in you code, and most importantly, loading the body of the procs the first time the DRC runs. Because the create_wrapper_proc is called inside namespace "XYZ_rule_set1", the "*.*tcl*" files found inside the subdirectory by the name of the namespace are loaded the first time the wrapper proc is called.

```
###### XYZ_rule_set1_drc_registrations.tcl #####

namespace eval XYZ_rule_set1 {

    # DRC registrations

    register_drc_class XYZ_S
    register_drc 1 \
       -class XYZ_S \
       -short_help "All ports must go through a buffer before any other
logic" \
       -context_list "dft -no_sub_context" \
       -drc_proc [create_wrapper_proc XYZ_S1_drc] \
       -rtl_synthesis_proc [create_wrapper_proc XYZ_S_synth]

    register_drc 2 \
       -class XYZ_S \
      -short_help "No combinational path except pure feed-through paths" \
       -context_list "dft -no_sub_context" \
       -drc_proc [create_wrapper_proc XYZ_S2_drc] \
       -rtl_synthesis_proc [create_wrapper_proc XYZ_S_synth]

    register_drc 3 \
       -class XYZ_S \
       -short_help "No inter-domain path not going through sync cells" \
       -context_list "dft -no_sub_context" \
       -drc_proc [create_wrapper_proc XYZ_S3_drc] \
       -rtl_synthesis_proc [create_wrapper_proc XYZ_S_synth]

    register_drc 4 \
       -class XYZ_S \
       -short_help "Input pins must be sourced by output pins with
consistent xyz_function" \
       -context_list "dft -no_sub_context" \
       -drc_proc [create_wrapper_proc XYZ_S4_drc] \
       -rtl_synthesis_proc [create_wrapper_proc XYZ_S_synth]
}
```

The next code block section shows the content of the *xyz_s1.tcl* file which contains the body definition of the XYZ_S1 DRC rule. The first proc is the main proc that has the mode and drc_name arguments. When called with mode equal to "enabled", the proc returns the enable flag of the XYZ_S1 DRC found in the *xyz_s_drc_dict*. The command used to populate the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dictionary was presented above in the second code block of Example 1 and is found in file *xyz_s_commands.tcl*. The proc that actually checks if any of the situations described in Figure 6-5 are present is the second proc named "XYZ_S1_check_port". When this proc detects an error condition, it returns a Tcl dictionary with the following keys: "path_list0", "path_list1", and "msg_list". The returned value is then used to add the DRC violation with the add_drc_violation command. As stated in the add_drc_violation command section, it requires one argument, which is an object of an ITCL class having a prescribed set of methods. A complete ITCL class definition is already provided in the tool, and you typically never need to define your own as the definition is very flexible and should be enough to cover most of your needs. See the add_drc_violation command section for a detailed description of this ITCL class and the method it provide to configure it and to service the DRC commands.

You create a new object of the DrcViolation class using the following command:

```
set violation [DrcViolation #auto]
```

You use the "dict_set" method of the object stored in the "violation" variable to configure the object. Once configured, you add the object as a violation using the following command:

```
add_drc_violation [namespace current]::$violation
```

This DRC is not doing complex tracing, and the tracing does not need to traverse cells, nor does the tracing need to consider simulation values. This is why the "XYZ_S1_check_port" proc makes use of the get_fanins and get_fanouts commands instead of the trace_flat_model command. In such simple situation, using the get_fanins/get_fanouts is better because the commands operate on the hierarchical data model. If you have a port that goes down through several levels of hierarchy but ends up having no fanout to a cell, then the port is seen as having no destination in the flat model; however, because this DRC is using get_fanouts, the violation highlights the path to the internal pin, which has no further connection. In more complex DRC like the other three examples, using the trace_flat_model command is needed because the DRCs involve tracing through multiple levels of logic and must consider a simulation context.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
##### xyz_s1.tcl #####
proc XYZ_S1_drc {mode drc_name} {
  variable xyz_s_drc_dict
  if {$mode eq "enabled"} {
    return [dict get $xyz_s_drc_dict drc_indexes 1 enable]
  }
  set excluded_ports [dict get $xyz_s_drc_dict drc_indexes 1\
                              excluded_ports]
  # Ports defined as clocks are auto excluded
  append_to_collection excluded_ports \
       [filter_collection [get_clocks -silent] "object_type==port"]
  foreach direction {forward backward} {
    if {$direction eq "forward"} {
      set ports_to_trace \
       [remove_from_collection [get_ports -direction input] \
                              $excluded_ports]
    } else {
      set ports_to_trace \
       [remove_from_collection [get_ports -direction output] \
                              $excluded_ports]
    }
    foreach_in_collection port_to_trace $ports_to_trace {
      set return_dict [XYZ_S1_check_port $port_to_trace \
                                         direction $direction]
      if {[dict size $return_dict] > 0} {
        # There was a violation
        set violation [DrcViolation #auto]
        set path_list0 [dict get $return_dict path_list0]
        $violation dict_set list_of_paths side0 $path_list0
        $violation dict_set path_direction side0 $direction
        if {[sizeof_collection $path_list0] > 1} {
          $violation dict_set end_callout_string side0 \
                              "Invalid connection"
        }
        if {[dict exists $return_dict path_list1]} {
          set path_list1 [dict get $return_dict path_list1]
          $violation dict_set list_of_paths side1 $path_list1
          $violation dict_set path_direction side1 $direction
          if {[sizeof_collection $path_list1] > 1} {
            $violation dict_set end_callout_string side1 \
                              "Second invalid connection"
          }
        }
        $violation dict_set message_string_list \
            [dict get $return_dict msg_list]
        $violation dict_set start_callout_string side0 \
            "Port with XYZ_S1 violation."
        add_drc_violation [namespace current]::$violation
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
#####################################################################
proc XYZ_S1_check_port {port args} {
  variable xyz_s_drc_dict
  array set ARGS {direction forward}
  array set ARGS $args
  set direction $ARGS(direction)
  set msg [XYZ_S1_check_object_has_no_add_dft_logic $port]
  if {[string length $msg] > 0} {
    dict lappend return_dict msg_list \
                        "The port '[get_single_name $port]' $msg"
    dict lappend return_dict path_list0 $port
    return $return_dict
  }
  if {$direction eq "forward"} {
    set objects [get_fanouts $port -stop_on cell_pin]
  } else {
    set objects [get_fanins $port -stop_on cell_pin]
  }
  # Filter out ports and nets in fanout or fanout
  set pins [filter_collection $objects "object_type==pin"]
  if {[sizeof_collection $pins] == 0} {
    # There are no pins
    dict lappend return_dict msg_list \
         "The port '[get_single_name $port]' is not connected to a pin."
    dict lappend return_dict path_list0 $port
    return $return_dict
  }
  if {[sizeof_collection $pins] > 1} {
    # There are more than one pin
    dict lappend return_dict msg_list \
         "The port '[get_single_name $port]' is connected to
'[sizeof_collection $pins]' pins:"
    set cnt 1
    set max 3
    foreach name [get_name_list $pins] {
       if {$cnt > $max} {
         dict lappend return_dict msg_list "  ..."
         break
       }
       dict lappend return_dict msg_list "  $name"
       incr cnt
    }
    #Show the first two in the schematic
    dict lappend return_dict path_list0 \
        [add_to_collection $port [index_collection $pins 0]]
    dict lappend return_dict path_list1 \
        [add_to_collection $port [index_collection $pins 1]]
    return $return_dict
  }
  # There is one pin
  # check if it was not the target of a add_dft_* command
  set msg [XYZ_S1_check_object_has_no_add_dft_logic $pins]
  if {[string length $msg] > 0} {
    dict lappend return_dict msg_list \
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
          "The port '[get_single_name $port]' is connected to a pin \
          '[get_single_name $pins]'"
      dict lappend return_dict msg_list "which ${msg}"
      dict lappend return_dict path_list0 [add_to_collection $port $pins]
      return $return_dict
    }
    set instance [get_instances -of_pins $pins]
    set simulation_function \
            [get_attribute_value_list $instance -name simulation_function]
    if {$simulation_function ne "buffer"} {
      #Not a pin of a buffer cell
      dict lappend return_dict msg_list \
         "The port '[get_single_name $port]' is connected to a pin \
'[get_single_name $pins]'"
      dict lappend return_dict msg_list \
        "which is not a pin of an instance with simulation_function equal to
'buffer'."
      dict lappend return_dict path_list0 [add_to_collection $port $pins]
      return $return_dict
    }
    return {}
  }

  ##########################################################################
  # This proc check if the add_dft_node_filter evaluates true on the object.
  # If it does, the object was the target of an add_dft_* command
  # The return list explains which one.
  # If it does not, an empty list is returned
  #
  proc XYZ_S1_check_object_has_no_add_dft_logic {object} {
    set object_with_add_dft_logic [filter_collection $object\
      "[get_add_dft_node_filter hier] || dft_clock_mux_destinations"]
    if {[sizeof_collection $object_with_add_dft_logic] > 0} {
      return [get_add_dft_node_message $object_with_add_dft_logic hier]
    } else {
      return [list]
    }
  }
```

Below is the source code used to request various parts of the design to be synthesized based on which DRC was enabled. The “drc_name” argument is used to decide what needs to be synthesized or not. The XYZ_S1, XYZ_S2, and XYZ_S4 DRC rules use the selective approach because they check the fanin and fanout of ports and pins. The XYZ_S3 DRC uses the complete synthesis methods because the DRC checks for inter-domain paths between sequential elements, and the sequential elements can only be identified after complete synthesis. Refer to the -rtl_synthesis_proc *rtl_synthesis_proc* argument section above for more details about the various argument values for this proc.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
##### xyz_synth.tcl #####
proc XYZ_S_synth {mode drc_name args} {
  variable xyz_s_drc_dict
  set drc_index [string range $drc_name 5 end]
  switch -exact $mode {
    "enabled" {
       return [dict get $xyz_s_drc_dict drc_indexes $drc_index enable]
    }
    "complete_start_module" {
      if {$drc_name in {XYZ_S3}} {
        return [get_current_design]
      }
    }
    "complete_stop_module" {
    }
    "selective_fanin" {
      if {$drc_name in {XYZ_S1}} {
        return [get_xyz_s1_selective_fanin_objects output]
      }
      if {$drc_name in {XYZ_S2}} {
        return [get_ports -direction output]
      }
      if {$drc_name in {XYZ_S4}} {
        return [get_xyz_s4_selective_fanin_objects]
      }
    }
    "selective_fanin_stop_equation" {
      if {$drc_name in {XYZ_S4}} {
        return "xyz_function"
      }
    }
    "selective_fanout" {
      if {$drc_name in {XYZ_S1}} {
        return [get_xyz_s1_selective_fanin_objects input]
      }
    }
    "selective_fanout_stop_equation" {
    }
  }
  return {}
}
#######################################################################
proc get_xyz_s1_selective_fanin_objects {direction} {
  variable xyz_s_drc_dict
  set excluded_ports [filter_collection [get_clocks -silent] \
                                        "object_type == port"]
  append_to_collection excluded_ports \
               [dict get $xyz_s_drc_dict drc_indexes 1 excluded_ports] \
               -unique
  return [remove_from_collection [get_ports -direction $direction]\
                                 $excluded_ports]
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

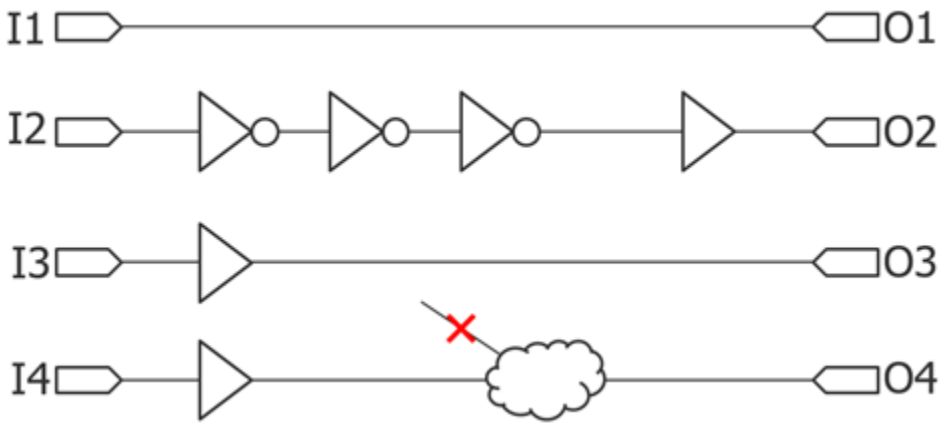
```
#######################################################################
proc get_xyz_s4_selective_fanin_objects {} {
  set attributed_objects \
           [get_attributed_objects -attribute_name xyz_function \
                                   -object_types {port pin}]
  return [filter_collection $attributed_objects \
                     "object_type == port && direction == output || \
                      object_type == pin && direction == input"]
}
```

Example 2

This example implements a DRC rule that verifies that the design does not have combinational path between input and output ports unless they are pure feed-through paths containing only single-input cells such as inverters and buffers. Figure 6-6 shows the kind of paths that need to be permitted or rejected. The example DRC rule only detects that a path has no multi-input gate along the way. If you also run XYZ_S1 presented in Example 1, you would reject the path from I1 to O1 because it has no buffer. You could also chose to write another DRC to make sure there is no inversion in the feed-through paths if this is one of your requirements. You would do this after you have found the path meeting XYZ_S2. You would use trace_flat_model with the "-controllability *controlling*" switch and use the "-store_polarity_to_tagged_objects" switch to store the polarity of the paths.

In the XYZ_S2 DRC, the trace_flat_model command is used to trace the connected fanin of every output ports and tag any path that reaches an input port of the current design. A simple filter_collection command using the "gate_input_count" Gate_pin attribute is used to detect if any multi-input gate exists along the path between the PO and the PI. As you can see below, the code to implement this DRC is very simple and fits in a page of code.

Figure 6-6. Combinational Paths Permitted and Rejected by XYZ_S2



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
#### xyz_s2.tcl #####
proc XYZ_S2_drc {mode drc_name} {
  variable xyz_s_drc_dict
  if {$mode eq "enabled"} {
    return [dict get $xyz_s_drc_dict drc_indexes 2 enable]
  }
  set output_ports [get_ports -direction output -silent]
  foreach_in_collection output_port $output_ports {
    set hits [trace_flat_model \
                 -from $output_port \
                 -direction backward \
                 -controllability connected \
                 -stop_at_first_tag on \
                 -tag_condition "primitive_name==PI" \
                 -store_paths_to_tagged_objects paths_to_tagged_objects]

    set path_to_tagged_objects [lindex $paths_to_tagged_objects 0]
    set multi_input_gates [filter_collection $path_to_tagged_objects \
                        "gate_input_count > 1 && direction == output"]
    set multi_input_cells {}
    set msg_list [list]
    if {[sizeof_collection $multi_input_gates] > 0} {
      append_to_collection multi_input_cells \
        [get_instances -of_gate_pins $multi_input_gates -silent] -unique
      set multi_input_cell_count [sizeof_collection $multi_input_cells]
      set violation [DrcViolation #auto]
      $violation dict_set list_of_paths side0 \
                                [list $path_to_tagged_objects]
      $violation dict_set path_direction side0 backward
      $violation dict_set start_callout_string side0 \
            "PO with path to PI with multi input gates along the path"
      $violation dict_set end_callout_string side0 "PI in fanin of PO"
      set start_port [index_collection $path_to_tagged_objects 0]
      set end_port [index_collection $path_to_tagged_objects \
               [expr [sizeof_collection $path_to_tagged_objects] - 1]]
      lappend msg_list "The output port '[get_single_name $start_port]'
has a combinational path from the input port '[get_single_name
$end_port]'"
     lappend msg_list "but the path is not a pure feed-through as the path
crosses $multi_input_cell_count multi-input cells:"
      set cnt 1
      set max 3
      foreach_in_collection multi_input_cell $multi_input_cells {
        if {$cnt > $max} {
          lappend msg_list "  ..."
          break
        }
        lappend msg_list "  [get_single_name $multi_input_cell]"
        incr cnt
      }
      $violation dict_set message_string_list $msg_list
      add_drc_violation [namespace current]::$violation
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

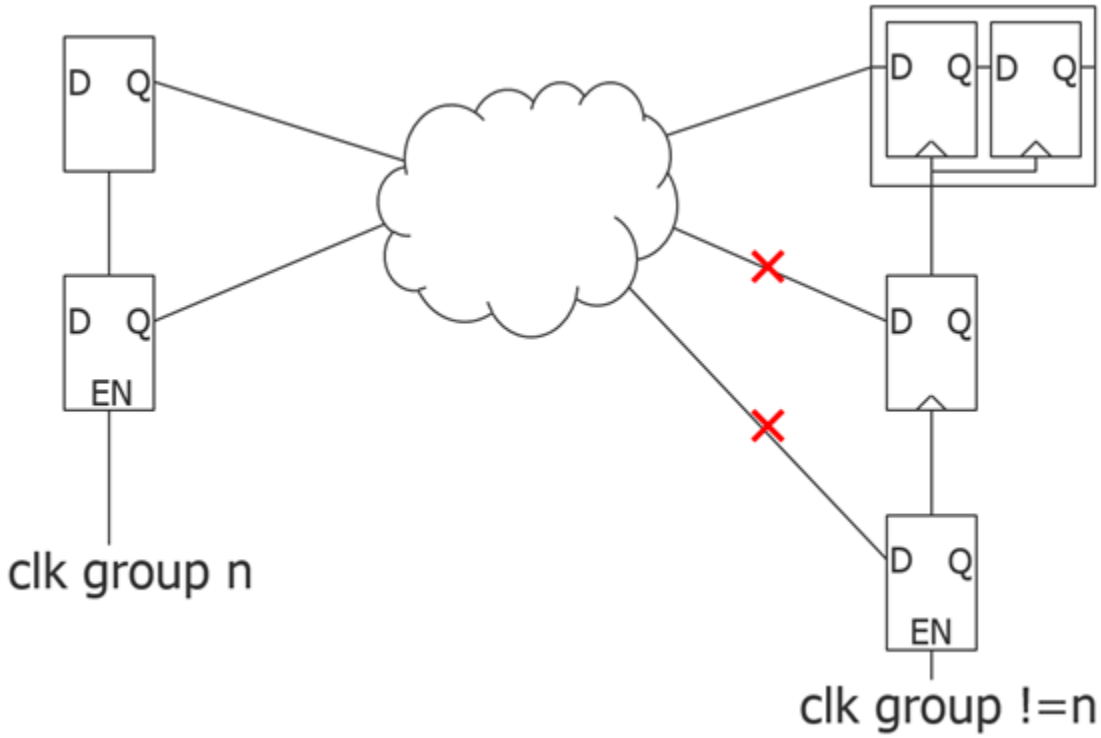
This example implements a DRC rule that verifies there is no path from a sequential cell on a given synchronous clock group that reaches a sequential cell on another synchronous clock group unless the destination sequential cell is a synchronizer cell. The types of violations that are to be found are illustrated in Figure 6-7. The implementation of this DRC is very simple once DFT_C6 has run and properly set the clock_domain_label attribute on each flip-flops. The implementation is shown in the example. You can see that it creates a collection of all DFF Q gate_pins. It then filters them per synchronous clock groups and runs one trace_flat_model command per groups. The hit equation is any DFF or DLAT gate_pin on a different synchronous clock group. The "xyz_exclude" attribute is used to tag the inputs of the synchronizer cells and any gate_pin of a sequential cell belonging to an excluded clock domain. If you go back to the example code block describing the content of the file *XYZ_rule_set1_command_registrations.tcl* in Example 1, you see that the "set_xyz_s_drc_options" command was equipped with an -excluded_clocks_from_xyz_s3 switch to exclude some clock domains from the check. For example, it is common to enable interactions between the IJTAG static signal (TCK domain) and the functional logic without going through a synchronizer cell. The "xyz_exclude" attribute was registered in the file *XYZ_rule_set1_attribute_registrations.tcl* file, and its content is shown below.

Finally, notice that the trace_flat_model command is used to do a forward walk. Figure 6-8 on page 1622 and Figure 6-9 on page 1622 illustrate the difference between using a forward versus a backward walk. The trace_flat_model command is very efficient and never traverses the same gate twice in a given trace. You would be visiting each gate potentially thousands of times if you traced through a gate as part of all possible paths it belongs to. When you use a backward trace, you are guaranteed to find all hits in the fanin, but not all the sources that may reach the hits. The same is true in the other direction. In this example DRC, we are interested in finding all the destination gates sourced by at least one sequential cell on a different domain. We do not need to find all the source elements, just the failing destinations which is why a forward tracing is used. If you wanted to find all elements sourcing a different domain and all elements receiving from a different domain, you would need to use two traces, one forward and one backward. The backward trace would identify the TX flops and the forward trace would identify the RX flop.

This DRC example also makes use of the register_callback command to implement a check when check_design_rules is called to make sure the synchronizer cell was defined using the set_xyz_s_drc_options command and that the DFT_C6 rule was turned on. The files to register the callback and the one containing the body definition are listed at the end of this example. Notice that DFT_C6 is enabled by using the "set_dft_specification_requirements -logic_test on" command. That also enables DFT_C9. If you want to ignore the asynchronous Set and Reset issues, you can disable DFT_C9 using the "set_drc_handling dft_c9 ignore" command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 6-7. Violation Detected by XYZ_S3




```
##### xyz_s3.tcl #####
proc XYZ_S3_drc {mode drc_name} {
  variable xyz_s_drc_dict
  if {$mode eq "enabled"} {
    return [dict get $xyz_s_drc_dict drc_indexes 3 enable]
  }
  set excluded_clocks [dict get $xyz_s_drc_dict drc_indexes 3
excluded_clocks]
  set sync_clock_label_groups [get_synchronous_clock_label_groups \
                                  -excluded_clocks $excluded_clocks]
  reset_attribute_value -object_type gate_pin -name xyz_exclude
  set synchronizer_modules [dict get $xyz_s_drc_dict synchronizer_cells]
  if {[sizeof_collection $synchronizer_modules] > 0} {
    set synchronizer_instances
           [get_instances -of_modules $synchronizer_modules -silent]
    if {[sizeof_collection $synchronizer_instances] > 0} {
      set synchronizer_cell_pins \
           [get_pins -of_instances $synchronizer_instances]
      set_attribute_value \
        [get_gate_pins -of_pins $synchronizer_cell_pins] -name xyz_exclude
    }
  }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set seq_q [get_gate_pins -of_gate_type sequential \
                          -filter {primitive_port_name==Q}]

  set_current_simulation_context stable_after_setup

  foreach sync_clock_label_group $sync_clock_label_groups {
    set equal_list [list]
    set not_equal_list [list]
    # Example sync_group =A, excluded_clocks = TCK
    # Start from all SEQ with "clock_domain_label == A"
    # Tag all seq element with "clock_domain_label != A &&
    #                           clock_domain_label!= TCK"
    foreach label $sync_clock_label_group {
      lappend equal_list     " clock_domain_label == $label "
      lappend not_equal_list " clock_domain_label != $label "
    }
    set filter        [join $equal_list     ||]
    if {[sizeof_collection $excluded_clocks] > 0} {
      foreach_in_collection excluded_clock $excluded_clocks {
        lappend not_equal_list " clock_domain_label != [get_clock_option
$excluded_clock -label] "
      }
    }
    set tag_condition [join $not_equal_list &&]
    #Stop on xyz_exclude to not traverse data pins of synchronyzer cells
    set hits [trace_flat_model -from [filter_collection $seq_q $filter] \
                   -direction forward \
                   -controllability unblocked \
                   -stop_condition "xyz_exclude" \
                   -tag_condition "primitive_name == DFF && \
                                   primitive_port_name == D && \
                                   $tag_condition && !xyz_exclude" \
                  -store_paths_to_tagged_objects paths_to_tagged_objects]
    foreach path_to_tagged_objects $paths_to_tagged_objects {
      set msg_list [list]
      set mapped_path_to_tagged_objects \
         [filter_collection $path_to_tagged_objects fault_site]
      set start_gate_pin [index_collection $path_to_tagged_objects 0]
      set start_clock_domain_label \
         [get_attribute_value_list $start_gate_pin \
                                   -name clock_domain_label]
      set end_gate_pin [index_collection $path_to_tagged_objects \
         [expr [sizeof_collection $path_to_tagged_objects] - 1]]
      set end_clock_domain_label \
         [get_attribute_value_list $end_gate_pin \
                                   -name clock_domain_label]
      set mapped_start_name [get_single_name [index_collection\
                             $mapped_path_to_tagged_objects 0]]
      set mapped_end_name   [get_single_name [index_collection \
                                 $mapped_path_to_tagged_objects \
          [expr [sizeof_collection $mapped_path_to_tagged_objects] - 1]]]
      set violation [DrcViolation #auto]
      $violation dict_set simulation_context \
                          [get_current_simulation_context]
      $violation dict_set list_of_paths side0 \
                          [list $path_to_tagged_objects]
      $violation dict_set path_direction side0 forward
      $violation dict_set start_callout_string side0 "Source sequential
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
cell on domain label '$start_clock_domain_label'"
     $violation dict_set end_callout_string side0 "Destination sequential
cell on domain label '$end_clock_domain_label'"
      lappend msg_list "The sequential cell '$mapped_end_name' which is
not a synchronizer cell is on clock domain label
'$end_clock_domain_label'."
      lappend msg_list "It has sequential cell '$mapped_start_name' on
clock domain label '$start_clock_domain_label' in its fanin"
      lappend msg_list "which is not a synchronizer cell."
      $violation dict_set message_string_list $msg_list
      add_drc_violation [namespace current]::$violation
    }
  }
}

##### XYZ_rule_set1_attribute_registrations.tcl #####
namespace eval XYZ_rule_set1 {

    ### Attribute registrations
    register_attribute -name xyz_function \
                       -object_type {port pin gate_pin} \
                       -value_type string \
                       -default ""

    set_attribute_options -name xyz_function \
                          -object_type {port pin} \
                          -preserve_boundary_in_flat_model on \
                          -display_in_gui on

    register_attribute -name xyz_exclude \
        -object_type gate_pin \
        -value_type boolean

}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 6-8. Forward Walk to Find All Destination Hits

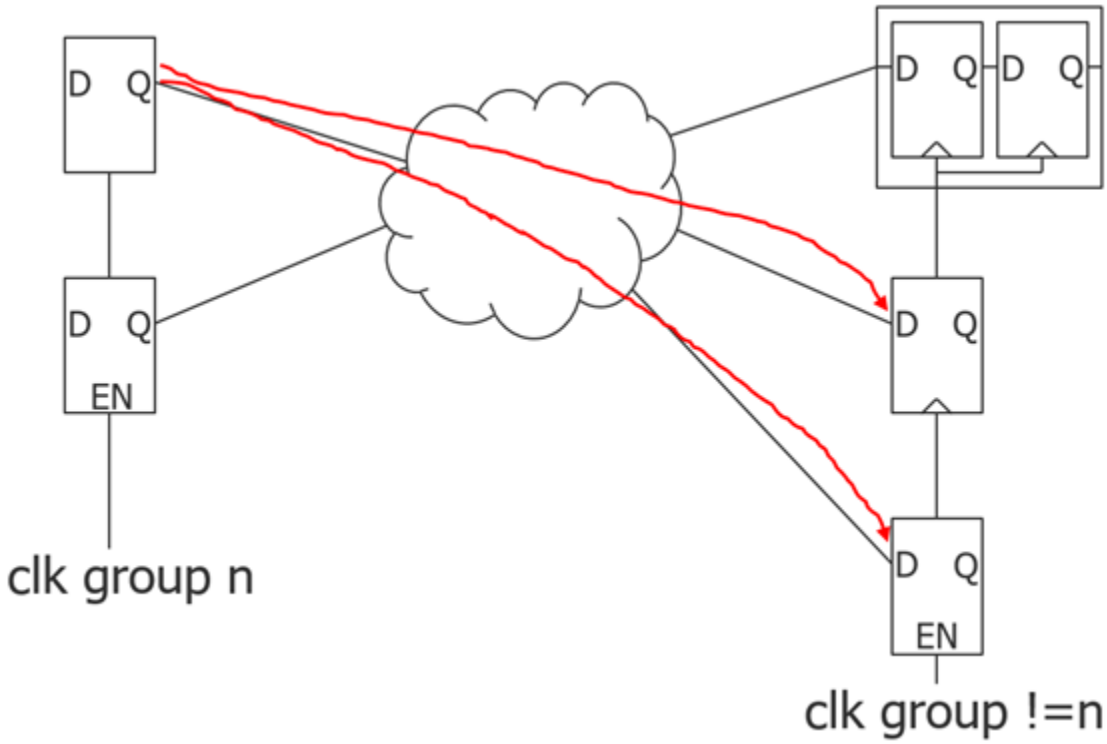
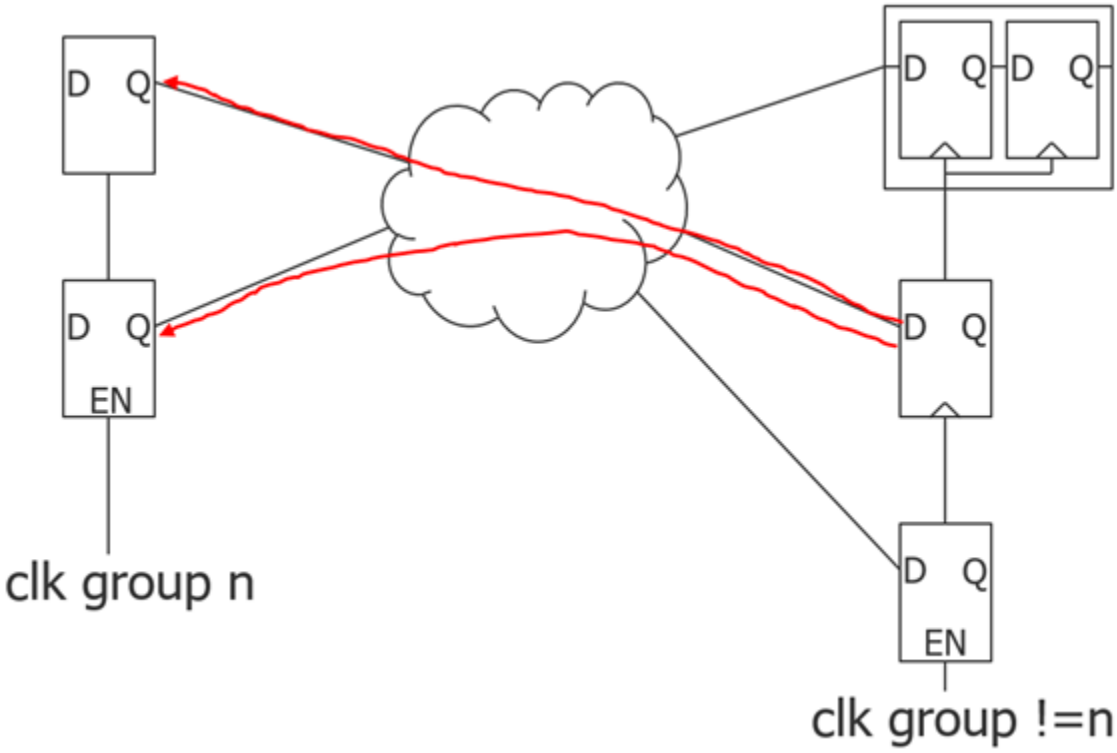


Figure 6-9. Backward Walk to Final All Source Hits



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
##### XYZ_rule_set1_callback_registrations.tcl #####
namespace eval XYZ_rule_set1 {

    register_callback elaboration_creation.post  \
                    [create_wrapper_proc clear_xyz_s_drc_options]
    register_callback system_mode_transition.pre  \
                    [create_wrapper_proc check_xyz_s3_requirements]

}

##### xyz_s_callbacks.tcl #####
# elaboration_creation.post
proc clear_xyz_s_drc_options {args} {
  variable xyz_s_drc_dict
  variable  xyz_s_drc_reset_dict
  dict set xyz_s_drc_dict drc_indexes 1 excluded_ports {}
  dict set xyz_s_drc_dict drc_indexes 3 excluded_clocks {}
  dict set xyz_s_drc_dict synchronizer_cells {}
}
# system_mode_transition.pre
proc check_xyz_s3_requirements {from to args} {
  variable xyz_s_drc_dict
  set error_status 0
  if {$from eq "setup" && $to eq "analysis" && [get_context] eq "dft" && \
      [get_context -no_sub_context] && \
      [dict get $xyz_s_drc_dict drc_indexes 3 enable]} {
    if {[sizeof_collection \
          [dict get $xyz_s_drc_dict synchronizer_cells]] == 0} {
      display_message -error \
        "You can't enable XYZ_S3 without having specified the -
synchronizer_cells switch."
      set error_status 1
    }
    if {[get_dft_specification_requirement -logic_test] eq "off"} {
     set    msg "You can't enable XYZ_S3 without also enabling logic_test
rules using the "
      append msg "\n'set_dft_specification_requirements -logic_test on'
command because it depends on "
      append msg "\nthe 'clock_domain_label' attribute set by DFT_C6."
      display_message -error $msg
      set error_status 1
    }
    if {$error_status} {
      return -code error
    }
  }
}
```

Example 4

This example implements a DRC rule that verifies that output ports or input pins with a given xyz_function attribute value are controlled by input ports or output pins with the same xyz_function attribute value. This DRC could be used to verify that, for example, special pins of pad buffers are connected to the right programing logic. The kind of violation detected by this DRC is illustrated in Figure 6-10. The names shown on the pins or ports are not their names but the value of the xyz_function attribute. The V1 violation detects an input port with xyz_function

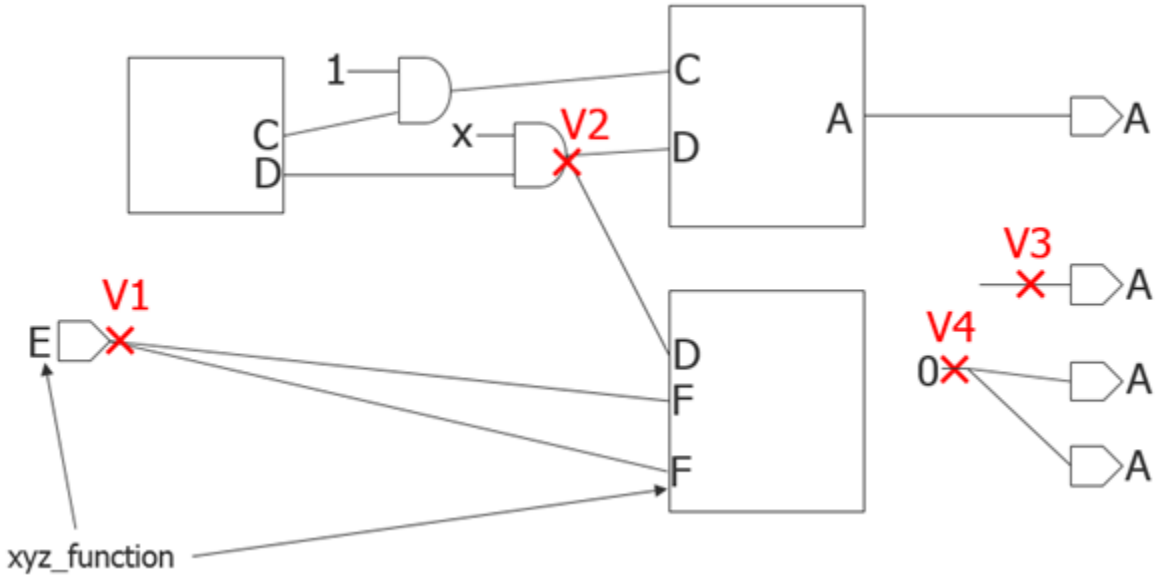
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

equal to E driving two pins with xyz_function equal to F. The V2 violation detects two pins with xyz_function equal to D driven by a gate that blocks the controlling path from the valid source. The V3 violation is an output port with no source. Finally, the V4 violation is two output ports driven by a common source which is a constant value. The DRC is implemented such that it only reports one violation per invalid source and not one violation per incorrect destinations. You see this is achieved by tracing backward from the output port and the input pins with an xyz_function value. When the controlling walk stops, a forward walk is used to collect all output ports and input pins with the same xyz_function value in the controlling fanout of the stop point. This avoids tracing backward from all incorrectly connected destination sharing a common source. This make sure that V1 and V2 in Figure 6-10 are issued once instead of twice. For the V4 violation, because the first port to trace evaluate to a constant, a constant source tracing is done to find the source of the constant, the constant source is then forced to X and a forward controlling walk is performed to collect all destinations sharing the same invalid constant source.

Note

This DRC relies on having the user-defined attribute called xyz_function set on pins of instances and ports of the current design. Refer to Example 5 below for a description on how you can use configuration files to automate the process of setting the attribute.

Figure 6-10. Violations Detected by XYZ_S4_DRC



The xyz_attribute is registered in the file *XYZ_rule_set1_attribute_registrations.tcl* and its content was shown in Example 3. Again the trace_flat_model command is used. This time the controllability switch is set to controlling and to constant when tracing a pin that has a constant value.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
##### xyz_s4.tcl #####
proc XYZ_S4_drc {mode drc_name} {
  variable xyz_s_drc_dict
  if {$mode eq "enabled"} {
    return [dict get $xyz_s_drc_dict drc_indexes 4 enable]
  }
  # Stable_after_setup resets the ijtag network and include any
  # set_test_setup_icall that maybe needed to power-up the device
  # This disables the DFT logic that may exists along the paths.
  set_current_simulation_context stable_after_setup

  #Transfer the xyz_function from hierarchical model to flat model
  #Used storage instead of inheritance because of performance and
  #relatively low volume of data.

  reset_attribute_value -object_type gate_pin -name xyz_function
  #Using get_attributed_objects is instantaneous as compared to
  #[get_pins -filter xyz_function]
  foreach_in_collection attributed_object \
       [get_attributed_objects -attribute_name xyz_function \
                               -object_types {port pin}] {
    set current_value [lindex [get_attribute_value_list \
                         $attributed_object -name xyz_function] 0]
    set gate_pin [get_gate_pins -from_objects $attributed_object]
    set_attribute_value $gate_pin -name xyz_function \
                                  -value $current_value -silent
  }

  set gate_pins_with_xyz_function [get_attributed_objects \
                   -attribute_name xyz_function -object_types gate_pin]
  set destinations [filter_collection $gate_pins_with_xyz_function \
                                      "direction == input"]
  while {[sizeof_collection $destinations] > 0} {
    set start_point [index_collection $destinations 0]
    set expected_value [lindex [get_attribute_value_list $start_point \
                                          -name xyz_function] 0]
    set simulation_value [lindex [get_attribute_value_list $start_point \
                                          -name simulation_value] 0]
    if {$simulation_value in {0 1}} {
      #Find source of constant to report it
      set stop_point [trace_flat_model \
                         -from $start_point \
                         -direction backward \
                         -controllability constant \
                         -store_path_to_stop_point path_to_stop_point \
                         -map_tag_to_design_module_boundary on]

      XYZ_S4_give_constant_violation $simulation_value $start_point \
                     $stop_point $path_to_stop_point $expected_value
      XYZ_S4_remove_all_destinations_with_that_common_source \
                     $simulation_value $start_point $stop_point \
                     $expected_value
    } else {
      set stop_point [trace_flat_model \
                         -from $start_point \
                         -direction backward \
                         -stop_condition "xyz_function" \
                         -controllability controlling \
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
                              -store_path_to_stop_point path_to_stop_point \
                              -map_tag_to_design_module_boundary on]

        if {[sizeof_collection $stop_point] == 0} {
          XYZ_S4_give_no_source_violation $start_point $expected_value
          set destinations [remove_from_collection $destinations \
                                                   $start_point]
        } else {
          set current_value [lindex [get_attribute_value_list $stop_point \
                                     -name xyz_function] 0]
          if {$current_value ne $expected_value} {
            XYZ_S4_give_incorrect_source_violation $start_point \
                         $stop_point $path_to_stop_point $current_value \
                         $expected_value
          }
          XYZ_S4_remove_all_destinations_with_that_common_source \
                         $simulation_value $start_point $stop_point \
                         $expected_value
        }
      }
    }
  }
  ####################################################################
  proc XYZ_S4_give_no_source_violation {start_point expected_value} {
    set violation [DrcViolation #auto]
    set start_object_type [get_attribute_value_list [get_design_objects \
                          -from_objects $start_point] -name object_type]
    set start_object_name [get_single_name $start_point]
    $violation dict_set simulation_context [get_current_simulation_context]
    $violation dict_set list_of_paths side0 [list $start_point]
    $violation dict_set path_direction side0 backward
    $violation dict_set start_callout_string side0 \
          "Object with xyz_function=$expected_value with no source"
    lappend msg_list "The $start_object_type '$start_object_name' with
  xyz_function '$expected_value' has no source."
    $violation dict_set message_string_list $msg_list
    add_drc_violation [namespace current]::$violation
  }
  ####################################################################
  proc XYZ_S4_give_constant_violation {simulation_value start_point
  stop_point path_to_stop_point expected_value} {
    set violation [DrcViolation #auto]
    set start_object_type [get_attribute_value_list [get_design_objects \
                            -from_objects $start_point] -name object_type]
    set start_object_name [get_single_name $start_point]
    set stop_object_type [get_attribute_value_list [get_design_objects \
                            -from_objects $stop_point] -name object_type]
    set stop_object_name [get_single_name $stop_point]
    $violation dict_set simulation_context [get_current_simulation_context]
    $violation dict_set list_of_paths side0 [list $path_to_stop_point]
    $violation dict_set path_direction side0 backward
    $violation dict_set start_callout_string side0 \
                  "Start point with xyz_function=$expected_value"
    $violation dict_set end_callout_string side0 "Source of constant value"
    lappend msg_list "The $start_object_type '$start_object_name' with
  xyz_function '$expected_value' is source by a constant $simulation_value."
    lappend msg_list "The source of the constant value is $stop_object_type
  '$stop_object_name'."
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
  $violation dict_set message_string_list $msg_list
  add_drc_violation [namespace current]::$violation
}
#################################################################
proc XYZ_S4_give_incorrect_source_violation {start_point stop_point
path_to_stop_point current_value expected_value} {
  set start_port [index_collection $path_to_stop_point 0]
  set violation [DrcViolation #auto]
  set start_object_type [get_attribute_value_list [get_design_objects \
                         -from_objects $start_point] -name object_type]
  set start_object_name [get_single_name $start_point]
  set stop_object_type [get_attribute_value_list [get_design_objects \
                         -from_objects $stop_point] -name object_type]
  set stop_object_name [get_single_name $stop_point]
  $violation dict_set simulation_context [get_current_simulation_context]
  $violation dict_set list_of_paths side0 [list $path_to_stop_point]
  $violation dict_set path_direction side0 backward
  $violation dict_set start_callout_string side0 \
               "Start point with xyz_function=$expected_value"
  if {$current_value eq ""} {
    lappend msg_list "The $start_object_type '$start_object_name' with
xyz_function '$expected_value' backward tracing stopped at"
    lappend msg_list "$stop_object_type '$stop_object_name' and this
$stop_object_type does not have an xyz_function value."
    $violation dict_set end_callout_string side0 \
                        "Source with no xyz_function"
  } else {
    lappend msg_list "The $start_object_type '$start_object_name' with
xyz_function '$expected_value' is sourced by"
    lappend msg_list "$stop_object_type '$stop_object_name' but its
xyz_function value is '$current_value'."
    $violation dict_set end_callout_string side0 "Source with incorrect
xyz_function"
  }
  $violation dict_set message_string_list $msg_list
  add_drc_violation [namespace current]::$violation
}
#################################################################
proc XYZ_S4_remove_all_destinations_with_that_common_source
{simulation_value start_point stop_point expected_value} {
  upvar destinations destinations
  #This proc finds all the destination with the same xyz_funtion value
  #in the controlling fanout of the source to only give one violation
  #per bad source. It is also much more efficient when tracing a high
  #fanout function.
  set current_simulation_context [get_current_simulation_context]
  if {$simulation_value in {0 1}} {
    #If the stop point is constant, clone the simulation condition and\
    #force it to X to enable doing a controlling fanout trace.
    if {"xyz_delta" ni [get_simulation_context_list -user_defined]} {
      add_simulation_context xyz_delta -silent
    } else {
      set_current_simulation_context xyz_delta
    }
    copy_simulation_context -from $current_simulation_context
    add_simulation_forces $stop_point -values X
    simulate_forces
  }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
    set hits [trace_flat_model \
                        -from $stop_point \
                        -direction forward \
                        -controllability controlling \
                        -stop_condition "xyz_function" \
                        -tag_condition "xyz_function==$expected_value"]
    # In case the constant destination is not in controlling fanout when
    # source is set to X
    set destinations [remove_from_collection $destinations \
                          [add_to_collection $hits $start_point]]
    if {[get_current_simulation_context] ne $current_simulation_context} {
      set_current_simulation_context $current_simulation_context
      # clean up memory
      delete_simulation_context xyz_delta
    }
  }
```

Example 5

This example does not show a new DRC example, but instead a method to populate the xyz_function attribute needed for the XYZ_S4 DRC rule shown in the previous example. Configuration data is a very convenient way of defining attributes on ports of a module. With a simple callback shown below, you can easily transfer the attribute values from the configuration file to the ports of the current design and the pins of the instances of the modules.

Below is an example configuration file describing attributes on ports of modules. Even though the example has multiple Core wrappers in the same file, you can choose to use one file per Core wrapper or merge them in a single file.

```
##### modules.tcd_xyz #####
Core(epoch_counter) {
  Xyz {
    Port(tic_enable) {
      function : tic_enable;
    }
  }
}
Core(code_gen) {
  Xyz {
    Port(tic_enable) {
      function : tic_enable;
    }
  }
}
Core(time_base) {
  Xyz {
    Port(tic_enable) {
      function : tic_enable;
    }
    Port(rstn) {
      function : reset;
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Core(gps_baseband) {
  Xyz {
    Port(hw_rstn) {
      function : reset;
    }
  }
}
```

Below is the metadata used to define the legal syntax of the XYZ wrapper. See the "Metadata Configuration Syntax" chapter for more information about the metadata syntax. The location of the metadata file within the plugin was shown in Figure 6-4 on page 1601.

```
##### XYZ_rule_set1.tessent_meta #####

Partitions: default, tcd;
Wrapper(/Core) {
  Wrapper(Xyz) {
    Repeatable: off;
    Property(version) {
      Value(version) {
        Type: int;
        DefaultValue : 1;
      }
    }
    Wrapper(Port) {
      Id(port_name) {
        Type: string;
      }
      Property(function) {
        Value(enum) {
          LegalString : "",tic_enable, pre_tic_enable, reset;
          DefaultValue : "";
        }
      }
    }
  }
}
```

You use the read_core_descriptions command to load the *.tcd_xyz* files. Below is a callback procedure that you can use in your plugin to automatically transfer the xyz_function value to the ports of the current design and pins of instances of the sub modules. Notice the use of the -use_module_matching_options switch in the get_modules command usage. This ensures that if your module names were uniquified during synthesis that the modules are still matched to the RTL module names found in the Core wrapper. See the set_module_matching_options command for more information about module matching.

```
# Callback registration
namespace eval XYZ_rule_set1 {
  module_matching.post [create_wrapper_proc apply_xyz_function]
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# Body of callback

proc apply_xyz_function {args} {
  reset_attribute_value -object_types {port pin} -name xyz_function
  foreach_in_collection tcd_xyz_wrapper [get_config_elements core/xyz \
                                          -partition tcd -silent] {
    set core_wrapper [get_config_value $tcd_xyz_wrapper -parent_object]
    set core_name [get_config_value $core_wrapper -id <0>]
    set matched_modules [get_module [list $core_name] \
                              -use_module_matching_options -silent]
    if {[sizeof_collection $matched_modules] > 0} {
      if {[compare_collections $matched_modules \
                              [get_current_design]] == 0} {
        set is_current_design 1
      } else {
        set is_current_design 0
        set matched_instances [get_instances \
                                 -of_modules $matched_modules -silent]
      }
      if {$is_current_design || \
          [sizeof_collection $matched_instances] > 0} {
        foreach_in_collection port_wrapper [get_config_elements Port \
                           -below_wrappers $tcd_xyz_wrapper -silent] {
          set port_name [get_config_value $port_wrapper -id <0>]
          set function_name [get_config_value function \
                                        -in_wrapper $port_wrapper]
          if {$is_current_design} {
            set objects [get_ports [list $port_name]]
          } else {
            set objects [get_pins [list $port_name] \
                              -of_instances $matched_instances]
          }
          set_attribute_value $objects -name xyz_function \
                                     -value $function_name
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

register_drc_class

Context: dft. patterns, unspecified

Mode:setup

Registers a new class of DRC rules such that a DRC of that class can be registered

Usage

register_drc_class ***class_name*** [-runs_after *class_name*]

Description

A command used to register a new class of DRCs. Once you register a class, you use the "register_drc *drc_name* -class *class*" command to register DRC rules of that class. If you are registering your own DRC rules, it is best to use your own class name containing something specific to your company in the name such that you will never collide with future built-in DRC rules. In the example shown in the register_drc command section, the class name "XYZ_S" was used. The "XYZ" part is meant to represent your company and the "S" represent the further classification you may want to use.

Arguments

- ***class_name***

  A required string that must start with a capital letter and is followed by only capital letters, numbers, and underscores.

- -runs_after *class_name*

  An optional switch-value pair that specifies the order in which the current DRC class is to run relative to the other custom-registered DRC classes. When unspecified, the class will run after the class that it follows in a numerical sorted list of the custom-registered class names.

  When you register a DRC using the register_drc command, you can control precisely in which order the DRC runs. It is not a requirement that all DRC rules of a class run before the DRC rules of another class. You can intermix them as you needed. The class run order only affects registered DRC rules that do not use the -runs_after switch.

Examples

The following example registers a new DRC class called XYZ_T and specifies that is runs before the DRC class XYZ_D. If the -runs_after switch was not specified, the order would be reversed because XYZ_T is after the XYZ_D in a numerical sort.

```
register_drc_class XYZ_T -runs_after XYZ_D
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

register_static_dft_signal_names

Context: dft, patterns

Mode: setup, analysis, insertion

Registers new static DFT signal names and their attributes.

Usage

register_static_dft_signal_names {***name*** ...}
[-usage {global_dft_control | logic_test_control | scan_mode}]
[-scan_mode_type {unwrapped | internal | external | retargeting}]
[-default_value_in_all_test {reset_value | 0 | 1}]
[-reset_value {0 | 1}]
[-value_in_pre_scan_drc {x | 0 | 1}]

Description

Registers new static DFT signal names and their attributes. The tool has built-in pre-registered DFT signal names that you can view using the report_dft_signal_names command.

You use the register_static_dft_signal_names command to create a new dft_signal and specify its behavior (for example, reset_value, default_value_in_all_test).

You only use this command if the pre-defined dft_signals do not fit your needs. For example, if you have logic you want controlled in different test modes and none of the pre-defined dft_signals work for your logic, then you can register a new dft_signal.

After you have registered your dft_signal, you use the add_dft_signals command to create the hardware needed to implement your new dft_signal. You are now able to use your new dft_signal to control logic by referencing your new dft_signal with the "dft_signal_source_name" option of the add_dft_control_points or add_dft_clock_mux commands.

You can also register new scan_mode signals using this command if you find that you do not have the one you need in the pre-registered list. It is important that you set the proper -scan_mode_type value when registering a DFT signal of usage scan_mode. See the description of the set_static_dft_signal_values command to understand how the scan_mode_type affects the implied value of the int_ltest_en and ext_ltest_en signals.

Arguments

- ***name***

  A required repeatable string that specifies the name of the DFT signals to register. The string is also permitted to be a well formatted Tcl list of names. The names must start with a letter and be followed by only letters, numbers, or underscores.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -usage {global_dft_control | logic_test_control | scan_mode}

  An optional switch and literal pair that specifies the usage of the newly-registered DFT signal names. When the -usage switch is not specified, the default is global_dft_control.

  Choose one from the following:

  global_dft_control — This usage is for signals controlling global resources like clocking and power management signals. Those signals are automatically set to their specified "default_value_in_all_test" values unless the value is explicitly requested to be differently set using either the DftControlSetting wrapper of the Patterns wrapper or using the set_static_dft_signal_values command. See Table 3-12 on page 210 in the add_dft_signals command description of the pre-registered global DFT controls.

  logic_test_control — This usage is for signals used to control aspects of the logic test modes. Unlike the signal with usage scan_mode, the signals are not mutually exclusive to each other and can be set high or low when using the same scan configuration. See Table 3-12 on page 210 in the add_dft_signals command description of the pre-registered logic test controls.

  scan_mode — This usage is for signals used to control the scan chain configurations within a module. All scan_mode signals are mutually exclusive and only one of them per instance can be set to 1.

- -scan_mode_type {unwrapped | internal | external | retargeting}

  A switch and literal pair that specifies the type of scan_mode signal. You must ensure you set the proper -scan_mode_type value when registering a DFT signal of usage scan_mode. See the description of the set_static_dft_signal_values command to understand how the scan_mode_type affects the implied value of the int_ltest_en and ext_ltest_en signals.

  The -scan_mode_type switch is mandatory when the -usage switch is scan_mode

  Choose one from the following:

  unwrapped — Used for scan_mode signal enabling scan configuration having no wrapper isolation. When a scan_mode signal having the type unwrapped is set to 1 using the set_static_dft_signal_values command, the int_ltest_en and ext_ltest_en signal that may exist in the same instance are automatically set to 0.

  internal — Used for scan_mode signal enabling scan configuration using the inward isolation of the wrapper chains. When a scan_mode signal having the type internal is set to 1 using the set_static_dft_signal_values command, the int_ltest_en and ext_ltest_en signal that typically exist in the same instance are automatically set to 1 and 0, respectively.

  external — Used for scan_mode signal enabling scan configuration using the outward isolation of the wrapper chains. When a scan_mode signal having the type external is set to 1 using the set_static_dft_signal_values command, the int_ltest_en and ext_ltest_en signal that typically exist in the same instance are automatically set to 0 and 1, respectively.

  retargeting — Used for scan_mode signal enabling of channel access to combinations of lower-level blocks. The lower-level blocks typically have one of their internal scan_mode signals activated.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -default_value_in_all_test {reset_value | 0 | 1}

  An optional switch and literal pair that specifies the default value the signals with usage global_dft_control should have in all tests. The default value of reset_value means that the value specified for the -reset_value switch is also used as the default value in all tests.

- -reset_value {0 | 1}

  An optional switch and literal pair that specifies the reset value for the corresponding global_dft_control signal. This value corresponds to the value you want this signal to have in functional mode. It is illegal to specify -reset_value 1 when the -usage switch is not global_dft_control. If you want to register a DFT signal with a reset value of 1, then you must register the signal as global_dft_control.

- -value_in_pre_scan_drc {x | 0 | 1}

  An optional switch and literal pair that specifies the value the logic_test_signal should be set to when running the Pre-DFT or Pre-Scan DRC rules. You must register the logic_test_signal for logic_test_control usage (for example, -usage logic_test_control), or Tessent Shell will report an error. The specified value, with the Xs converted to 0s, also serves as the default value the given logic test control is set to when using the set_static_dft_signal_values command. See the set_static_dft_signal_values section for a description of where you use this value.

Examples

Example 1

The following example registers a new global DFT control signal to control the select of a clock multiplexer. By default, the select is driven high in all DFT test modes but the select can be set to 0 using the DftControlSettings wrapper in the Patterns wrapper or using the set_static_dft_signal_values command to configure the test_setup of a logic test mode.

```
register_static_dft_signal_name clockb_select -reset_value 0 -default_value_in_all_test 1

add_dft_control_points clock_mux/s -dft_signal_source_name clockb_select
```

Example 2

The following example illustrates how you can create a plugin to augment the *.tcd_scan* syntax with a new logic_test_control DFT signal. The DFT signal is automatically registered. If included inside a *.tcd_scan* file, the DFT signal is automatically understood during sub-chain tracing in context dft -scan (see set_context). Also, if the port is seen to be tied off on the instance, the scan insertion tool automatically attaches the port to the source of the DFT signal as returned by the get_dft_signal command.

Below is the content of file *path/my_plugin/metadata/tcd_scan.tessent_meta*. As you can see, it defines a new logic_test_control called my_sig1. The tracing_condition property defines the value that must be forced on the signal when tracing the sub-chain. This value becomes the "value_in_pre_scan_drc" value of the registered DFT signal. You must use a different usr[#] index for each signal you registered. In the following example, you can only modify the items in bold. The rest of the entries must appear exactly as shown.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Partitions: default,tcd;

Wrapper (/Core/Scan) {
  Wrapper (MySig1) {
    Options(PreDftDrcs) {
      dft_control_type  : scan_control;
      dft_scan_signal   : usr[0]:0;
      dft_scan_function : my_sig1; //Any name
      tracing_condition : active; //active or inactive
      reset_state       : inactive;//active or inactive
    }
    Id(port_name*) {
      type : string;
    }
    // use active_polarity_default_all_zeros when the default
    // active polarity of the signal is 0.
    reference =
/tcd_scan/active_polarity_default_all_ones/active_polarity;
  }
}
```

To have the metadata automatically loaded by the tool when you invoke Tessent Shell, set the TESSENT_PLUGIN_PATH environment variable to *path/my_plugin*. For more information about using plugins, refer to "Adding Custom Plugin Functionality" on page 4981.

Related Topics

report_static_dft_signal_settings

unregister_static_dft_signal_names

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

register_tcl_command

Context: unspecified, all contexts

Mode: all modes

Registers a Tcl command and makes it available in the global name space.

Usage

register_tcl_command ***command_name***
-syntax_bnf *syntax_bnf*
-tcl_proc *tcl_proc*
-context_list *context_list*
-system_mode_list *system_mode_list*
[-resource_list *resource_list*]
[-extra_help *extra_help*]
[-integer_list_option *integer_list_option*]
[-string_list_option *string_list_option*]
[-real_list_option *real_list_option*]
[-integer_list_value *integer_list_value*]
[-string_list_value *string_list_value*]
[-real_list_value *real_list_value*]
[-boolean_option *boolean_option*]
[-integer_option *integer_option*]
[-string_option *string_option*]
[-real_option *real_option*]
[-integer_value i*nteger_value*]
[-string_value *string_value*]
[-real_value *real_value*]
[-transcript_style {tool_command | normal}]

Description

Registers a Tcl command and makes it available in the global name space.

The registered command displays as a normal tool command with proper option parsing and online help messages. The body of the registered command is implemented in a Tcl proc referenced by the -tcl_proc option. After the options are validated, they are packaged into a Tcl list of option-value pairs that are packed into a Tcl array for easy random access as shown in Example 1.

You control the availability of the newly registered command using the -context_list and -system_mode_list options.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Commands in Tessent Shell are not case-sensitive. Therefore, if you register a Tcl command that contains upper-case letters, the tool does not preserve the case of that command, and they are displayed as lower case in the online help.

When registering a Tcl command with embedded commands, you use a double hyphen (--) to specify the end of the command options passed to the embedded command. Passing an argument value to the registered Tcl command that starts with a hyphen (-) can result in an error when the value is ultimately used as an argument to an embedded command in the proc that is registered.

The following Tessent Shell transcript shows the behavior of the is_collection command in various situations that require the usage of double hyphens:

```
SETUP> set v -a
-a
# Here, 'is_collection' wrongly tries to interpret '-a' as one of its
# arguments
SETUP> is_collection $v
  // Error: Unrecognized argument '-a'. ( help is_collection )

# With the double hyphen, the value is recognized as NOT being a
# collection.
SETUP> is_collection -- $v
0
# Other arguments recognized by the command can be passed BEFORE the
# double hyphen
SETUP> is_collection -single_type_object -- $v
0
SETUP>
```

The above usage of the double hyphen (--) is demonstrated in the Example 1.

Tip

When you register a set command, especially when it can only be called in setup mode, use the add_tcl_variables_to_flat_model command to store the Tcl variables holding the setting values in the flat model. This ensures that the settings are restored when reading the flat model.

Arguments

- ***command_name***

  A required string that specifies the name of the command to register. Because the command exists in the global name space, its name must be unique so that it does not collide with other already registered commands. You can unregister any registered Tcl command using unregister_tcl_command until you run the lock_current_registration command. After you issue lock_current_registration, all registered commands and attributes are locked and can no longer be modified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

All commands loaded with an automated plugin mechanism are locked before the interactive shell is released to the user. "Example 2" on page 1648 explains how to initially debug your commands interactively when it is still possible to unregister them, and then shows you how to package them into a plugin directory which is automatically loaded at tool invocation. For more information about plugins, see the lock_current_registration command.

- **-syntax_bnf *syntax_bnf***

  A required switch and string value pair that specifies the grammar rules of your command using simple Backus-Naur Form (BNF). You should place the syntax_bnf string in braces to avoid needing to escape square brackets which are used to express that an option or value is optional. You should use parentheses to group options that are exclusive to others. You can use the | symbol to denote exclusivity of options. BNF uses the symbols you defined when declaring the *option and *value arguments; the symbols must always be enclosed in <> characters.

  The following example defines five arguments; the first two are required and the last three are optional. The fifth argument is mutually exclusive of the third and fourth arguments. An argument is repeatable if its tag is declared with an * as its last character.

  ```
  { <arg1> <arg2> ([arg3] [arg4]) | [arg5] }
  ```

- **-tcl_proc *tcl_proc***

  A required switch and string pair that specifies the name, including its namespace, of the Tcl proc this command calls. To facilitate packaging the new command as a plugin that is automatically loaded into local namespaces, you should specify tcl_proc using the following format: [namespace current]::proc_name. This format ensures that the registered Tcl command always knows where to look for the Tcl proc regardless of which namespace the command is evaluated in.

  For easy interactive debugging that can be easily deployed as a plugin, you should package both your Tcl proc and the register_tcl_command into a single file and surround it with a namespace eval name_space {} wrapper. Do not put :: in front of the specified name_space string so that this namespace string is automatically appended to the one automatically created for all plugins. See the examples for more information on how to do this.

- **-context_list *context_list***

  A required switch and string pair that specifies a Tcl list of contexts the command is enabled in. Each element of the Tcl list is a context with an optional sub-context option, as shown here:

  ```
  -context_list {{dft} {patterns -ijtag}}
  ```

  The available literals are: all, dft, patterns. The available sub-contexts are listed under "Contexts" in the *Tessent Shell User's Manual.*

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-system_mode_list *system_mode_list***

  A required switch and string pair that specifies a Tcl list of system modes in which you want to enable the command. The available literals are: all, setup, analysis, and insertion.

- -resource_list *resource_list*

  An optional switch and string pair that specifies the list of resources required for the command to be available; if any of the specified resources is not available, the command is not available. The resource_list argument is a Tcl list of strings that identify the resources required. Possible values are:

  design_is_elaborated — Used to check that the hierarchical data model has been created using the set_current_design command.

  design_modules_exist — Used to check that at least one Verilog or VHDL module exists in memory and can be accessed using the get_modules command.

  flat_model_exists —Used to check that the flat data model is available. The flat data model was created by either the create_flat_model command or by the "set_system_mode analysis" command.

  icl_is_elaborated —Used to check that the ICL hierarchical data model exists. It exists after the set_current_design command if an ICL module corresponding to the current design existed prior to issuing the command. It also exists in the analysis mode if the top-level ICL module was successfully extracted with ICL extract.

  icl_modules_exist —Used to check that at least one ICL module exists in memory and can be accessed using the get_icl_modules command.

  rtl_mode_enabled — Used to check that the -rtl option switch was used when specifying the context and that read-in Verilog and VHDL files can contain RTL constructs.

- -extra_help extra_help

  An optional switch and string pair that enables you to define a multi-line help paragraph that is displayed at the bottom of the online help. You can use \n to break your strings into multiple lines. The easiest way to create a multi-line help string is to construct it line by line as shown here:

```
set extra_help "This is line1\n"
append extra_help "and this is line 2\n"
append extra_help "and this the last line."
register_tcl_command -extra_help "$extra_help" ...
```

- -integer_list_option *integer_list_option*

  An optional switch and string pair that defines an argument that is an option-value pair where the value is a Tcl list of integers. The integer_list_option_list string is a Tcl list with three required elements and five optional elements as shown here:

```
{{<symbol>} {-option_name} {tag} [{description_string}] [{range}]
[{enum_list}] [{<help_symbol>}][{<visibility>}]}
```

  <symbol> — The first element of the list is required. It defines a unique symbol for the argument that you use in the -syntax_bnf syntax_bnf string to refer to it and it also

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

displays in the online help description. The symbol string must always begin with a < character and end with a > character.

-option_name — The second element of the list is required. It defines a unique option name. The option name must start with a - character. The online help displays this option as shown here: “-option_name <symbol>”.

tag — The third element of the list is required. It defines a unique tag that is used to identify the argument value when invoking the associated Tcl proc. You can use an * character as the last character of the tag string to denote that it is repeatable. If the argument is not repeatable (specifically the tag does not end with an *), the Tcl list passed to the Tcl proc includes the tag string followed by the value specified for it. If the argument is repeatable, the tag is appended with “,#” where # is an integer incremented from 0 to uniquely identify each instance of the argument. For example, if the tag is specified as range* and the argument is specified three times, the Tcl list passed to the Tcl proc includes those six elements as shown here:

```
{{range,0}{value1}{range,1}{value2}{range,2}{value3}}
```

description_string — The forth element is optional. It can be used to define a one-line description for the argument that displays in the online help. If you need to specify the range or enum_list element but do not want to specify a description string, specify this fourth element as empty by using an empty brace pair.

range — The fifth element is optional. It consists of one or two integers specifying the minimum and maximum value for the argument. When the range is not specified or is specified as empty, the smallest permitted value is a MIN_INT. When the second integer is not specified, the largest permitted value is MAX_INT. If you need to specify the enum_list element but do not want to limit the range, specify this fifth element as empty using an empty brace pair.

enum_list — The sixth element is optional. It enables you to specify strings as permitted values. For example, you may want to define an option that permits a positive integer or the literal “last”. This is achieved by using a minimum value of 0 and an enum string called “last”.

<help_symbol> — The seventh element is optional. It enables you to specify a different symbol for the option to display in the help message. By default the <symbol> specified as the first element of the list is used in the help message. However, the <symbol> must be unique across all options as it is referenced in the syntax_bnf string. For example, the help message of the get_instances command shows “[-parent_of_instances <instance_objects> | -below_instances <instance_objects>]”. The symbol for each option has to be different such as <inst_for_parent_of> and <inst_for_below> but the display symbol in the help message can be made to be <instance_objects> for both by specifying the eight element.

visibility — The eighth element is optional. It enables you to specify if you want the option to be hidden from the help message. You use a null string if you want the option to show up in the help message and you use the string “hidden” if you want the option to be hidden.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -string_list_option *string_list_option*

  An optional switch and string pair that defines an argument that is an option-value pair where the value is a Tcl list of strings. The string_list_option is a Tcl list with three mandatory elements and four optional elements as shown here:

  ```
  {{<symbol>} {-option_name} {tag} [{description_string}]
  [{enum_list}] [{<help_symbol>}] [{<visibility>}]}
  ```

  <symbol> — The first element of the list is required. It defines a unique symbol for the argument that you use in the -syntax_bnf syntax_bnf string to refer to it and it also displays in the online help description. The symbol string must always begin with a < character and end with a > character.

  -option_name — The second element of the list is required. It defines a unique option name. The option name must start with a - character. The online help displays this option as shown here: "-option_name <symbol>".

  tag — The third element of the list is required. It defines a unique tag that is used to identify the argument value when invoking the associated Tcl proc. You can use an * character as the last character of the tag string to denote that it is repeatable. If the argument is not repeatable (specifically the tag does not end with an *), the Tcl list passed to the Tcl proc includes the tag string followed by the value specified for it. If the argument is repeatable, the tag is appended with ",#" where # is an integer incremented from 0 to uniquely identify each instance of the argument. For example, if the tag is specified as range* and the argument is specified three times, the Tcl list passed to the Tcl proc includes those six elements as shown here:

  ```
  {{name,0}{value1}{name,1}{value2}{name,2}{value3}}
  ```

  description_string — The forth element is optional. It can be used to define a one-line description for the argument that displays in the online help. If you need to specify the range or enum_list element but do not want to specify a description string, specify this fourth element as empty by using an empty brace pair.

  enum_list — The fifth element is optional. It enables you to specify strings as permitted values. When omitted or specified as an null string, the string_list_option permits any string values.

  <help_symbol> — The sixth element is optional. It enables you to specify a different symbol for the option to display in the help message. By default the <symbol> specified as the first element of the list is used in the help message. However, the <symbol> must be unique across all options as it is referenced in the syntax_bnf string. For example, the help message of the get_instances command shows "[-parent_of_instances <instance_objects> | -below_instances <instance_objects>]". The symbol for each option has to be different such as <inst_for_parent_of> and <inst_for_below> but the display symbol in the help message can be made to be <instance_objects> for both by specifying the eight element.

  visibility — The seventh element is optional. It enables you to specify if you want the option to be hidden from the help message. You use a null string if you want the option to show up in the help message and you use the string "hidden" if you want the option to be hidden.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -real_list_option *real_list_option*

  An optional switch and string pair that defines an argument that is an option-value pair where the value is a Tcl list of real numbers. The real_list_option is a Tcl list with three required elements and five optional elements as shown here:

  ```
  {{<symbol>} {-option_name} {tag} [{description_string}] [{range}]
  [{enum_list}] [{<help_symbol>}] [{<visibility>}]}
  ```

  <symbol> — The first element of the list is required. It defines a unique symbol for the argument that you use in the -syntax_bnf syntax_bnf string to refer to it and it also displays in the online help description. The symbol string must always begin with a < character and end with a > character.

  -option_name — The second element of the list is required. It defines a unique option name. The option name must start with a - character. The online help displays this option as shown here: "-option_name <symbol>".

  tag — The third element of the list is required. It defines a unique tag that is used to identify the argument value when invoking the associated Tcl proc. You can use an * character as the last character of the tag string to denote that it is repeatable. If the argument is not repeatable (specifically the tag does not end with an *), the Tcl list passed to the Tcl proc includes the tag string followed by the value specified for it. If the argument is repeatable, the tag is appended with ",#" where # is an integer incremented from 0 to uniquely identify each instance of the argument. For example, if the tag is specified as range* and the argument is specified three times, the Tcl list passed to the Tcl proc includes those six elements as shown here:

  ```
  {{range,0}{value1}{range,1}{value2}{range,2}{value3}}
  ```

  description_string — The forth element is optional. It can be used to define a one-line description for the argument that displays in the online help. If you need to specify the range or enum_list element but do not want to specify a description string, specify this fourth element as empty by using an empty brace pair.

  range — The fifth element is optional. It consists of one or two integers specifying the minimum and maximum value for the argument. When the range is not specified or is specified as empty, the smallest permitted value is a MIN_DOUBLE. When the second integer is not specified, the largest permitted value is MAX_DOUBLE. If you need to specify the enum_list element but do not want to limit the range, specify this fifth element as empty using an empty brace pair.

  enum_list — The sixth element is optional. It enables you to specify strings as permitted values. For example, you may want to define an option that permits a positive integer or the literal "last". This is achieved by using a minimum value of 0 and an enum string called "last".

  <help_symbol> — The seventh element is optional. It enables you to specify a different symbol for the option to display in the help message. By default the <symbol> specified as the first element of the list is used in the help message. However, the <symbol> must be unique across all options as it is referenced in the syntax_bnf string. For example, the help message of the get_instances command shows "[-parent_of_instances <instance_objects> | -below_instances <instance_objects>]".

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The symbol for each option has to be different such as <inst_for_parent_of> and <inst_for_below> but the display symbol in the help message can be made to be <instance_objects> for both by specifying the eight element.

visibility — The eighth element is optional. It enables you to specify if you want the option to be hidden from the help message. You use a null string if you want the option to show up in the help message and you use the string "hidden" if you want the option to be hidden.

- -boolean_option *boolean_option*

  An optional switch and string pair that defines an argument that is an option without a value. The boolean_option string is a Tcl list with three required elements and two optional elements as shown here:

  ```
  {{<symbol>} {-option_name} {tag} [{description_string}]
  [{<visibility>}]}
  ```

  <symbol> — The first element of the list is required and defines a unique symbol for the argument. You use this element in the -syntax_bnf syntax_bnf string to refer to it. The symbol string must always begin with a < character and end with a > character.

  -option_name — The second element of the list is required and defines a unique option name. The option name must start with a - character. The online help displays this option as shown here: "-option_name".

  tag — The third element of the list is required and defines a unique tag that is used to identify the argument value when invoking the associated Tcl proc.

  description_string — The forth element is optional and can be used to define a one-line description for the argument that displays in the online help.

  visibility — The fifth element is optional and can be used to specify whether the option is included in the help message. You specify a null string "" if you want the option to show up in the help message and you specify the string "hidden" if you want the option to be hidden.

- -integer_list_value *integer_list_value*

  An optional switch and string pair that is identical to the -integer_list_option switch and string pair except that it defines a value as opposed to an option-value pair.

  The integer_list_value string is a Tcl list with two required elements and five optional elements. This string has exactly the same definition as the integer_list_option string except that the second element does not exist because there is no option name to define. The tag is not permitted to end with an * character because only options can be defined as repeatable.

  {{<*symbol*>} {tag} [{*description_string*}] [{range}] [{*enum_list*}] [{<help_symbol>}] [{<visibility>}]}

- -string_list_value *string_list_value*

  An optional switch and string pair that is identical to the -string_list_option switch and string pair except that it defines a value as opposed to an option-value pair.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The string_list_value string is a Tcl list with two required elements and four optional elements. This string has exactly the same meaning as the string_list_option string except that the second element does not exist because there is no option name to define. The tag is not permitted to end with an * character as only options can be defined as repeatable.

{{<*symbol*>} {tag} [{*description_string*}] [{*enum_list*}] [{<help_symbol>}] [{<visibility>}]}

- -real_list_value *real_list_value*

  An optional switch and string pair that is identical to the -real_list_option switch and string pair except that it defines a value as opposed to an option-value pair.

  The real_list_value string is a Tcl list with two required elements and five optional elements. This string has exactly the same meaning as the real_list_option string except that the second element does not exist because there is no option name to define. The tag is not permitted to end with a * as only options can be defined as repeatable.

  {{<*symbol*>} {tag} [{*description_string*}] [{range}] [{*enum_list*}] [{<help_symbol>}] [{<visibility>}]}

- -integer_option *integer_option*

  An optional switch and string pair that is identical to the -integer_list_option switch and string pair except that only a single integer is permitted as the value as opposed to a Tcl list of integers.

- -string_option *string_option*

  An optional switch and string pair that is identical to the -string_list_option switch and string pair except that only a single string is permitted as the value as opposed to a Tcl list of strings.

- -real_option *real_option*

  An optional switch and string pair that is identical to the -real_list_option switch and string pair except that only a single real is permitted as the value as opposed to a Tcl list of real numbers.

- -integer_value *integer_value*

  An optional switch and string pair that is identical to the -integer_list_value switch and string pair except that only a single integer is permitted as the value as opposed to a Tcl list of integers.

- -string_value *string_value*

  An optional switch and string pair that is identical to the string_list_value switch and string pair except that only a single string is permitted as the value as opposed to a Tcl list of strings.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -real_value *real_value*

  An optional switch and string pair that is identical to the real_list_value switch and string pair except that only a single real number is permitted as the value as opposed to a Tcl list of real numbers.

- -transcript_style {tool_command | normal}

  An optional switch and literal that specifies whether the command transcribes like a normal command or like a tool command. The default is tool_command.

  tool_command —Transcribes tool commands, including all arguments, as the commands run. The command string is transcribed after being evaluated by Tcl so that all variable and command references are resolved. For more information on how a tool command transcribes, see the "set_transcript_style full" command and option.

  normal — Transcribes the command before any Tcl evaluation is done. This includes all Tcl commands and constructs. For more information on how a tool command transcribes, see the "set_transcript_style tool_commands_only" command and option.

Examples

Example 1

The following example defines a simple proc called "example" that displays the arguments it receives. The register_tcl_command defines a command called "example" that accepts a repeatable integer list option, a non-repeatable string list option, and a string value. Notice that the registered command is located in the global namespace and the Tcl proc that it calls is located in the ::ExampleCommands namespace so that the two identical proc names do not collide.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
namespace eval ExampleCommands {
  proc example {args} {
    # Initialize default values
    array set all_args {object ""}
    # convert args list into an array for easy random access
    array set all_args $args
    puts "chain = $all_args(chain)"
    # Without '--', 'is_collection' would fail if 'all_args(object)'
    # is a string that starts with a '-' since it would try to interpret
    # this value as an option to the command.
    if { [is_collection -single_type_objects -- $all_args(object)] && \
         [sizeof_collection $all_args(object)] > 0 } {
      puts "Object name(s) { [get_name_list -silent $all_args(object)] }"
      puts -nonewline " of type "
      set obj [index_collection $all_args(object) 0]
      puts "'[get_attribute_value_list -name object_type $obj -silent]'"
    } elseif { [is_collection -- $all_args(object)] && \
               [sizeof_collection $all_args(object)] > 0 } {
      foreach_in_collection obj $all_args(object) {
        puts -nonewline "Object name '[get_single_name $obj -silent]' \
        of type "
        puts "'[get_attribute_value_list -name object_type $obj -silent]'"
      }
    } else {
      puts "Object name(s) verbatim '$all_args(object)'"
    }
    # Range is repeatable, so its tags are indexed
    for {set i 0} {1} {incr i} {
      if {[info exists all_args(range,$i)]} {
      puts "range $i = $all_args(range,$i)"
    } else {
      break
    }
  }
}
```

```
register_tcl_command example \
  -string_value {{<chain_name>} {chain}} \
  -string_list_option { {<object>} {-object} {object} } \
  -integer_list_option { {<range>} {-range} {range*} } \
  -syntax_bnf {<chain_name> [<object>] [<range>]} \
  -context_list {dft patterns} \
  -system_mode_list {setup insertion} \
  -tcl_proc "[namespace current]::example"
}
```

Here is a transcript of a Tessent Shell session that uses that register command, assuming that the above content has been sourced beforehand:

SETUP> set_context dft -rtl

```
# show the help
```

SETUP> example

```
// Error: Missing options ( <chain_name> [ -object <object> ]
// [ {-range <range> ...} ] ) ( help example )
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# Basic usage with single collection type for '-object'
```

SETUP> example my_chain

```
chain = my_chain
Object name(s) verbatim ''
```

SETUP> set C [add_config_element tmp(1)]

```
// sub-command: add_config_element tmp(1)
{/tmp(1)}
```

SETUP> example my_chain -object $C

```
chain = my_chain
Object name(s) { /tmp(1) }
  of type 'config_wrapper'
```

SETUP> append_to_collection C [add_config_element tmp(2)]

```
{/tmp(1) /tmp(2)}
```

SETUP> example my_chain -object $C

```
chain = my_chain
Object name(s) { /tmp(1) /tmp(2) }
  of type 'config_wrapper'

# Example with mixed collection
```

SETUP> set_system_mode insertion
INSERTION> set m [create_module my_mod]

```
// sub-command: create_module my_mod
{my_mod}
```

INSERTION> append_to_collection C $m

```
{/tmp(1) /tmp(2) my_mod}
```

INSERTION> example my_chain -object $C

```
chain = my_chain
Object name '/tmp(1)' of type 'config_wrapper'
Object name '/tmp(2)' of type 'config_wrapper'
Object name 'my_mod' of type 'module'

# Show what happens when passing lists instead of objects

#   Empty list
```

INSERTION> example my_chain -object [list]

```
chain = my_chain
Object name(s) verbatim ''

#   List which starts with a '-'
```

INSERTION> example my_chain -object [list -a]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
chain = my_chain
Object name(s) verbatim '-a'

#   List with many elements
```

INSERTION> example my_chain -object [list -a -b -c]

```
chain = my_chain
Object name(s) verbatim '-a -b -c'

#   Show how the '-range' can be repeated
```

INSERTION> example my_chain -range 1 -range 4 -range 10

```
chain = my_chain
Object name(s) verbatim ''
range 0 = 1
range 1 = 4
range 2 = 10
```

INSERTION>

Example 2

The following example defines a wrapper Tcl proc that is used to call the intercept_with_cell procedure described in Example 5 of the get_dft_cell command. The code is surrounded in a namespace eval block which puts the intercept_with_cell_wrapper proc in that name space and avoids polluting the global namespace with the Tcl procs when sourcing it. The [namespace current] command, used with the register_tcl_command -tcl_proc option, ensures the tool knows to search in the correct name space. It also ensures that the registered Tcl command in the global namespace does not collide with the Tcl proc of the same name.

After you verify that your registered commands and attributes work correctly, you can make them into a *plugin* directory that Tessent Shell automatically loads during initialization. To create the plugin directory, simply place your files in a directory called <plugin>/tcl_modules/<MySpace> where <plugin> and <MySpace> are arbitrary directory names. Set a TESSENT_PLUGIN_PATH environment variable to point to the <plugin> directory and any files with a .tcl extension found inside <plugin>/tcl_modules/<MySpace> are evaluated into the Tcl namespace ::tcl_modules::<MySpace>. Just like the PATH environment variable, you can specify more than one search location by separating multiple directory names with a colon. A utility command initialize_module is used to load all files ending with .tcl found inside a local directory having the exact same name as the name space.

In the example below, the namespace is called InterceptWithCell which means that initialize_module sources all files matching*./InterceptWithCell/*.tcl*. The advantage of using the initialize_command is that the Tcl proc is only loaded the first time it is needed.

If you name your <plugin> directory name “tessent_plugin” and place it one level above the bin directory of Tessent Shell, the directory is automatically loaded unless the environment variable TESSENT_PLUGIN_IGNORE_DEFAULT_PATH is defined. Once the plugins are loaded, the lock_current_registration command is issued so that they cannot be unregistered accidentally.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
namespace eval InterceptWithCell {
    proc intercept_with_cell_wrapper {args} {
       initialize_module
       intercept_with_cell_body {*}$args
}
set extra_help "Command accepts a pin, port, or net object as first"
append extra_help "argument. It is intercepted with cell returned by \n"
append extra_help "get_dft_cell <cell_function_name>."

set cell_list "One of mux, clock_mux, and, clock_and, clock_or,"
append cell_list "buffer, clock_buffer, or inverter."

set bnf {<node> <function_name> [<input2_source>] [<select_source>] }
append bnf {[<level>] [<reset_intercept_list>]}

register_tcl_command intercept_with_cell \
    -context_list {dft} \
    -system_mode_list {insertion} \
    -resource_list {design_is_elaborated} \
    -extra_help $extra_help \
    -string_value {"<node>" {node} {} } \
    -string_option [list "<function_name>" "-cell_function_name" \
        {cell_sel} $cell_list {mux clock_mux and clock_and or \
        clock_or buffer clock_buffer inverter}] \
    -string_option {"<input2_source>" "-input2" {input2} {}} \
    -string_option {"<select_source>" "-select" {select} {}} \
    -integer_option {"<level>" "-cascading_level" {level} {} {1 9}} \
    -boolean_option {"<reset_intercept_list>" "-reset_intercept_list" \
        {clear}} \
    -syntax_bnf "$bnf" \
    -transcript tool_command \
    -tcl_proc "[namespace current]::intercept_with_cell_wrapper"
}
```

The online help for the registered command is shown here:

```
INSERTION> help intercept_with_cell
     intercept_with_cell <node> -cell_function_name <function_name>
              [-input2 <input2_source>] [-select <select_source>]
              [-cascading_level <level>] [-reset_intercept_list]

DESCRIPTION
   -cell_function_name: One of mux, clock_mux, and, clock_and, or,
clock_or, buffer, clock_buffer

Command accepts a pin, port, or net object as first argument.It is
intercepted with cell returned by get_dft_cell <cell_function_name>.
```

Related Topics

display_message

lock_current_registration

unregister_tcl_command

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

remove_from_collection

Context: unspecified, all contexts

Mode: all modes

Removes objects found in obj_spec from a collection, resulting in a new collection.

Usage

remove_from_collection ***base_collection obj_spec***

Description

Removes objects found in obj_spec from a collection, resulting in a new collection.

The base_collection remains unchanged.

If obj_spec is a null collection, the resulting collection is a copy of the base collection. If everything in base_collection is included in obj_spec, the result is an empty collection.

Arguments

- ***base_collection***

  A required value that specifies the base collection from which objects within obj_spec are to be removed.

- ***obj_spec***

  A required value that specifies a Tcl list of one or more object names or a collection of one or more objects.

Examples

The following example creates a collection of all input and inout ports minus the CLK port.

set data_in [remove_from_collection [get_ports -direction input inout] CLK]

```
{"in1", "in2"}
```

Related Topics

add_to_collection

append_to_collection

compare_collections

copy_collection

index_collection

is_collection

filter_collection

foreach_in_collection

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

range_collection

sizeof_collection

sort_collection

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

rename_instance

Context: all contexts

Mode: insertion

Renames the leaf name of an instance.

Usage

rename_instance ***obj_spec*** **-new_leaf_name** ***leaf_inst_name*** [-silent]

Description

Renames the leaf name of an instance.

This command returns a collection containing the renamed instance.

If the instance within obj_spec does not exist or if an object already exists with the same ***leaf_inst_name*** within the *parent_instance*, the tool generates an error. You can use the -silent switch to specify to not generate an error message and ignore the renaming request in which case the command returns a null collection.

Arguments

- ***obj_spec***

  A required value that specifies a single instance name or a collection of a single instance object as returned by the get_instances command.

- **-new_leaf_name** ***leaf_inst_name***

  A required switch and value pair that specifies the new leaf name to assign to the instance object referenced in ***obj_spec***.

- -silent

  An optional argument that specifies to ignore any of the two following error conditions: the instance within obj_spec does not exist or if an object already exists with the same ***leaf_inst_name*** within the *parent_instance*.

Examples

Example 1

The following example renames the leaf name and3 to my_and3.

```
rename_instance /modB/and3 -new_leaf_name my_and3
```

Example 2

The following example identifies all instances having the user-defined attribute is_my_type set to true and adds a special prefix called “myPrefix_” to their leaf instance name such that it is easier to find them in the downstream steps of the flow using the SDC command “get_cells myPrefix_* -hierarchical”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
foreach_in_collection inst [ get_instance -filter “is_my_type”] {
   set leaf_name [get_attribute_value_list $inst -name leaf_name]
   rename_instance $inst -new_leaf_name myPrefix_${leaf_name}
}
```

Related Topics

create_instance

delete_instances

get_common_parent_instance

replace_instances

uniquify_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

rename_module

Context: dft

Mode: insertion

Assigns a new name to an existing module.

Usage

rename_module ***object_spec*** **-new_name** ***module_name*** [-silent]

Description

Assigns a new name to an existing module. If the ***object_spec*** is a parameterized view of a module, only the particular view is renamed.

When specified, all instances of ***object_spec*** in the hierarchy are bound to the new module. The old module still exists, and instances that are not in the current hierarchy are not bound to the new module but are still bound to the old one.

Arguments

- ***object_spec***

  A required string or collection that specifies the module.

  The ***object_spec*** must match exactly one module/parameterized view; otherwise, an error is given.

- **-new_name** ***module_name***

  A required switch and string that specifies the new module name for ***object_spec***.

- -silent

  An optional argument that suppresses the error issued if the ***object_spec*** does not exist or if more than one module is specified.

Related Topics

rename_instance

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

rename_net

Context: dft

Mode: insertion

Renames the leaf name of a net.

Usage

rename_net ***obj_spec*** **-new_leaf_name** ***leaf_net_name*** [-silent]

Description

Renames the leaf name of a net.

If a net already exists with the same ***leaf_net_name***, the tool chooses an unused net name by adding the net uniquification suffix.

Arguments

- ***obj_spec***

  A required value that specifies a single net name or a collection of a single net objects as returned by the get_nets command.

- **-new_leaf_name** ***leaf_net_name***

  A required switch and value pair that specifies the new leaf name to assign to the net object referenced in ***obj_spec***.

- -silent

  An optional argument that specifies to ignore the following error condition: the net within ***obj_spec*** does not exist.

Examples

Example 1

The following example demonstrates the use of this command:

INSERTION> get_nets

```
{joe[1] joe[0]}
```

INSERTION> rename_net [get_nets joe] -new_leaf_name jack

```
{jack[1] jack[0]}
```

INSERTION> get_nets

```
{jack[1] jack[0]}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example illustrates the uniquification suffix if a net already exists with the same ***leaf_net_name***:

INSERTION> get_nets

```
{w1 w2}
```

INSERTION> rename_net [get_net w1] -new_leaf_name w2

```
{w2_ts1}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

rename_pin

Context: dft

Mode: insertion

Assigns a new name to an existing instance pin.

Usage

rename_pin ***pin_object*** **-new_leaf_name** ***leaf_pin_name*** [-silent]

Description

Assigns a new name, specifically the leaf name, to an existing instance pin.

The rename_pin command takes either a name or pin object. In both cases, only one pin or multiple pins of the same bus are permitted. When you specify a bus pin or multiple pins of the same bus, the whole bus is renamed.

Arguments

- ***pin_object***

  A required value that specifies an existing instance pin name or a collection of single pin objects as returned by the get_pins command. This is the pin or pin objects you want to rename.

- **-new_leaf_name** ***leaf_pin_name***

  A required switch and value pair that specifies the new name of the scalar or bus pin.

- -silent

  An optional switch that suppresses all errors. For example, if the pin cannot be renamed because the pin has the is_non_editable attribute set.

Examples

The following examples shows various uses of the rename_pin command:

```
rename_pin my_pin -new_leaf_name your_pin
rename_pin bus1 -new_leaf_name bus0
rename_pin bus2[0] -new_leaf_name newBusName[0]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

rename_port

Context: dft

Mode: insertion

Assigns a new name to an existing port.

Usage

rename_port ***port_object***
[-on_module *module_object*]
-new_name *port_name*
[-silent]

Description

Assigns a new name to an existing port.

Note

The command does not support renaming ports on VHDL modules.

Arguments

- ***port_object***

  A required value that specifies an existing single-port name or a collection of a single port objects as returned by the get_ports command. This is the port or port objects you want to rename.

- -on_module *module_object*

  An optional switch and value pair that specifies only an existing port or port objects on a module specified by *module_object*.

- **-new_name *port_name***

  A required switch and value pair that specifies the new name for the single port name or a collection of single port objects.

- -silent

  An optional argument that specifies to ignore the following error condition: the port within ***port_object*** does not exist.

Examples

Example 1

The following example demonstrates using the rename_port command to rename the existing port “joe” to “jack”.

INSERTION> get_ports

```
{joe[1] joe[0]}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

INSERTION> rename_port {joe} -new_name jack

```
{jack[1] jack[0]}
```

INSERTION> get_ports

```
{jack[1] jack[0]}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

replace_instances

Context: dft

Mode: insertion

Replaces the module definition of an instance with a new module definition.

Usage

replace_instances ***obj_spec1*** **-with_module** ***obj_spec2*** [-silent]

Description

Replaces the module definition of an instance with a new module definition. The module definition assigned to the instance during instantiation is replaced with the module specified by the module within ***obj_spec2***. If the new module has new ports that have the function attribute set to “tie0” or “tie1”, then these are automatically tied on the new instance pins.

Connections to pins that exist on both the original and new modules remain unchanged. New pins that did not exist on the original module are left open after the replacement. Pins that existed on the original module but do not exist on the new module are silently disconnected.

This command returns a collection containing the new instance.

If ***obj_spec1*** does not exist, the tool generates an error. You can use the -silent switch to specify to not generate an error message when the object within ***obj_spec1*** does not exist. If ***obj_spec2*** is the same module as the original module, no change is made.

Arguments

- ***obj_spec1***

  A required value that is Tcl list of one or more instance names, or a collection of one or more instances as generated by the get_instances command.

- **-with_module** ***obj_spec2***

  A required switch and value pair that specifies the new module object to assign to the instances within ***obj_spec1***. The ***obj_spec2*** value must be a single module name or a collection of a single module object as created by the s command.

- -silent

  An optional argument that specifies to ignore specified objects of ***obj_spec1*** or ***obj_spec2*** that do not exist and to not generate error messages.

Examples

Example 1

The following example replaces the module definition of the instance u1/u2 with the module named nor2.

```
replace_instances u1/u2 -with_module nor2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example replaces the module definition for instance u1/u2 to ModB. The original connection to pin u1/u2/a is preserved while the new pin u1/u2/b is left open.

get_attribute_value_list u1/u2 -name module_name

```
{ModA}
```

get_fanin u1/u2/a -stop_on net

```
{u1/n45}
```

replace_instances u1/u2 -with_module ModB

```
{u1/u2}
```

get_fanin u1/u2/a -stop_on net

```
{u1/n45}
```

get_fanin u1/u2/b -stop_on net

```
{}
```

Related Topics

create_instance

copy_module

delete_instances

get_common_parent_instance

uniquify_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_aborted_faults

Context: dft -edt, patterns -scan

Mode: analysis

Displays information on some potentially testable faults that remain undetected (UD) after the ATPG process.

Usage

report_aborted_faults [*format_type*] [{> | >>} *file_pathname*]

Description

Displays information on testable faults that remain undetected (UD) after the ATPG process.

The report_aborted_faults command can help you determine why the tool aborted faults in the undetected fault list. You can then analyze whether to change the current abort limit to possibly enable ATPG to generate tests for those faults before the tool aborts them. To change the abort limit, use the set_abort_limit command.

Arguments

- *format_type*

  An optional literal that specifies the type of information that you want the tool to display regarding the aborted faults. The literal choices for the *format_type* argument are as follows:

  SUmmary — A literal that displays a summary of the number of aborted faults for each category. This is the default.

  All — A literal that displays all the aborted faults that are currently in the undetected fault list.

  Backtrack — A literal that displays all the aborted, undetected faults that exceeded the backtrack limit.

  CLOck_control — A literal that displays all the aborted, undetected faults that are due to clock control.

  CONtention — A literal that displays all the aborted, undetected faults due to bus contention or ram write contention.

  DECisions — A literal that displays all the aborted, undetected faults that exceeded the maximum number of decisions.

  DETected — A literal that displays all the faults that the tool aborted and then later detected.

  Edt — **(Available only when EDT is on)** A literal that displays EDT aborted faults. When the tool generates an effective fault test but is unable to compress the pattern, the fault is classified as an EDT aborted fault. Each of the following increases the probability of EDT aborted faults:

  - Relatively aggressive compression (large chain-to-channel ratio)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Large number of ATPG constraints
- Relatively small design

IDdq_restriction — A literal that displays all the undetected faults that the tool aborted while trying to satisfy the IDDQ restrictions.

INterrupt — A literal that displays all the undetected faults that the tool aborted because you interrupted the ATPG process with a Control-C.

MULtiple_clock_restriction — A literal that displays all the aborted, undetected faults due to multiple clock restrictions.

TRansition — A literal that displays all the undetected faults that the tool aborted because of inconsistencies between the initial and final transition pattern.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

report_aborted_faults

The following example displays the default summary of all the aborted faults for the design.

report_aborted_faults

```
10 backtrack
1  clock_control
2  mult_clock_restriction
```

This is the same as the command report_aborted_faults summary.

Example 2

This example displays all the aborted faults that are currently in the undetected fault list.

report_aborted_faults all

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
1    UO.AAB   /I$1/I$954/I4
0    UO.AAB   /I$1/I$952/OUT
1    UO.AAB   /I$1/I$952/OUT
0    EQ       /I$1/I$952/I3
0    EQ       /I$1/I$952/I2
0    EQ       /I$1/I$952/I1
0    EQ       /I$1/I$952/I0
1    UO.AAB   /I$1/I$952/I3
1    UO.AAB   /I$1/I$952/I2
1    UO.AAB   /I$1/I$952/I1
1    UO.AAB   /I$1/I$952/I0
```

Related Topics

report_faults

set_abort_limit

set_atpg_limits

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_atpg_constraints

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays all the current ATPG state restrictions and the pins on which they reside.

Usage

report_atpg_constraints [{> | >>} *file_pathname*]

Description

Displays all the current ATPG state restrictions and the pins on which they reside.

The report_atpg_constraints command displays the pins and their state restrictions defined using the add_atpg_constraints command. The tool uses the state restrictions (constraints) during the ATPG process.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following example creates two ATPG pin constraints and then displays the information on those definitions:

```
add_atpg_functions and_b_in and /i$144/q /i$141/q /i$142/q
add_atpg_constraints 0 /i$135/q
add_atpg_constraints 1 and_b_in
report_atpg_constraints
```

```
0 /I$135/Q (23)
1 and_b_in
```

Related Topics

add_atpg_constraints

delete_atpg_constraints

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_atpg_functions

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays all the current ATPG function definitions.

Usage

report_atpg_functions [{> | >>} *file_pathname*]

Description

Displays all the current ATPG function definitions.

The report_atpg_functions command displays the definitions of the ATPG functions created using the add_atpg_functions command. You can use an ATPG function as an argument to the add_atpg_constraints command, which then lets you create state restrictions on pins that the tool uses during the ATPG process.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example creates two ATPG functions and then displays their definitions:

add_atpg_functions and_b_in and /i$143/q /i$141/q /i$142/q
add_atpg_functions select_b_in select /i$144/q /i$142/q
report_atpg_functions

```
USER_AND and_b_in
  Input 0:  /I$143/Q (27)
  Input 1:  /I$141/Q (23)
  Input 2:  /I$142/Q (25)
SELECT   select_b_in
  Input 0:  /I$144/Q (29)
  Input 1:  /I$142/Q (25)
```

Related Topics

add_atpg_constraints

add_atpg_functions

delete_atpg_functions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_atpg_timing

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Reports the current ATPG timing setup or statistical analysis of delay data.

Usage

report_atpg_timing [-DELAY_DISTribution { Static | Actual } [-SLACK]
[-RESOlution *integer*] [-DIFFerence]]
[{> | >>} *file_pathname*]

Description

By default, reports the current ATPG timing setup. Refer to "Timing-Aware ATPG" in the *Tessent Scan and ATPG User's Manual* for complete information.

The basic report uses the following format:

```
//  ATPG timing     = { ON | OFF }
//  Timing source   = { no timing |
                                          SDF minimum delay |
                                          SDF typical delay |
                                          SDF maximum delay }
//  Hold PIs        = { ON | OFF }
//  Mask POs        = { ON | OFF }
//  Exclude inter-clock paths         = { ON | OFF ]
//  Timing scale    = { N/A | 1 ms | 1 us |
                                          1 ns | 1 ps }
//  Slack margin for fault dropping   = { OFf | slack_margin |
                                          slack_margin_percentage% }
//  Delay fault distribution function = { UNIFORM | HIstogram
    | EXPONENTIAL | INV_POLY }
//        [ f(t)=function(t), (begin time = min and end time = max ) ]
//  Longest delay (rising, falling)   = ( max_rising, max_falling )
//  Clock waveform  = { UNDEFINED | USER DEFINED}
//
//    Clock Name Test Clock    System Clock
//      {Period, Offset, Width}   {Period, Offset, Width}
//    ---------------------------------------------------------------
//    clock_name     num1,num2,num3   num1,num2,num3
...
```

- **ATPG timing** — The status of timing-aware ATPG.
- **Timing source** — The source of the timing information.
- **Hold PIs** — The state of holding PIs when doing static timing analysis for arrival time, propagation time, path delay, and slack.
- **Mask POs** — The state of masking POs when doing static timing analysis for arrival time, propagation time, path delay, and slack.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **Exclude inter-clock paths** — The state of excluding inter-clock paths when doing static timing analysis for arrival time, propagation time, path delay, and slack.
- **Timing scale** — The time units used. The timing scale is set by the SDF file and defaults to 1 ns.
- **Slack margin for fault dropping** — The margin specified using the set_atpg_timing command.
- **Longest delay** — From static timing analysis.
- **Clock waveform** — The clock information specified by the set_atpg_timing command and reports the following:
  - **Clock Name**
  - **Test Clock** — The ATE testing clock frequency, which may not be the same as the normal operation frequency (System Clock). Timing-aware ATPG uses the test clock frequency.
  - **System Clock** — Normal operating frequency.

With the -DELAY_DISTribution switch, this command reports statistical analysis for delay data based on static or actual delay data. This command can perform the analysis based on minimum or maximum slack values, or based on differences between maximum and minimum values or between static and actual delay.

Arguments

- -DELAY_DISTribution { Static | Actual }

  An optional switch and literal that specifies to perform statistical analysis for delay data:

  Static — Performs analysis for static delay data. See Table 6-5.

  Actual — Performs analysis for actual delay data. See Table 6-6.

  How the analysis is performed is affected by the combination of whether the -slack and -difference switches are specified, as shown in the following tables:

Table 6-5. "-delay_distribution static" With -slack and -difference

-difference	-slack	
	Specified	**Unspecified**
Specified	Illegal	Differences between maximal static path delay and minimal static path delay through each fault
Unspecified	Minimal static slack values through each fault	Maximal static slack values through each fault

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 6-6. "-delay_distribution actual" With -slack and -difference

-difference	-slack	
	Specified	**Unspecified**
Specified	Differences between minimal actual slack and minimal static slack through each fault	Differences between maximal static path delay and maximal actual path delay through each fault
Unspecified	Minimal actual slack values through each fault	Maximal actual slack values through each fault

Note

For most cases, the command "report_atpg_timing -delay_distribution static -difference" is the most useful.

Use this command to determine if a design is a good candidate for timing-aware ATPG. If there is only a minor difference throughout the design, the transition patterns are similar to timing-aware patterns. Therefore, timing-aware ATPG may provide only limited benefit. If there is a larger difference, then timing-aware ATPG generally shows a more significant improvement over transition patterns.

- -SLACK

  An optional switch that specifies using minimal slack values in performing statistical analysis; otherwise, maximal values are used.

  Refer to Table 6-5 and Table 6-6 for information on how this switch interacts with the -delay_distribution and -difference switches.

- -DIFFerence

  An optional switch that specifies using differences between slack values in performing statistical analysis; otherwise, full slack values are used.

  Refer to Table 6-5 and Table 6-6 for information on how this switch interacts with the -delay_distribution and -slack switches.

- -RESOlution *integer*

  An optional switch and integer that specifies the number of segments to use during statistical analysis. If this is not specified, a default value of 64 segments is used.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example shows the timing-aware ATPG settings.

```
read_sdf top_worst.sdf
set_atpg_timing -clock test_clk_dd 16000 8000 8000
set_atpg_timing -clock test_clk_cc 8000 4000 4000
set_atpg_timing -clock test_clk_aa 4000 2000 2000
set_atpg_timing -clock default 36000 18000 18000
set_atpg_timing on
report_atpg_timing
```

```
//  ATPG timing      = ON
//  Timing source    = SDF typical delay
//  Time scale       = 1 ns
//  Slack margin for fault dropping    = OFF
//  Delay fault distribution function  = EXPONENTIAL
// f(t) = 0.00158*e^(-0.0021t) + 4.94e-06, (begin time=0, end time=50000)
//  Longest delay (rising, falling)    = (50547.0000, 50547.0000)
//  Clock waveform   = USER DEFINED
//
//    Clock Name                Test Clock                System Clock
//                        {Period, Offset, Width}   {Period, Offset, Width}
//    ----------------------------------------------------------------
//    /test_clk_dd (4)     16000, 8000, 8000        16000, 8000, 8000
//    /test_clk_cc (3)      8000, 4000, 4000         8000, 4000, 4000
//    /test_clk_aa (2)      4000, 2000, 2000         4000, 2000, 2000
//    /clk_in (6)          36000, 18000, 18000      36000, 18000, 18000
```

Example 2

The following example uses statistical analysis on delay distribution to determine which of two designs is a better candidate for timing-aware ATPG.

Design 1

```
report_atpg_timing -delay_distribution static -difference
```

```
   No. of faults             Range
------------------  --------------------
    1351               [0.0000, 0.2552]
    181                [0.2552, 0.5103]
    153                [0.5103, 0.7655]
    175                [0.7655, 1.0207]
    143                [1.0207, 1.2759]
    122                [1.2759, 1.5310]
    125                [1.5310, 1.7862]
    137                [1.7862, 2.0414]
    …
    4                  [15.3103, 15.5655]
    2                  [15.5655, 15.8207]
    3                  [16.0758, 16.3310]
//
//  4241 faults are analyzed for distribution about the differences
//  between maximal and minimal static path delays through each fault.
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Design 2

report_atpg_timing -delay_distribution static -difference

```
   No. of faults             Range
------------------  --------------------
    838                [0.0000, 0.7480]
    410                [0.7480, 1.4960]
    350                [1.4960, 2.2440]
    300                [2.2440, 2.9919]
    410                [2.9919, 3.7399]
    35                 [3.7399, 4.4879]
    43                 [4.4879, 5.2359]
    41                 [5.2359, 5.9839]
    …
    3                  [45.6270, 46.3750]
    1                  [46.3750, 47.1230]
    4                  [47.1230, 47.8710]
//
//  3156 faults are analyzed for distribution about the differences
//  between maximal and minimal static path delays through each fault.
//
```

Design 2 shows less balance (that is, the differences between maximal and minimal slack is larger), so it is likely to be a better candidate for timing-aware ATPG.

Related Topics

read_sdf

set_atpg_timing

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_attributes

Context: unspecified, all contexts

Mode: all modes

Creates a readable report for the set of attributes based on the usage options.

Usage

Usage 1: Creates a report of attributes defined on a list of objects

report_attributes ***obj_spec*** [-predefined | -user_defined] [-silent] [-verbose]

Usage 2: Creates a report of attributes defined for specific object types

report_attributes **-object_types *type_list*** [-predefined | -user_defined] [-silent] [-verbose]

Usage 3: Creates a report for a specific attribute name

report_attributes **-name *attribute_name*** [-silent] [-verbose]

Usage 4: Creates a report of all registered attributes

report_attributes [-predefined | -user_defined] [-silent] [-verbose]

Description

In all usages, specifying the -predefined option restricts the report to only predefined attributes, and specifying the -user_defined option restricts the report to only user-defined attributes.

Usage 1: Creates a tabular report for the attributes set on any object found in obj_spec. The attributes on an object that are inherited from an associated object are reported. For example, an attribute registered for the module object type and set on a given module object is reported in the list of attributes for the instances of that module. By default, attributes that have not been set on the objects are not listed in the report unless the attribute was registered with the -show_default option. If an object in obj_spec does not exist, an error message is displayed unless the -silent switch is specified.

Usage 2: Creates a tabular report for all attributes registered on the specified object types. These do not include the inherited attributes from one object type to another such as the module attributed inherited by the associated instances.

Usage 3: Creates a detailed tabular report for a specific attribute.

Note

The Constraints column in the tabular report output reflects the min, max, and enum constraints you set while registering the attribute with the register_attribute command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***obj_spec***

  A required value used in Usage 1 that specifies a Tcl list of one or more object names or a collection of one or more objects.

- **-object_types *type_list***

  A required value used in Usage 2 to enumerate the object types for which the attribute report is requested. The supported object types are module, port, instance, pin, and net.

- -name *attribute_name*

  An optional switch and string pair that specifies the name of the attribute whose information is to be reported. This argument is mutually exclusive with the ***obj_spec*** and -object_types *type_list* arguments.

- -predefined | -user_defined

  An optional switch that constrains the report to only pre-defined attribute names or only user-defined attribute names. If this argument is not specified, by default, no constraints are applied, and both pre-defined and user-defined attributes are reported.

- -silent

  An optional switch that specifies to suppress error messages if there are objects in the ***obj_spec*** that do not exist.

- -verbose

  An optional switch that specifies to report the full list of possible attribute values. By default, only one line is dedicated to reporting the attribute values; additional values are truncated and replaced with "...".

Examples

Example 1

The following example reports the attribute definitions using the report_attributes command:

report_attributes

```
Attribute Definition Report

Attributes defined for all objects:

Name     Object   Type     Class          Default  Constraints
-----------------------------------------------------------------------
name     all      string   Pre-defined    -        -
speed    pin      integer  User-defined   200      100 to 500
added_by net      string   User-defined   John     John Patrick Susan
```

The same result would be achieved by executing:

report_attributes -object_types all

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example reports the attribute definitions for the pin object type:

report_attributes -object_types pin

```
Attribute Definition Report

Attributes defined for 'pin' objects:

Name     Object   Type     Class          Default  Constraints
-------------------------------------------------------------------
name     all      string   Pre-defined    -        -
speed    pin      integer  User-defined   200      100 to 500
```

Example 3

Scenario 1

The following example reports the attribute definitions only for a list of pin objects:

report_attributes {u7/CLK u6/A2}

```
Attribute Definition Report

Attributes defined for object 'u7/CLK':
Name                    Value   Inheritance
---------------------   ------  -----------
base_name               u7      -
direction               input   -
is_valid                true    -
leaf_name               CLK     -
module_name             dff     -
name                    u7/CLK  -
object_type             pin     -
parent_instance                 -
parent_is_hard_module   true    -

Attributes defined for object 'u6/A2':
Name                    Value  Inheritance
---------------------   -----  -----------
base_name               u6     -
direction               input  -
is_valid                true   -
leaf_name               A2     -
module_name             xor3   -
name                    u6/A2  -
object_type             pin    -
parent_instance                -
parent_is_hard_module   true   -
```

Scenario 2

This example reports the attributes for all scan elements of the class wrapper.

report_attributes [get_scan_elements -class wrapper]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Attribute Definition Report
Attributes defined for object '/\cnt_s_reg[0] ':
Name                      Value                      Inheritance
----------------------    ----------------------     -----------
at_speed_control_group    2                          -
cell_type                 scan_cell                  module
class                     wrapper                    -
clk_equation                                         -
clock_domain              inferred_occ_clock0        -
clock_edge                pos_edge                   -
clock_group               -1                         -
exists                    true                       -
has_children              false                      -
has_no_definition         false                      module
id                        3                          -
is_created                false                      module
is_hard_module            true                       module
is_modified               false                      module
is_non_editable           true                       instance
is_non_editable_reason    hard_module                instance
is_unsaved                false                      module
is_valid                  true                       -
language                  tessent_lib                module
leaf_name                 \cnt_s_reg[0]              instance
length                    1                          -
master_module_name        sff                        instance
module_name               sff                        instance
name                      /\cnt_s_reg[0]             -
object_type               scan_element               -
parent_instance                                      instance
power_domain_island                                  -
scan_in                   SI                         -
scan_out                  Q                          -
se_equation               se[0]                      -
si_clock                  inferred_occ_clock0        -
si_clock_edge             pos_edge                   -
si_equation                                          -
simulation_function       scan_cell                  module
so_clock                  inferred_occ_clock0        -
so_clock_edge             pos_edge                   -
so_equation                                          -
state                     usable                     -
static_control_group      -1                         -
type                      leaf_cell                  -
unrolled_leaf_name        \cnt_s_reg[0]              instance
unrolled_name             \cnt_s_reg[0]              instance
used                      true                       -
wrapper_type              output                     -
wrapper_type_reason       inferred_from_analysis     -

Attributes defined for object '/\cnt_s_reg[1] ':
Name                      Value                      Inheritance
----------------------    ----------------------     -----------
at_speed_control_group    2                          -
cell_type                 scan_cell                  module
class                     wrapper                    -
clk_equation                                         -
clock_domain              inferred_occ_clock0        -
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
clock_edge              pos_edge                -
clock_group             -1                      -
exists                  true                    -
has_children            false                   -
has_no_definition       false                   module
id                      4                       -
is_created              false                   module
is_hard_module          true                    module
is_modified             false                   module
is_non_editable         true                    instance
is_non_editable_reason  hard_module             instance
is_unsaved              false                   module
is_valid                true                    -
language                tessent_lib             module
leaf_name               \cnt_s_reg[1]           instance
length                  1                       -
master_module_name      sff                     instance
module_name             sff                     instance
name                    /\cnt_s_reg[1]          -
object_type             scan_element            -
parent_instance                                 instance
power_domain_island                             -
scan_in                 SI                      -
scan_out                Q                       -
se_equation             se[0]                   -
si_clock                inferred_occ_clock0     -
si_clock_edge           pos_edge                -
si_equation                                     -
simulation_function     scan_cell               module
so_clock                inferred_occ_clock0     -
so_clock_edge           pos_edge                -
so_equation                                     -
state                   usable                  -
static_control_group    -1                      -
type                    leaf_cell               -
unrolled_leaf_name      \cnt_s_reg[1]           instance
unrolled_name           \cnt_s_reg[1]           instance
used                    true                    -
wrapper_type            output                  -
wrapper_type_reason     inferred_from_analysis  -
```

Example 4

The following example reports the attribute definitions for the speed attribute:

report_attributes -name speed

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Attribute Definition Report

Name: Speed
Description:

   This attribute is used to hold the target frequency that the pin is
   allowed to have.
   The allowed frequency range of 100 to 500 was chosen to match
   the technology spec.

Global Option Settings:
Setting              Value
--------------------------------------------
Class                User-Defined
Type                 int
Default              200
Show_Default         false
Minimum              100
Maximum              500
Object               pin

Specific 'pin' Type settings:
Name                                 Value
--------------------------------------------
dense                                false
export_to_netlist                    off
preserve_boundary_in_flat_model      off
```

Example 5

The following example reports the possible cell_type attribute values with the -verbose switch both enabled and disabled.

report_attribute -name cell_type

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Attribute Definition Report

Name:
        cell_type

Description:
This is a library model attribute

Setting       Value
------------  -------------------------------------
Class         Pre-Defined
Value_Type    string
Enum          and buffer clock_and clock_buffer ...

GUI_Strings   cell_type = %
Object_Types  module

Specific 'module' Object Type settings:
Name                              Value
-------------------------------   -----
export_during_write               auto
preserve_boundary_in_flat_model   off
applies_to_child_instances        on
display_in_gui                    off
gui_marking_index                 1
```

report_attribute -name cell_type -verbose

```
Attribute Definition Report

Name:
        cell_type

Description:
This is a library model attribute

Setting    Value
---------- ------------------------------------------------------------
Class      Pre-Defined
Value_Type string
Enum       and buffer clock_and clock_buffer
           clock_gating_and clock_gating_or clock_inverter clock_mux clock_or
           dff inverter latch mux nand
           nor or pad scan_cell synchronizer_cell
           xor
GUI_Strings   cell_type = %
Object_Types  module

Specific 'module' Object Type settings:
Name                              Value
-------------------------------   -----
export_during_write               auto
preserve_boundary_in_flat_model   off
applies_to_child_instances        on
display_in_gui                    off
gui_marking_index                 1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

get_attribute_list

get_attribute_option

get_attribute_value_list

register_attribute

set_attribute_options

unregister_attribute

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_bisr_repair_register_icl_instances

Context: all contexts

Mode: all modes

Reports a list of BISR repair registers found in the design.

Usage

report_bisr_repair_register_icl_instances [-power_domain_group *pdg_list*]

Description

Reports the list of available memory BISR repair registers found in the design, providing the ICL instance name, design instance name and power domain group for each register. You can use the -power_domain_group argument to specify a listing of repair registers within specific power domain groups.

You must have already inserted the ICL modules with the process_dft_specification command and extracted the top-level ICL description with the extract_icl command before using report_bisr_repair_register_icl_instances.

Arguments

- -power_domain_group *pdg_list*

  An optional argument and value pair that specifies one or more power domain groups from which the report lists available memory BISR repair register instances.

  The *pdg_list* value supports glob expressions to aid in defining the desired list. For example, providing the argument -power_domain_group [list pdg*] reports the BISR register ICL instances for all registers that are on a power domain group that begins with “pdg”.

Examples

The following example reports the list of available memory BISR repair register instances on power domain groups “-” and “pdgA”:

report_bisr_repair_register_instances -power_domain_group [list - pdgA]

```
// BISR Repair Registers
// =====================
// ----------------  -----------------------  ---------------------------
// Power Domain Group   ICL Instance             Design Instance
// ----------------  -----------------------  ---------------------------
// -             core_blkA_i1_mem6_bisr_inst  core/blkA_i1/mem6_bisr_inst
// pdgA          core_blkA_i2_mem6_bisr_inst  core/blkA_i2/mem6_bisr_inst
// -             core_blkB_i1_memA_bisr_inst  core/blkB_i1/memA_bisr_inst
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_bist_capture_ranges

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays a list of the currently defined BIST capture ranges.

Usage

report_bist_capture_ranges

Description

The report_bist_capture_ranges command displays a list of the capture ranges that are currently defined as a result of the add_bist_capture_range command. A BIST capture range is defined by a named capture procedure that is stored in the test procedure file, and a range of patterns that use this capture procedure. The complete range of patterns reported by the report_bist_capture_ranges command typically represents a small subset of the entire pseudo-random pattern space, and is applied to the complete pattern space by repeating the sequence until all the pseudo-random patterns have been simulated.

Arguments

None

Examples

The following example shows the output from the report_bist_capture_ranges command.

report_bist_capture_ranges

```
// List of currently defined Logic BIST capture ranges
// Procedure name  from pattern  to pattern
// --------------  ------------  ----------
// cap1                0           14
// cap2               15           15
//
// NOTE: This list defines the active named capture range during pseudo-
// random pattern simulation. The tool automatically restarts at the top
// of the list when the pattern counter matches the final "to pattern"
// value.
```

Related Topics

add_bist_capture_range

delete_bist_capture_ranges

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_black_boxes

Context: all contexts

Mode: setup, analysis

Displays information on blackboxes and undefined models.

Usage

report_black_boxes
{{**-INStances** [*ins_pathnames*]} | {**-Modules** [*module_names*]} | **-All** | **-Undefined**}
[-INClude_pins]
[{> | >>} *file_pathname*]

Description

Displays information on blackboxed modules, instances, and undefined modules.

The report_black_boxes command reports on the status of any instance-based or module-based blackboxes, or undefined models that are not yet blackboxed. The tool displays the string MODULE/MODEL/INSTANCE followed by the name in the following format:

```
// Black Boxed Modules and Instances:
// Default tie value = X
// MODULE:
// MODEL:
// INSTANCE:
```

Displaying Pins

When you specify the optional -INClude_pins switch, the tool displays a list of pins as in the following example:

```
//  Black Boxed Modules and Instances:
//  Default tie value = X
//    MODULE: 'foo'
//      USER:   Output pin 'b'  tied to X
//    MODEL: xor02
//      USER:   Output pin 'Y'  tied to X
//    INSTANCE: 'ha1'
//      USER:   Output pin 's'  tied to X
//      USER:   Output pin 'c_out'      tied to X
```

Arguments

- **-Instances** [*ins_pathnames*]

  A switch and optional string that specify for the tool to display information on instance-based blackboxes. The *ins_pathnames* argument can be a Tcl list of one or more instances, or a collection of one or more instances. If you do not specify an *ins_pathnames*, the tool displays information on all instance-based blackboxes. If you specify an *ins pathnames*, it reports on that single, instance-based blackbox.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-Modules** [*module_names*]

  A switch and optional string that specify for the tool to display information on module-based blackboxes. The *module_names* argument can be a Verilog module or Tessent Cell library model, a Tcl list of modules, or a collection of modules. If you do not supply a *module_name*, the tool displays information on all module-based blackboxes. If you specify a *module_name*, it reports on that single, module-based blackbox.

- **-All**

  A switch that specifies for the tool to display information on all defined blackboxes, as well as undefined models. This is the command default.

- **-Undefined**

  A switch that specifies for the tool to display information on undefined models that have not yet been blackboxed. Use this switch to determine whether your design is complete, or is missing library models. If you intend to blackbox undefined models, this report enables you to verify that only the intended models are undefined.

- -INClude_pins

  An optional switch that specifies reporting pins for defined blackboxes. See "Displaying Pins."

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of file_pathname.

Examples

The following example defines module- and instance- based blackboxes then reports on them.

```
add_black_box -module halfadder -pin s 1
add_black_box -instance final_carry -pin Y 0
report_black_box -include_pins
```

```
//  Black Boxed Modules and Instances:
//  Default tie value = X
//    MODULE: 'halfadder'
//      USER:   Output pin 's'  tied to 1
//      USER:   Output pin 'c_out'      tied to X
//    INSTANCE: 'final_carry'
//      USER:   Output pin 'Y'  tied to 0
```

Related Topics

add_black_boxes

delete_black_boxes

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_boundary_scan_port_options

Context: dft, patterns

Mode: all modes

Displays a table listing all of the ports with their inferred or specified cell_options.

Usage

report_boundary_scan_port_options

Description

Displays a table listing all of the ports with their inferred or specified cell_options as specified by the set_boundary_scan_port_options command.

When the design level is physical_block or sub_block, as specified using the set_design_level command, all ports not listed in the "set_boundary_scan_port_options -pad_io_ports" list are inferred as dont_touch. For more information on the dont_touch, see Table 13-5 for a description of these values.

Arguments

None.

Examples

The following example runs the report_boundary_scan_port_options command on a design in which the level was set to physical_block using the set_design_level command:

```
set_design_level sub_block
set_boundary_scan_port_options -pad_io_ports [get_port gpio*]
set_boundary_scan_port_options gpio00 -cell_options clock
report_boundary_scan_port_options
```

```
//  Pad io pins : gpio00 gpio01 gpio02 gpio03 gpio04 gpio05 gpio06 gpio07
//
//   Port name                Options
//   ----------------------------------
//   gpio00                   clock
//   gpio01                   normal
//   gpio02                   normal
//   gpio03                   normal
//   gpio04                   normal
//   gpio05                   normal
//   gpio06                   normal
//   gpio07                   normal
//   clk                      dont_touch
//   ina                      dont_touch
//   outa                     dont_touch
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following example runs the report_boundary_scan_port_options command on a design in which the level was set to chip using the set_design_level command, and has had pin mapping set for several ports with the set_boundary_scan_port_options command:

```
set_design_level chip
set_boundary_scan_port_options {tms tck trst tdi tdo} \
   -packed_pin_name {jtag1 jtag2 jtag3 jtag4 jtag5}
report_boundary_scan_port_options
```

```
//   Port name Options Packed pin name
//   ---------------------------------
//   tdo       normal  jtag5
//   tdi       normal  jtag4
//   tms       normal  jtag1
//   tck       normal  jtag2
//   trst      normal  jtag3
//   a         normal
//   b[1]      normal
//   b[0]      normal
//   c[1]      normal
//   c[0]      normal
//   d         normal
//   e         normal
//   f[1]      normal
//   f[0]      normal
//   g         normal
//   clk1      normal
//   clk2      normal
```

Related Topics

get_boundary_scan_port_option

set_boundary_scan_port_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_bsdl

Writes a report of the BSDL data stored for the die in a 2.5D design.

Usage

report_bsdl

Description

Writes a report of the BSDL data stored in memory by the read_bsdl command.

Arguments

None.

Examples

This example reports the BSDL data for three instances of a die in a 2.5D design.

```
report_bsdl
//
//  // ------  --------  ----------------------------------------
//  // Entity  Instance  BSDL File
//  // ------  --------  ----------------------------------------
//  // diee    i1        tsdb_outdir/inst/diee.inst/diee_base.bsdl
//  //         i2        tsdb_outdir/inst/diee.inst/diee_iop0.bsdl
//  //         i3        tsdb_outdir/inst/diee.inst/diee_iop1.bsdl
//  //
```

If you read a BSDL file without using the -for_instances switch, "all" appears in the instance column.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_bsdl_extraction_options

Context: dft (with no subcontext)

Mode: setup, analysis

Reports all values for various BSDL attributes previously set with the set_bsdl_extraction_options command.

Usage

report_bsdl_extraction_options

Description

Displays a table listing for various BSDL attributes previously set by the set_bsdl_extraction_options command for the BSDL extraction.

Arguments

- None.

Related Topics

delete_bsdl_extraction_option

get_bsdl_extraction_option

set_bsdl_extraction_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_bsdl_extraction_user_instructions

Context: dft (with no subcontext)

Mode: setup, analysis

Reports added user instructions.

Usage

report_bsdl_extraction_user_instructions

Description

Writes a report listing all user instructions added with the add_bsdl_extraction_user_instruction command.

Arguments

None.

Related Topics

add_bsdl_extraction_user_instruction

delete_bsdl_extraction_user_instruction

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_bus_data

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays the bus data information for either an individual bus gate or for the buses of a specific type.

Usage

report_bus_data ***type*** [{> | >>} *file_pathname*]

Description

Displays the bus data information for either an individual bus gate or for the buses of a specific type.

The report_bus_data command displays the following bus information:

- Instance name
- Gate identification number
- Contention handling (pass, bidi, fail, or abort)
- Type of bus (strong or weak)
- Number of drivers on the bus
- Any learned behavior of the bus

The design rule that checks for bus contention mutual exclusivity is rule E10. For more information on rule E10, refer to the "Extra Rules (E Rules)" section.

If you enable the learn reporting by using the set_learn_report command, the tool provides the following two additional lines of information with the report_bus_data command:

- Information on whether or not the bus is capable of being set to an X, set to a Z, or having multiple drivers turned on.
- A list of drivers and their corresponding gate type. Drivers that are equivalent have a gate type of EQ.

The design rule that checks to see if there is any possible input combination that can force a bus into the high-impedance state (Z) is rule E11. For more information on rule E11, refer to the "Extra Rules (E Rules)" section.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***type***

  A required literal or integer that specifies the type of bus for which you want the tool to display information. The choices for the ***type*** argument are as follows:

  gate_id# — An integer that specifies the gate identification number whose bus data you want to display.

  ALl — A literal that displays the bus data for all buses.

  Weak — A literal that displays the bus data for the weak buses.

  Strong — A literal that displays the bus data for the strong buses.

  Dominant — A literal that displays the bus data for the final bus of every set of cascaded buses.

  NONDominant — A literal that displays the bus data for all but the final bus in every set of cascaded buses.

  Pass — A literal that displays the bus data for the buses that passed the contention mutual exclusivity checking.

  BIdi — A literal that displays the bus data for the bidirectional buses that have possible contention problems. For the tool to place a bus in this category, the bidirectional pin must have only a single tri-state driver.

  BIDI_Constrained — A literal that displays the bus data for the bidirectional buses that have possible contention problems and that are constrained.

  Fail — A literal that displays the bus data for the buses that failed the contention mutual exclusivity checking.

  ABort — A literal that displays the bus data for the buses that aborted contention mutual exclusivity checking.

  Buf — A literal that displays the bus data for all buses that have the buffer learned behavior.

  Xor — A literal that displays the bus data for all buses that have the exclusive OR learned behavior.

  Mux — A literal that displays the bus data for all buses that have the multiplexer learned behavior.

  AND — A literal that displays the bus data for all buses that have the AND learned behavior.

  OR — A literal that displays the bus data for all buses that have the OR learned behavior.

  POSS_Mult_dr_on — A literal that displays the bus data for all buses that have a possibility of having multiple drivers turned on at the same time.

  POSS_X — A literal that displays the bus data for all buses that have a possibility of being at an unknown state.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

POSS_Z — A literal that displays the bus data for all buses that have a possibility of being at the high-impedance state.

HIstogram — A literal that displays a summary of information identifying the number of buses that are in each of the possible categories.

ZPass — A literal that displays the bus data for the buses that pass the E11 design rule.

ZFail — A literal that displays the bus data for the buses that fail the E11 design rule check.

ZAbort — A literal that displays the bus data for the buses that abort the E11 design rule check.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

The following example displays the information on a specific bus gate—an inverter (INV):

report_bus_data 31

```
/FA1/XOR1/OUT/ (31)Handling=pass type=strong #Drivers=2 (INV)
   Bus Drivers:  30(SW) 28(SW)
```

Example 2

The following example first enables access to the static learning data, then displays both the learned information and the bus data on a specific bus gate, again, an inverter (INV):

set_learn_report on
report_bus_data 31

```
/FA1/XOR1/OUT/ (31)Handling=pass type=strong #Drivers=2 (INV)
   Learn Data :  poss_X=no, poss_Z=no, poss_mult_drivers_on=no
   Bus Drivers:  30(SW) 28(SW)
```

Related Topics

set_learn_report

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_bypass_chains

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Displays the defined EDT bypass scan chain connections.

Usage

report_bypass_chains [-All_blocks | {-Edt_chain *chain_name*} | {-Bypass_chain *chain_number*}] [{> | >>} *file_pathname*]

Description

Displays the defined EDT bypass scan chain connections. If issued from Setup mode, the report_bypass_chains command lists scan chain connections for bypass scan chains defined using the set_bypass_chains command.

If issued from analysis mode, all bypass scan chain connections are reported, whether defined by the tool or with the set_bypass_chains command.

You can display the chain connection information for all bypass chains, or optionally include the -Edt_chain or -Bypass_chain switch to display only the information for a particular bypass chain. The order, left to right, in which the display lists the scan chains in a particular bypass chain is the order in which the chains are series-connected from channel input to channel output.

The -All_blocks switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the chain connection information as described above, but only for the EDT block you have specified as the current EDT block. If you include the -All_blocks switch, the command displays information for all the EDT bypass chains in every EDT block. You set the current EDT block using either the add_edt_blocks or set_current_edt_block command. “EDT block” is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the “Modular Compressed ATPG” chapter of the *Tessent TestKompress User’s Manual*.

Arguments

- -All_blocks

  Note

  This switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. If you are using a standard (non-modular) Tessent TestKompress flow, you do not need this switch.

  An optional switch that specifies to display all chain connection information for all EDT blocks. Without this switch, the tool displays information for only the current EDT block, if

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

it is defined. If a current EDT block is not defined, the tool displays the information for all EDT blocks.

- **-**Edt_chain *chain_name*

  An optional switch and string pair that specify to display the scan chain connections for only the EDT bypass chain that includes the EDT scan chain, chain_name. You can use this switch to see which bypass chain or scan channel is associated with a particular scan chain.

- **-**Bypass_chain *chain_number*

  An optional switch and positive integer pair that specify to display the scan chain connections for the EDT bypass chain, chain_number. Use this switch to list the scan chain connections for a particular EDT bypass chain. The chain_number is the same as the number of the scan channel connected to the bypass chain.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

Assume a design has eight scan chains named “chain1” through “chain8” and that the chains have been incorporated into two EDT bypass channels configured from two EDT scan channels. The following example displays the scan chain connection information for the EDT bypass chain containing chain3. You can see from the report that chain3 is part of a bypass chain connected to scan channel 2.

report_bypass_chains -edt_chain chain3

```
//     Reporting user-defined bypass chains:
//
//     Bypass#        EDT chains
//     -------        ------
//        2         chain2, chain3, chain4, chain7, chain8
```

Example 2

The following example displays the connection information for EDT bypass chain 1 (scan channel 1).

report_bypass_chains -bypass_chain 1

```
//    Reporting user-defined bypass chains:
//
//    Bypass#     EDT chains
//     -------    ------
//         1     chain1, chain5, chain6
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Again, notice that the EDT bypass chain number (Bypass# in the display) is the same as the number of the scan channel to which the bypass chain is connected.

Related Topics

reset_bypass_chains

set_bypass_chains

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_capture_procedures

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: analysis

Displays the cyclized information for a specified named capture procedure.

Usage

report_capture_procedures [-CLock_sequence] [-CReated_capture_procedure | -Loaded_capture_procedure | -ENabled | -Disabled | *capture_procedure_name*...] [-All | -Summary | -Internal | -External] [{> | >>} *file_pathname*]

Description

Cyclizing is a processing step performed prior to ATPG that merges internal mode cycles, when needed, to optimize them and expand other internal mode cycles to ensure correct simulation results.

For example, when a named capture procedure contains a cycle that pulses multiple clocks at different times, the cycle must be expanded into multiple cycles (one for each clock pulse) in order to simulate it. The cyclized information shows the cycles ATPG actually uses when creating patterns after any merging or expansion of cycles. The display also includes comments about the timing of the cycles and where the at-speed sequences begin and end to aid your understanding.

The displayed information resembles the original procedure as defined in the test procedure file, but it does not have any timeplates. Therefore, it does not work if copied and pasted as-is into a dofile; you must add timeplates. To display the procedure as it actually exists in the test procedure file, use the report_procedures command.

The number of cycles reported is the sequential depth used during test generation. The display of merged or expanded cycles is illustrated in the examples. See also "Named Capture Procedures Display" in the *Tessent Scan and ATPG User's Manual*.

Note

If the named capture procedure is modified to cyclize it for ATPG, the modifications are internal only; the named capture procedure is not altered in the test procedure file.

Arguments

- -CLock_sequence

  An optional switch that displays the clock sequence of the named capture procedures. The scan load cycle(s), at-speed cycle(s), and the number of patterns the named capture procedure is associated with are also displayed.

- -CReated_capture_procedure

  An optional switch that displays only the created named capture procedures.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Loaded_capture_procedure

  An optional switch that displays only the named capture procedures loaded from the read_procfile command or from the procedure file associated with the scan group definition.

- -ENabled

  An optional switch that displays only enabled named capture procedures.

- -Disabled

  An optional switch that displays only named capture procedures that are disabled.

- *capture_procedure_name*

  An optional, repeatable string that displays only the specified named capture procedures. By default, information for all named capture procedures is displayed.

- -All

  An optional switch that displays information for both the internal and external modes of the named capture procedure. This is the default.

- -Summary

  An optional switch that displays only summary information about the named capture procedure.

- -Internal

  An optional switch that displays information for only the internal mode of the named capture procedure.

- -External

  An optional switch that displays information for only the external mode of the named capture procedure.

- > *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list to create or replace the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list to append the contents of *file_pathname*.

Examples

Example 1

The following example displays summary information for the named capture procedures nam_cap1, nam_cap2, my_cap1, and my_cap2, including the scan load cycle(s), the at-speed cycle(s), and the number of patterns with which the named capture procedures are associated. It also indicates nam_cap1 and nam_cap2 were loaded from an external file, and my_cap1 and my_cap2 were internally created.

report_capture_procedures -summary

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
capture_proc_name  cycles  at_speed_cycle  active  source  # associated patterns
                                                            internal   external
-----------------  ------  --------------  ------  -------  ----------  --------
nam_cap1             1          1           no     loaded        0         -
nam_cap2             1          1           yes    loaded       10         -
my_cap1              2       {1 to 2}       yes    created       5         -
my_cap2              2       {1 to 2}       no     created       0         -
```

Example 2

The following example displays the clock sequence of the named capture procedures capture1, capture2, user_capture_proc, and user_capture_proc_1 in addition to the summary information.

report_capture_procedures -clock_sequence

```
capture_proc_name cycles at_speed_cycle active source #associated patterns capture_clock_sequence
                                                       internal   external
----------------- ------ -------------- ------ ------- ---------- -------- ----------------------
capture1            1         1         yes   loaded       6        -      S{c4,c3,c2,c1,c0}
capture2            1         1         yes   loaded       8        -      S{c9,c8,c7,c6,c5}
user_capture_proc   3     {1 to 3}      yes   created      0        -      S{c0){c1,c0}{c3,c2}
user_capture_proc_1 3     {1 to 3}      yes   created      0        -      S{c0}{c1,c0}{c3,c2}
```

Example 3

The following example displays information for all the named capture procedures (capture1, capture2, and my_cap1). The first line displayed for each named capture procedure indicates if the procedure is currently enabled for test generation (ACTIVE) or disabled (INACTIVE). You can enable or disable capture procedures with the set_capture_procedures command.

report_capture_procedures

```
procedure capture "capture1" = //  (INACTIVE) (loaded capture procedure)
   cycle = // cycle 1 starts at time 0 (Begin_AC_test) (Force PI) (Measure PO)
      condition  "/inst_0/clkenhold/reg_sco/"  1 ;
      force  "/clock[4]"  0 ;
      force  "/clock[3]"  0 ;
      force  "/clock[2]"  0 ;
      force  "/clock[1]"  0 ;
      force  "/clock[0]"  0 ;
      force  "/clear"  0 ;
      force  "/scan_en"  0 ;
      force  "/test_clk"  0 ;
      force_pi ;
      measure_po ;
      pulse "/clock[1]" ;
      pulse "/clock[0]" ;
   end ; // cycle 1 (End_AC_test)
   // mode external is not defined.
end ; // procedure "capture1"
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
procedure capture "capture2" = //  (ACTIVE) (loaded capture procedure)
   cycle = // cycle 1 starts at time 0 (Begin_AC_test) (Force PI) (Measure PO)
      condition  "/inst_0/clkenhold/reg_sco/"  0 ;
      force  "/clock[4]"  0 ;
      force  "/clock[3]"  0 ;
      force  "/clock[2]"  0 ;
      force  "/clock[1]"  0 ;
      force  "/clock[0]"  0 ;
      force  "/clear"  0 ;
      force  "/scan_en"  0 ;
      force  "/test_clk"  0 ;
      force_pi ;
      measure_po ;
      pulse "/clock[4]" ;
      pulse "/clock[3]" ;
   end ; // cycle 1 (End_AC_test)
   // mode external is not defined.
end ; // procedure "capture2"

procedure capture "my_cap1" = //  (ACTIVE) (created capture procedure)
   cycle = slow // cycle 1 starts at time 0 (Force PI)
      force  "/clock[4]"  0 ;
      force  "/clock[3]"  0 ;
      force  "/clock[2]"  0 ;
      force  "/clock[1]"  0 ;
      force  "/clock[0]"  0 ;
      force  "/clear"  0 ;
      force  "/scan_en"  0 ;
      force  "/test_clk"  0 ;
      force_pi ;
      measure_po ;
      pulse "/clock[4]" ;
      pulse "/clock[3]" ;
   end ; // cycle 1
   cycle = // cycle 2 starts at time 50 (Begin_AC_test) (Force PI) (Measure PO)
      condition  "/inst_0/clkenhold/reg_sco/"  0 ;
      force  "/clock[4]"  0 ;
      force  "/clock[3]"  0 ;
      force  "/clock[2]"  0 ;
      force  "/clock[1]"  0 ;
      force  "/clock[0]"  0 ;
      force  "/clear"  0 ;
      force  "/scan_en"  0 ;
      force  "/test_clk"  0 ;
      force_pi ;
      measure_po ;
      pulse "/clock[2]" ;
      pulse "/clock[1]" ;
   end ; // cycle 1 (End_AC_test)
   // mode external is not defined.
end ; // procedure "my_cap1"
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

The following example displays how the tool uses the cycle during ATPG (the command's output is excerpted for brevity):

report_capture_procedures -internal

```
...
   Mode internal =
      Cycle =   // cycle 1 starts at time 0 (Begin_AC_test)
                // (Force PI)
         Force_pi ;
         Pulse /clk1 ;
      End ;    // cycle 1 at time 20
      Cycle =  // cycle 2 starts at time 20
         Pulse /clk2 ;
      End ;    // cycle 2 at time 50
...
```

The display shows that for this particular design and cycle definition, the tool expands the one cycle from the test procedure file into two cycles internally. Notice that the combined time of the two cycles is the same as the period of the original cycle in the test procedure file. The tool automatically adjusts the sequential depth used during ATPG to accord with the resultant change in number of cycles.

Example 5

The next example shows a case where the tool merges two cycles described in the test procedure file into one cycle for ATPG. The relevant excerpts from the test procedure file are shown first; the example then displays the cyclized data for the three cycles.

```
mode_internal =
...
   timeplate tp2;
   cycle =
      force_pi;
      force tms 0;
      pulse clk;
   end;
   cycle =
      force tms 0;
   end;
   cycle =
      force tms 0;
      pulse clk;
   end;
...
```

```
timeplate tp2 =
   force_pi 0;
   pulse clk 10 10;
   period 50;
end;
```

report_capture_procedures -internal

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
...
   Mode internal =
      Cycle =  // cycle 1 starts at time 0 (Begin_AC_test)
         // (Force PI)
         Force_pi ;
         Force /tms 0 ;
         Pulse /clk ;
      End ;    // cycle 1 at time 100
      Cycle =  // cycle 2 starts at time 100
         Force /tms 0 ;
         Pulse /clk ;
      End ;    // cycle 2 at time 150
```

Related Topics

create_capture_procedures

report_procedures

report_timeplates

set_capture_procedures

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_capture_timing_options

Context: all contexts

Mode: all modes

Reports the capture timing options defined for a specified icl_module, module, or the current physical block.

Usage

report_capture_timing_options
[-icl_modules *icl_module_names* | -modules *module_names* | -current_physical_block | -all]

Description

Reports the capture timing options defined for a specified icl_module, module, or the current physical block.

If you specified a module with the set_current_physical_block command and the capture timing options for the module, the tool reports those values. If you did not specify a module, the tool traverses the design hierarchy until it finds a parent that was specified as a physical block. Otherwise, the tool uses the module inherited from the current design and reports the default values if no options were set.

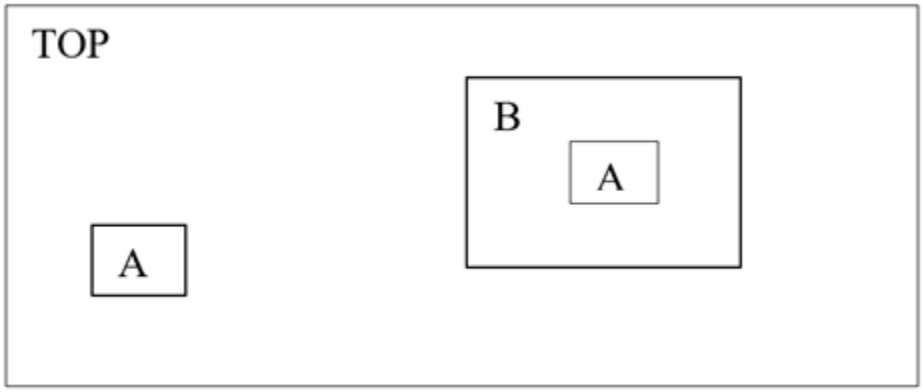
If the module has multiple instantiations and all its instances inherit from the same parent, then the tool reports the values of that parent. If the instances of module inherit from different parents (see Figure 6-11), then the tool issues the following error:

```
//  Error: The specified Module 'foo'  was not set as a physical block by
//  'set_current_physical_block ' command. It has multiple instances that
//  do not belong to the same physical block.
```

For example, Figure 6-11 shows that module A was never specified as a physical block, and module B was specified as a physical block. Because module A has two instances, one of them inherits from block B, and the other inherits from current design TOP. This scenario will result in an error.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 6-11. Design Inheritance and Module Matching



Arguments

- -icl_modules *icl_module_names*

  A switch and string pair that specifies the name of a valid ICL module. You can specify more than one ICL module name.

- -modules *module_names*

  A switch and string pair that specifies the name of a module. You can specify more than one module name.

- -current_physical_block

  A switch that specifies the current physical block.

- -all

  A switch that reports the capture timing options for all specified current physical blocks, including any previously specified using the set_current_physical_block command.

Related Topics

get_capture_timing_option

reset_capture_timing_options

set_capture_timing_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_cell_constraints

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays a list of the constrained cells.

Usage

report_cell_constraints [-Scan_cell | -Non_scan_cell]
[{-CHain *chain_name*} | {-CLock *clock_name*}]
[{> | >>} *file_pathname*]

Description

The report_cell_constraints command displays a list of all the scan cells (also non-scan cells for DX and SX constraints) that you previously constrained to a constant value using the add_cell_constraints command. The display may consist of up to seven columns, as follows (columns that contain no data for all cell constraints are not displayed):

- Constraint value — Specifies the constraint value.
- Chain name — Specifies the scan chain name if the constraint is on a scan cell. If the constraint is on a non-scan cell, this column is blank.
- Cell position — Specifies the position in the scan chain if the constraint is on a scan cell. If the constraint is on a non-scan cell, this column is blank.
- Cell instance name — Specifies the cell instance of the constraint. This column is blank unless you originally specified the constraint with a *pin_pathname*.
- Gate instance name — Specifies the gate instance of the constraint. This column is blank unless you originally placed the constraint on a non-scan cell using the -Clock switch.
- Constraint properties — Displays constraint properties, as applicable, from the following list:
  - Internal — Constraint was applied by the tool automatically in response to certain DRC results.
  - Whole Chain — Constraint was applied using the -Chain switch.
  - Drc C6 — Constraint was applied using the “-Drc C6” argument combination.
  - Clock — Constraint was applied using the -Chain switch.
  - Drc D8 Warn — Constraint was applied by the tool automatically to a D8 failing master cell when “set_drc_handling d8 warning” was in effect for DRC. See “D8” for more information.
  - Dynamic — Constraint was applied in analysis mode; DRCs are performed when switching from setup mode to analysis mode.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

  - Static — Constraint was applied in setup mode; DRCs use the static cell constraint so it cannot be removed in analysis mode.

- Pattern scope — Specifies the pattern scope of the constraint: whether the tool uses the constraint only when creating scan patterns, or uses the constraint for both scan and chain pattern creation.

Arguments

- -Scan_cell

  An optional switch that specifies to report_cell_constraints at scan cells only. This switch is valid in analysis modes only.

- -Non_scan_cell

  An optional switch that specifies to report_cell_constraints at non-scan cells only. This switch is valid in analysis modes only.

- -CHain *chain_name*

  An optional switch and string pair that specifies to report_cell_constraints added to the scan chain, chain name, by an earlier add_cell_constraints -Chain command. This switch is valid in analysis modes only.

- -CLock *clock_name*

  An optional switch and string pair that specifies to report the cell constraints added by an earlier add_cell_constraints -Clock command to latches and flip-flops controlled by the clock, clock_name. This switch is valid in analysis modes only.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

The following example begins in Setup mode and constrains a single scan cell to be at a constant zero for the generation of all patterns. The example then displays current cell constraints, leaves Setup mode, triggering the tool's automatic DRC process and again reports the cell constraints:

add_cell_constraints sdff1/q c0 -all_patterns
report_cell_constraints

```
// constraint cell_instance  constraint     pattern scope
//    value       name       properties
// ---------- -------------  ---------- ---------------------
//     C0        sdff1/q      (Static)  Chain + Scan Patterns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_system_mode analysis
report_cell_constraints

```
// const chain  cell    cell_inst constraint     pattern scope
// value  name  pos        name   properties
// ----- -----  -----  ---------- ---------- --------------------
//   C0    c1    2        sdff1/q  (Static)  Chain + Scan Patterns
```

Notice that once the tool has identified the scan chains during the DRC process, the report identifies the chain and cell position of the constrained scan cell.

Example 2

The next example adds DX constraints on all flip-flops clocked by clk2. You can see from the cell constraint report that these flip-flops are not scan cells (no chain or cell position). Notice also that the gate instances are reported for them since each flip-flop is a standalone gate, not part of a scan cell.

add_cell_constraints -clock /clk2 -type flop dx
report_cell_constraints

```
// const chain  cell  cell_inst gate_inst constraint     pattern scope
// value  name  pos      name     name    properties
// ----- ----- ------ --------- --------- ---------- --------------------
//   DX                          /dff6   (Clock)    Scan Patterns only
//                                       (Dynamic)
//   DX                          /dff5   (Clock)    Scan Patterns only
//                                       (Dynamic)
//   C0    c1    2      sdff1/q          (Static)   Chain + Scan Pattern
```

Related Topics

add_cell_constraints

delete_cell_constraints

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_cell_models

Context: dft -scan, dft -test_points

Mode: setup, analysis

Displays a list of either all cell models or the DFT library models associated with the specified cell type.

Usage

report_cell_models [-All | {-Type *cell_model_type*}]

Description

The report_cell_models command displays the cell models that you either added with the add_cell_models command, or described in the DFT library with the cell_type attribute.

Arguments

- -All

  An optional switch that specifies to display all cell model definitions that you added with the add_cell_models command. This is the default.

- -Type *cell_model_type*

  An optional switch and literal pair that specifies to display a listing of all the cell models of a particular type. The valid cell_model_types are as follows (with the minimum typing characters shown in uppercase):

  INV — A literal that specifies a one-input inverter gate.

  And — A literal that specifies a two-input AND gate.

  Buf — A literal that specifies a one-input buffer gate.

  OR — A literal that specifies a two-input OR gate.

  NAnd — A literal that specifies a two-input NAND gate.

  NOr — A literal that specifies a two-input NOR gate.

  Xor — A literal that specifies a two-input exclusive OR gate.

  Mux — A literal that specifies a 2-1 multiplexer.

  Scancell — A literal that specifies a cell with four input pins (clock, data, scan in, and scan enable) or clocked scan cell with four inputs (clock, data, scan clock, and scan enable).

  DFf — A literal that specifies a D flip-flop with two input pins (clock and data).

  LATCH — A literal that specifies a D latch with two input pins (enable and data).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example displays a list of all added cell models:

```
add_clocks 0 clk
set_test_logic -set on -re on -clock on
set_system_mode analysis
report_scan_elements
add_cell_models and2 -type and
add_cell_models or2 -type or
add_cell_models mux21h -type mux s a b
add_cell_models nor2 -type nor
report_cell_models
insert_test_logic
```

Related Topics

add_cell_models

delete_cell_models

set_test_logic

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_chain_masks

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Reports the list of masked scan chains and their associated unload value.

Usage

report_chain_masks

Description

Reports the list of masked scan chains and their associated unload value.

Arguments

None

Examples

The following example adds several chain masks and then reports the added chain masks:

add_chain_masks chain2 chain4 chain5
report_chain_masks

```
// Masked scan chain Unload value Cell constraints
// ----------------- ------------ ----------------
//      chain2             X              OX
//      chain4             X              OX
//      chain5             X              OX
```

The following example adds several chain masks, specifies load values, and then reports the added chain masks:

add_chain_masks chain2 -load_value 0
add_chain_masks chain4 -load_value 0
add_chain_masks chain5 -load_value 1
report_chain_masks

```
// Masked scan chain Load value Unload value Cell constraints
// ----------------- ---------- ------------ ----------------
//      chain2           0            X              OX
//      chain4           0            X              OX
//      chain5           1            X              OX
```

Related Topics

add_chain_masks

delete_chain_masks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_clock_controller_pins

Context: dft -logic_bist

Mode: setup, analysis

Reports the clock controller pins specified with the set_clock_controller_pins command.

Usage

report_clock_controller_pins [> | >> *file_pathname*]

Description

Reports the clock controller pins specified with the set_clock_controller_pins command.

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

This example reports all clock_controller pins specified so far using the set_clock_controller_pins command.

report_clock_controller_pins

```
//
//  PinType                  Connection
//  -------                  ----------
//  Shift clock              clock_control_i1/SLOW_CLK clock_control_i2/
SLOW_CLK clock_control_i3/SLOW_CLK
//  Scan enable              clock_control_i1/SCAN_EN clock_control_i2/
SCAN_EN clock_control_i3/SCAN_EN
//  LBIST enable            clock_control_i1/LBIST clock_control_i2/LBIST
clock_control_i3/LBIST
//  Capture procedure index  NCP_INDEX_1/A NCP_INDEX_0/A
//
```

Related Topics

set_clock_controller_pins

set_lbist_power_controller_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_clock_controls

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays current configuration information for clock control definitions.

Usage

report_clock_controls

Description

Displays current configuration information for clock control definitions.

For more information, see "Support for Internal Clock Control" in the *Tessent Scan and ATPG User's Manual*.

Arguments

None

Examples

The following example illustrates how clock control definitions are reported.

The following clock control is defined in a test procedure file.

```
clock_control clk =
   source_clock pll_clk;
   atpg_cycle 0=
     condition ctl_dff2/Q 1;
   end;
   atpg_cycle 1=
     condition ctl_dff1/Q 1;
   end;
end;
```

The Report Clock Control command reports the definition as follows:

report_clock_controls

```
CLOCK_CONTROL "/clk (6)" =
SOURCE_CLOCK "/pll_clk (4)";
   ATPG_CYCLE 0 =
   CONDITION "/ctl_dff2/ (56)" 1;
  END;
   ATPG_CYCLE 1 =
   CONDITION "/ctl_dff1/ (55)" 1;
  END;
END;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

set_clock_controls

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_clock_domains

Context: dft -edt, patterns -scan

Mode: analysis

Displays all clocks and their corresponding clock domain numbers.

Usage

report_clock_domains [{-Compatible_clocks [-Details]} | {-Race_points *clk1 clk2*}] [{> | >>} *file_pathname*]

Description

During ATPG, if “set_clock_restriction domain_clock” is set, the tool may simultaneously turn on any subset of compatible clocks. This enables fault effects to be captured by more than one clock in the same cycle without the risk of simulation mismatches due to clock skew. As a result, the test pattern count may be reduced and the fault coverage improved for the same clock sequential depth, compared with when only one clock is turned on in each cycle.

The report_clock_domains command lists defined clocks by groups based on their compatibility. Clocks are compatible (occupy the same clock domain) if simultaneous capture is reliable when using them. Use this command to assess the overall clock compatibilities in your design and to determine if using “set_clock_restriction domain_clock” would be beneficial.

Tip

The set_clock_restriction command enables different analyses. For example, the command report_clock_domains can base its compatibility analysis on same-edge interaction if you first run the "set_clock_restriction -edge_interaction" command. Another analysis is any interaction if you first run the "set_clock_restriction -any_interaction" command. For more information, refer to “set_clock_restriction” on page 2375.

Note

The basic display enables you to quickly evaluate the overall clock compatibility of your design. It is summary information, however, and does not necessarily show the compatibility relationship between all possible clock pairings. That is, two clocks listed with different clock domains are not necessarily incompatible. Use the -Compatible_clocks switch if you need to check the compatibility information for a specific pair of clocks.

Arguments

- -Compatible_clocks

  An optional switch that includes in the display the pair-wise clock compatibility information the tool uses during ATPG. Because two clocks with different clock domain numbers are not necessarily incompatible, this switch enables you to determine precisely whether two clocks are compatible or not.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The tool treats two clocks as compatible if the clocks can be in their "on" state at the same time without causing clock skew problems or race conditions. To make this determination, the tool analyzes the clocking relationship between source and sink memory elements where the source is clocked by one clock and the sink is clocked by a different clock.

Table 6-7 summarizes the conditions under which the tool determines the two clocks to be compatible (C) or incompatible (I).

Table 6-7. Clock Compatibility Summary

Source	Sink			
	LE[1]	TE[2]	AH[3](TE)	AL[4](LE)
LE	I	C	C	I
TE	C	I	I	C
AH (LE)	I	C	C	I
AL (TE)	C	I	I	C

1. LE = Captures on PI clock's leading edge
2. TE = Captures on PI clock's trailing edge
3. AH = Active high, level sensitive element
4. AL = Active low, level sensitive element

- -Details

  An optional switch that displays the same compatibility information provided by the -Compatible_clocks switch, but instead of using an "x" in the display to mark each incompatible clock pair, use as the marker the number of state elements at which the tool determined the pair to be incompatible.

- -Race_points *clk1 clk2*

  An optional switch and pair of strings that lists the instances that cause the two clocks, clk1, and clk2, to be incompatible. The tool includes the gate ID and type of state element (scan, non-scan, or RAM) for each instance listed.

- > *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list or creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list for appending to the contents of *file_pathname*.

Examples

Example 1

The following example displays the defined clocks and their design domains. The numbers in parentheses are the unique gate identification numbers assigned to each gate during the model flattening process.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_system_mode analysis
report_clock_domains

```
// No. | Clock name    Domain
// ----+---------------------
//   1 | '/ck11m' (8)       1
//   2 | '/ck22m' (9)       1
//   3 | '/ck2_75m' (10)    2
//   4 | '/ck5_5m' (11)     2
//   5 | '/nreset' (39)     3
//   6 | '/scan_reset' (63) 4
//
// 6 clocks classified into 4 domains.
// This is only one of the possible clock classifications. To see all
// dependencies of clock pairs, use 'report_clock_domains
// -Compatible_clocks'
```

Example 2

The following example displays the clock domains in example 1 with added information about the possible clock pairs in the design. Notice that although the tool assigned the /scan_reset clock to a domain different from the others, it is actually incompatible only with clocks /ck2_75m, and /nreset. This is indicated by an “x” in the /scan_reset row, above the index number that corresponds to each of those two clocks.

report_clock_domains -compatible_clocks

```
// No. | Clock name    Domain | Clock compatibility
// ----+----------------------+---------------------
//   1 | '/ck11m' (8)        1| .
//   2 | '/ck22m' (9)        1| ..
//   3 | '/ck2_75m' 10)      2| .x.
//   4 | '/ck5_5m' (11)      2| ....
//   5 | '/nreset' (39)      3| .xxx.
//   6 | '/scan_reset' (63)  4| ..x.x.
// --------------------------+---------------------
//                         No.| 123456
// . = Compatible (non-interacting) clock pair
// x = Incompatible  (interacting) clock pair
//
// 6 clocks classified into 4 domains.
// This is one of possible clock classifications.
```

Example 3

The following example displays the same information as the preceding example, but directs the tool to substitute for each “x” the number of state elements at which the tool found the pair to be incompatible.

report_clock_domains -compatible_clocks -details

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// No. | Clock name    Domain | Clock compatibility
// ----+----------------------+---------------------
//   1 | '/ck11m' (8)        1| .
//   2 | '/ck22m' (9)        1| . .
//   3 | '/ck2_75m' (10)     2| . 3 .
//   4 | '/ck5_5m' (11)      2| . . . .
//   5 | '/nreset' 39)       3| . 6 1 6 .
//   6 | '/scan_reset' (63)  4| . . 8 . 4 .
//----------------------------+----------------------
//                          No.| 1 2 3 4 5 6
// . = Compatible (non-interacting) clock pair
// x = Incompatible (interacting) clock pair
//
// 6 clocks classified into 4 domains.
// This is one of possible clock classifications.
```

The report shows there are three state elements that cause clocks /ck2_75m and /ck22m to be incompatible. The next example lists the instance name, gate ID and type of element for each of these elements.

report_clock_domains -race_points /ck2_75m /ck22m

```
//    /ff30/ (57) non-scan
//    /ff31/ (58) non-scan
//    /ff31b/ (58) non-scan
//  3 state elements cause clocks "/ck2_75m" and "/ck22m" to be
//  incompatible.
```

Example 4

This is an example of a clock domain report with a norace synchronous clock group:

report_clock_domains -compatible_clocks

```
//   No. | Clock Name  Domain | Clock Compatibility
//  -----+--------------------+--------------------
//     1 | '/c1' (4)        1 | .
//     2 | '/c2' (5)        2 | x.
//     3 | '/c3' (6)        1 | .n.
//  -------------------------+--------------------
//                       No. | 123
//  . = Compatible (non-interacting) clock pair
//  x = Incompatible (interacting) clock pair
//  n = Norace (synchronous group) specified between compatible clock pair (1)
//  N = Norace (synchronous group) specified between incompatible clock pair (0)
//
//  3 clocks grouped into 2 domains.
//  This is one of possible clock grouping that ATPG may use.
//  Compatibility analysis is based on same-edge clock interaction.
```

Example 5

This is an example of a clock domain report with a race specified between an incompatible clock pair:

report_clock_domains -compatible_clocks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//   No. | Clock Name  Domain | Clock Compatibility
//  -----+--------------------+--------------------
//     1 | '/c1' (4)        1 | .
//     2 | '/c2' (5)        2 | r.
//     3 | '/c3' (6)        1 | ...
//  --------------------------+--------------------
//                        No. | 123
//  . = Compatible (non-interacting) clock pair
//  x = Incompatible (interacting) clock pair
//  R = Race specified between compatible clock pair (0)
//  r = Race specified between incompatible clock pair (1)
//
//  3 clocks grouped into 2 domains.
//  This is one of possible clock grouping that ATPG may use.
//  Compatibility analysis is based on same-edge clock interaction.
```

Example 6

This is an example of a clock domain report including IJTAG and test clocks:

report_clock_domains -compatible_clocks -detail

```
// No. | Clock name        Domain | Clock compatibility
// ----+--------------------------+---------------------
//   1 | '/ijtag_tck' (112)      1| .
//   2 | '/test_clock_u' (120)   1| . .
//   3 | '/ck_out_mux_y1' (128)  1| . . .
//   4 | '/ck_out_mux_y2' (127)  2| . . 1 .
//   5 | '/ck_out_mux_y3' (126)  1| . . . .  .
//   6 | '/ck_out_mux_y4' (125)  1| . . . 44 . .
//   7 | '/ck_out_mux_y5' (124) 33| . . . 7  . 5 .
//--------------------------------+----------------------
//                            No.| 1 2 3 4  5 6 7
//        . = Compatible (non-interacting) clock pair
// <number> = Incompatible clock pair
//
// 7 clocks classified into 3 domains.
// This is one of possible clock classifications.
```

Note

The “Domain” column identifies the clock domains in the design, including clocks added by Tessent. This example shows that the IJTAG clock is not necessarily in its own domain.

Related Topics

add_clocks

add_synchronous_clock_group

delete_clocks

set_clock_restriction

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_clock_gating

Context: dft -scan, dft -test_points

Mode: analysis

Reports specified clock gating instances or modules in the design.

Usage

report_clock_gating ***object_spec***

Description

Reports specified clock gating instances or modules in the design. Only clock gaters with unconnected or tied test enable pins are reported.

The report includes the following information for each clock gater:

- **Test Enable Port** — Specifies the unconnected test enable port that is used to control the clock gater.
- **Shift Control** — Specifies the scan enable signal (or signals) that is used to enable the clock gater during shift. The value is only available after issuing analyze_scan_chains or if there is an enable signal defined for the clock gater with the set_clock_gating_enable command.
- **Capture Control** — If the clock gater drives input or output wrapper cells, the column displays the appropriate mode enable signal name that is ORed together with scan enable to guarantee correct wrapper chain operation. If the clock gater does not drive any input or output wrapper cells, the column shows “Functional”, which means that the clock gater is not enabled during capture, neither in internal nor external mode. The value is available only after issuing the analyze_wrapper_cells and analyze_scan_chains commands.
- **Origin** — Specifies how the tool first learned about the clock gater. The possible values are: User-specified, Cell library, Auto-identified.
- **Status** — Specifies the status of each clock gater. The possible values are:
  - Unconnected — The clock gater gets connected during scan insertion.
  - Hard module — The clock gater is inside a hard module and cannot be connected.
  - Excluded — Explicitly excluded by you with the “set_clock_gating off” command.
  - Non-scan — The clock gater does not drive any scannable cells.
  - Pre-connected — The test enable pin is pre-connected. For these clock gaters the signals listed in the Shift Control column and Capture Control column are combined with the existing connection, in order to enable the clock gater during shift. However, if the column says "Existing connection", the connectivity of the test enable pin remains unchanged.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Always active — The functional enable pin is constrained to hold the clock gater active during shift and capture. The tool reports this status only if the "set_clock_gating_options -fix_when_func_en_always_active" command is set to the default value of "off".
- ICL instrument — The clock gater is inside an ICL scan resource instrument (SRI), and the tool does not modify its test enable connection.

Note

You can use this command to report clock gaters in the -test_points context. However, the test enable signals of clock gaters are not fixed in this context.

Arguments

- ***object_spec***

  A Tcl list of one or more instances or modules or a collection of instances or modules.

Examples

The following example demonstrates the use of the report_clock_gating command:

report_clock_gating

```
report_clock_gating
--------------------------------------------------------------------------------
Clock Gating   Test Enable  Shift     Capture        Origin           Status
Instance       Port         Control   Control
--------------------------------------------------------------------------------
CG1                                                  User-specified   Excluded
CG2            TE           scan_en   int_ltest_en   Auto-identified  Unconnected
CG3            TE                                    Cell library     Non-scan
CG4            TE           scan_en   ext_ltest_en   Cell library     Unconnected
CG5            TE           Existing
                            connection               Cell library     Pre-connected
CG6            TE           scan_en   ext_ltest_en   Cell library     Pre-connected
--------------------------------------------------------------------------------
```

Related Topics

set_clock_gating

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_clocks

Context: all contexts

Mode: setup, analysis

Displays a list of user-defined clocks and SDC-defined clocks.

Usage

report_clocks [{-All | -User | -Sdc} | {*clock_spec* …}] [{> | >>} *file_pathname*]

Description

Displays a list of user-defined clocks and SDC-defined clocks. The report_clocks command can list all user-defined clocks created with the add_clocks command or all SDC-defined clocks created by an SDC file loaded with the read_sdc command or both. You can also request a list of specific clocks by identifiers such as name or clock object.

Arguments

- -All | -User | -Sdc

  A 1-of-3 choice of optional switches that report user-defined clocks or SDC clocks or both. User-defined clocks are created by the add_clocks command and SDC clocks are created by the SDC file loaded with the read_sdc command. The default option is -User. Note that the -Sdc switch is valid only when the flat model is available.

  For information about the user-defined clocks report, refer to Example 3. For information about the SDC clocks report, refer to Example 4.

- *clock_spec* …

  An optional list of clock names, clock label names, clock objects, or collections for the report. To include wildcards in the list, use the get_clocks command.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of file_pathname.

Examples

Example 1

The following example adds two clocks to the clock list and then displays a list of the clocks:

```
SETUP> add_clocks 1 l1
SETUP> add_clocks 0 l2
SETUP> report_clocks
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
User-defined Clocks (2):
============================
Sync and Async Source Clocks
============================
   ----  ---------  -----------  --------
   Name  Off State  Constraints  Internal
   ----  ---------  -----------  --------
    'I1' 1                       No
    'I2' 0                       No
```

Example 2

The following example adds two internal clocks to the clock list and merges them into a new primary input:

SETUP> add_clocks 1 u2/CLK u4/u5/u2/CLK -pseudo_port_name group_clk

```
//  Note: Primary input 'group_clk' is added, merging 2 pins.
```

SETUP> report_clocks

```
User-defined Clock (1):
============================
Sync and Async Source Clocks
============================
 --------- --------- ----------- -------- ------------------  ------------
   Name   Off State Constraints Internal   Other Properties    Pin(s)
 --------- --------- ----------- -------- ------------------  -----------
 group_clk     1                    Yes   Merged internal pin u2/CLK
                                                              u4/u5/u2/CLK
```

Example 3

The following example reports only user-defined clocks:

SETUP> add_clocks 1 u2/CLK u4/u5/u2/CLK -pseudo_port_name group_clk
// Note: Primary input 'group_clk' is added, merging 2 pins.
SETUP> add_clocks 0 I1 -period 50 ns
SETUP> add_clocks 1 I2 -pulse_always
SETUP> add_clocks I3 -pulse_in_capture -label clock_3
SETUP> report_clocks -user

```
User-defined Clocks (4):
==========================
Sync and Async Source Clocks
============================
--------- --------- --------- --------------- -------- --------- ------------------ ------
Name      Label     Off State  Constraints     Internal  Period   Other Properties   Pin(s)
--------- -------- --------- --------------- -------- --------- ------------------ ------
group_clk group_clk    1                        Yes              Merged internal pin u2/CLK
                                                                                    u4/u5/u2/CLK
I1        I1           -     Asynchronous        No    50.00ns
I2        I2           1     Pulse always        No
I3        clock_3      0     Pulse in capture    No
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following explains the columns in the report:

- Name — Lists the name of the clock defined with the -pseudo_port_name switch or by default.
- Label — Lists the label name specified with the -label switch or assigned by default.
- Off State — Lists the off state of the clock as defined by the add_clocks command.
- Constraints — Lists the special constraint of the clock specified with the add_clocks command:
  - Pulse always — The clock is defined with the -pulse_always switch.
  - Asynchronous — The clock is defined with the -period switch.
  - Pulse in capture — The clock is defined with the -pulse_in_capture switch.
- Internal — Indicates whether the clock is defined on an internal pin.
- Period — Lists the clock period defined with the -period switch.
- Other Properties — Lists any additional properties of the clock.
- Pin(s) — Lists one or more internal pins that are connected to the clock.

Example 4

The following example reports only SDC clocks:

> report_clocks -sdc

```
SDC-defined Clocks (3):
------------  ---------- ------------ ---------  ------ ----------------------------------
Type          Add_clocks Create_clock Divide_by   Name    Nodes
------------  ---------- ------------ ---------  ------ ----------------------------------
Source             X            X                clk1x  /clk1 (1)
  Generated                                 2     clk2x  /CG/occicg_clk2x_top_inst/gclk (26)
  Generated                                 4     clk4x  /CG/occicg_clk4x_top_inst/gclk (36)
```

The report groups the generated clocks under the source clock from which they're derived. The following explains the columns in the report:

- Type— Lists the type of SDC clock, which is either a Source clock (from SDC create_clock command) or a Generated clock (from the SDC command create_generated_clock).
- Add_clocks — Indicates if the clock is also defined with the add_clocks command.
- Create_clock — Indicates when a source clock is defined using SDC create_clock command.
- Divide_by — Lists the divide_by ratio of the clock (from the SDC command "create_generated_clock -divide_by").

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Name — Lists the name of the clock.
- Nodes — Lists the target nodes of the clock.

Example 6

The following example adds two internal clocks and then shows the output of the report_clocks command:

```
SETUP> add_clocks 0 clk1_inst/Y clk2_inst/Y clk3_inst/Y -pseudo_port_name clock_1x_int
SETUP> add_clocks 0 clk4_inst/Y clk5_inst/Y clk6_inst/Y -pseudo_port_name clock_2x_int
SETUP> report_clocks
```

```
User-defined Clocks (2):
------------ ---------  -----------  --------  ----------------------  ------
Name         Off State  Constraints  Internal   Other Properties       Pin(s)
------------ ---------  -----------  --------  ----------------------  ------
'clock_2x_int'   0       Yes                     Merged internal pi   clk4_inst/Y
                                                                     clk5_inst/Y
                                                                     clk6_inst/Y
'clock_1x_int'   0       Yes                     Merged internal pin  clk1_inst/Y
                                                                     clk2_inst/Y
                                                                     clk3_inst/Y
```

Example 7

The following example adds and then reports a clock branch:

```
SETUP>  add_clocks u4/u5/u2/CLK -branch -label branch_2
SETUP> report_clocks
```

```
User-defined Clock (1):
Branch Clock
=============
 ---------  ------------  -------
 Off State  Pin(s)        Label
 ---------  ------------  -------
     -      u4/u5/u2/CLK  branch_2
```

Example 8

The following example uses the set_clock_options command to specify a TCK MUX insertion point using the -dft_inject_node switch, introspects the status, and reports the clocks:

```
SETUP> add_clocks 0 clk
SETUP> set_clock_options clk2 -dft_inject_node udff15/CLK
SETUP> set_clock_options clk3 -dft_inject_node udff20/CLK
SETUP> puts [get_name_list [get_clock_option clk2 -dft_inject_node]] udff15/CLK
SETUP> puts [get_name_list [get_clock_option clk3 -dft_inject_node]] udff20/CLK
SETUP> report_clocks
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
User-defined Clocks (3):
========================
    Sync and Async Source Clocks
    ============================
    ------  ---------  ------------  --------  ---------  ---------------
    Name    Off State  Constraints   Internal  Period     DFT Inject Node
    ------  ---------  ------------  --------  ---------  ---------------
    'clk'      0                        No                     'clk'
    'clk2'     -       Asynchronous     No     10.00ns      'udff15/CLK'
    'clk3'     -       Asynchronous     No     10.00ns      'udff20/CLK'
```

Example 9

The following example uses the get_clocks command to report on all clocks whose names start with “tck”.

report_clocks [get_clocks tck*]

```
//       Name :   'tck2'
//      Label :   tck2
//       Type :   sync_source
//  Off State :   0
//   Internal :   Yes
//     Pin(s) :   'block1_I1/mx_I1/U10/Y'
//
//       Name :   'tck'
//      Label :   tck
//       Type :   sync_source
//  Off State :   1
//   Internal :   No
```

Related Topics

add_clocks

analyze_control_signals

delete_clocks

get_clocks

get_clock_option

read_sdc

set_clock_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_compactor_connections

Context: dft -edt, patterns -scan (EDT On), patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays defined scan chain connections for each EDT scan channel output.

Usage

report_compactor_connections [-All_blocks | {-CHANnel *channel_number*} | {-CHAIn *chain_name*}] [{> | >>} *file_pathname*]

Description

Displays defined scan chain connections for each EDT scan channel output. If issued from Setup mode, the report_compactor_connections command lists scan chain connections for EDT scan channel outputs previously defined using the set_compactor_connections command.

If issued from analysis mode, all compactor connections are reported, whether defined by Tessent TestKompress or with the set_compactor_connections command.

You can display the chain connection information for all scan channel outputs, or optionally include the -channel or -chain switch to display the information only for a particular scan chain or channel.

The -All_blocks switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the chain connection information as described above, but only for the EDT block you have specified as the current EDT block. If you include the -All_blocks switch, the command displays information for all the EDT scan channel outputs in every EDT block. You set the current EDT block using either the add_edt_blocks or set_current_edt_block command. “EDT block” is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the “Modular Compressed ATPG” chapter of the *Tessent TestKompress User’s Manual*.

Arguments

- -All_blocks

  Note

  This switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. If you are using a standard (non-modular) Tessent TestKompress flow, you do not need this switch.

  An optional switch that specifies to display all chain connection information for all EDT blocks. Without this switch, the tool displays the information for only the current EDT block, if it is defined. If a current EDT block is not defined, the tool displays the information for all EDT blocks.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-**CHANnel *channel_number*

  An optional switch and positive integer pair that specify to display the scan chain connections for only the scan channel identified by channel_number.

- **-**CHAIn *chain_name*

  An optional switch and string pair that specify to display the scan chain connections for only the scan channel whose compactor group includes chain_name. Use this switch to see which scan channel is associated with a particular chain.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

Assume a design has eight scan chains named “chain1” through “chain8”, whose outputs currently have user-specified connections through the compactor to two EDT scan channels. These user-defined connections are the result of previous set_compactor_connections commands.

The following example displays the compactor connection information for the chain3 scan chain. You can see from the report that chain3 is part of a compactor group that also includes chain4 and chain5; also that the outputs of these three chains are currently specified to be compacted into the output of scan channel number 2.

report_compactor_connections -chain chain3

```
//Reporting user-defined compactor connections:
//
//    Channel     Chains
//    -------     ------
//       2    chain3 - chain5
//
```

Example 2

The following example looks up the connection information for scan channel number 1.

report_compactor_connections -channel 1

```
//Reporting user-defined compactor connections:
//
//    Channel     Chains
//    -------     ------
//       1       chain1, chain2, chain6, chain7, chain8
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

reset_compactor_connections

set_compactor_connections

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_config_data

Context: unspecified, all contexts

Mode: all modes

Reports the content of configuration wrappers.

Usage

report_config_data [***config_object*** [-in_wrapper *wrapper_object_spec*]
[-partition *partition*]] [-show_unspecified]
[-levels *integer* | all] [-indent *integer*]
[-left_indent *left_indent*]

Description

Reports the contents of configuration wrappers in the transcript as it displays inside the file when the configuration data is written using the write_config_data command.

Like with all report commands, you can use the greater than symbol (>) to redirect the output to a file.

Arguments

- ***config_object***

  A required value that specifies the name of a configuration element or a collection containing one configuration element. When config_object is a collection, the -in_wrapper and -partition options cannot be used because the object contains all of the information about the configuration element. In order to use the -in_wrapper option, the config_object value must be a name and the name must be relative to the specified wrapper element. You can use a positional id for any level of hierarchy. You can also replace the id of a leaf name with the position. For example, the path “tmp(1)/<0>/wrp<1>” means “tmp(1)/wrp(abc)/wrp(def)” if “wrp(abc)” is the first element found inside “tmp(1)” and “wrp(def)” is the second instance of a wrapper with a base name of “wrp” inside “tmp(1)/wrp(abc)”.

- -in_wrapper *wrapper_object_spec*

  An optional switch and value pair that specifies the parent wrapper in which the configuration element exists. The wrapper_object_spec value is the name of a wrapper or a collection containing a wrapper element. The names in config_object are relative to the specified parent wrapper.

- -partition *partition*

  An optional switch and value pair that specifies the parent wrapper in which the configuration element exists. When the option is unspecified, the default partition is used, which is where all specifications (DftSpecification and DefaultsSpecification/DftSpecification) are located. You can access the Core wrappers comprising the Tessent Core Description syntax in the tcd partition. Using the partition names meta: and meta:tcd

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

enables you to access the metadata configuration elements that define the default and tcd partitions. Table 6-8 summarizes the available partition names.

Table 6-8. Available Partition for Introspection

Partition Name	Configuration Data Contained
	When the -partition option is unspecified, the Default partition is accessed which is where the DftSpecification and DefaultsSpecification/DftSpecification configuration data is located.
tcd	When the -partition option is specified with the tcd value, the Core wrappers comprising the Tessent Core Description syntax is accessed
meta	When the -partition option is specified with the meta value, the metadata configuration data defining the permitted syntax of the DftSpecification and DefaultsSpecification/ DftSpecification is accessed.
meta:tcd	When the -partition option is specified with the meta:tcd value, the metadata configuration data defining the permitted syntax of the Core wrappers is accessed.

- -show_unspecified

  An optional Boolean switch that specifies to include all properties and wrappers of type Repeatable:single, as described in the documentation for the Wrapper command, in the reported data as opposed to only those that are currently specified. The unspecified properties are shown with their default values.

- -levels *integer* | all

  A property that specifies the number of sub-wrapper levels to display. By default, the “all” value is used, which specifies to display the full content of all sub-wrappers. When an integer N value is specified, the contents of the sub-wrapper that is N levels below the top wrapper being reported is left empty.

- -indent *left_indent*

  An optional switch and integer pair that specifies the indentation to use for the inner wrappers. When this switch is not specified, each inner wrapper is indented two white spaces relative to their parent wrapper. This switch enables you to control the number of white spaces used when indenting a lower wrapper relative to its parent wrapper.

- -left_indent *left_indent*

  An optional switch and integer pair that specifies to add a number of white spaces in front of every row in the report. By default, the outer wrapper of the report has zero white spaces and each subsequent inner wrapper adds two white spaces. When -left_indent is specified, the outer wrapper has N number of white spaces, as specified by *left_indent*, and each inner wrapper adds two white spaces on top of N. This is useful when you want to report the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

contents of a wrapper and insert it inside a “read_config_data -from_string” command as shown in Example 3.

Examples

Example 1

The following example creates some configuration elements and then reports them. The last command uses “> myfile” to output the reported data to a file called *myfile*.

add_config_element tmp(1)

```
{/tmp(1)}
```

add_config_element tmp(1)/p1 -value v1 -type property

```
{/tmp(1)/p1}
```

add_config_element tmp(1)/p2 -value v2 -type property

```
{/tmp(1)/p2}
```

add_config_element tmp(1)/w1 -type wrapper

```
{/tmp(1)/w1}
```

add_config_element tmp(1)/w1/P1 -value v1 -type property

```
{/tmp(1)/w1/P1}
```

add_config_element tmp(1)/w1/P2 -value v2 -type property

```
{/tmp(1)/w1/P2}
```

add_config_element tmp(1)/w2 -copy_from tmp(1)/w1

```
{/tmp(1)/w2}
```

report_config_data

```
tmp(1) {
  p1 : v1;
  p2 : v2;
  w1 {
    P1 : v1;
    P2 : v2;
  }
  w2 {
    P1 : v1;
    P2 : v2;
  }
}
```

report_config_data tmp(1) -level 1

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
tmp(1) {
  p1 : v1;
  p2 : v2;
  w1 {
    // Not shown
  }
  w2 {
    // Not shown
  }
}
```

report_config_data tmp(1) -level 1 > myfile

Example 2

This example shows how config data is reported when it is of type CSV. The read_config_data command is used to parse a CSV file generated from Excel. Note how the format of the data is changed when only reporting the content of the CSV wrapper as opposed to the CSV Wrapper itself. The same feature exists in the write_config_data command such that you can export CSV data back into a spreadsheet application.

add_config_element csv(1)

```
CDV(1)
```

read_config_data mydata.csv -in csv(1)

report_config_data

```
csv(1) {
  "", "", "", "", "";
  a, b, c;
  "a,c", d;
  "", "", "";
}
```

report_config_data csv(1)

```
,,,,
a,b,c
"a,c",d
,,
```

Example 3

The following example uses the -left_indent switch to report the content of a wrapper and insert it inside a “read_config_data -from_string” command. Notice in the second invocation of the report_config_syntax command how the HostScanInterface(ijtag) wrapper is pushed to the right two white spaces because the command included the -left_indent option.

**ANALYSIS> report_config_data **
DftSpecification(blockA,rtl)/IjtagNetwork/HostScanInterface(ijtag)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
HostScanInterface(ijtag) {
  Sib(sti) {
    Attributes {
      tessent_dft_function : scan_tested_instrument_host;
    }
    Sib(mbist) {
    }
  }
}
```

**ANALYSIS> report_config_data **
**DftSpecification(blockA,rtl)/IjtagNetwork/HostScanInterface(ijtag) **
-left_indent 2

```
  HostScanInterface(ijtag) {
    Sib(sti) {
      Attributes {
        tessent_dft_function : scan_tested_instrument_host;
      }
      Sib(mbist) {
      }
    }
  }
```

You can then easily add the content of the reported config data into the “read_config_file -from_string” wrapper to have it automatically added.

**ANALYSIS> read_config_file **
**-in DftSpecification(blockA,rtl)/IjtagNetwork/HostScanInterface(ijtag) **
-from_string

```
  HostScanInterface(ijtag) {
    Sib(sti) {
      Attributes {
        tessent_dft_function : scan_tested_instrument_host;
      }
      Sib(mbist) {
      }
    }
  }
}
```

Related Topics

report_config_syntax

write_config_data

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_config_messages

Context: all contexts

Mode: all modes

Reports messages associated to a configuration element. When the -hierarchical option is used, it also reports the messages associated to elements below the specified wrapper.

Usage

report_config_message ***config_object***
[-hierarchical]
[-all | [-error] [-warning] [-info]]

Description

Reports messages associated to a configuration element. When the -hierarchical option is used, it also reports the messages associated to elements below the specified wrapper.

Arguments

- ***config_object***

  A required value that specifies the name of a configuration element or a collection containing one configuration element.

- -hierarchical

  An optional switch that can only be used when the config_object_spec is a wrapper and specifies that the command should also return elements below the specified wrapper having at least one config message of the specified types.

- -all

  An optional Boolean option that specifies that all message types are to be returned. The use of the -all option is mutually exclusive to the specification of the -error, -warning, and -info options. When none of the four options is specified, -all is assumed.

- -error

  An optional Boolean option that specifies that the messages to be returned are of type error. The use of the -error option is mutually exclusive to the specification of the -all, -warning, and -info options. When none of the -error, -warning, and -info options is specified, -all is assumed.

- -warning

  An optional Boolean option that specifies that the messages to be returned are of type warning. The use of the -error option is mutually exclusive to the specification of the -all, -warning, and -info options. When none of the -error, -warning, or -info options are specified, -all is assumed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -info

  An optional Boolean option that specifies that the messages to be returned are of type info. The use of the -error option is mutually exclusive with the specification of the -all, -warning, and -info options. When none of the -error, -warning, or -info options are specified, -all is assumed.

Examples

The following example adds configuration messages of type error, warning and info on some elements and reports them.

add_config_element tmp(1)

```
{/tmp(1)}
```

add_config_element tmp(1)/p1 -value v1 -type property

```
{/tmp(1)/p1}
```

add_config_element tmp(1)/p2 -value v2 -type property

```
{/tmp(1)/p2}
```

add_config_element tmp(1)/w1 -type wrapper

```
{/tmp(1)/w1}
```

add_config_element tmp(1)/w1/P1 -value v1 -type property

```
{/tmp(1)/w1/P1}
```

add_config_element tmp(1)/w1/P2 -value v2 -type property

```
{/tmp(1)/w1/P2}
```

add_config_element tmp(1)/w2 -copy_from tmp(1)/w1

```
{/tmp(1)/w2}
```

add_config_message "Message1" -config_object tmp(1)/w1 -error

```
{/tmp(1)/w1}
```

add_config_message "Message2" -config_object tmp(1)/w1/P2 -info

```
{/tmp(1)/w1/P2}
```

add_config_message "Message3" -config_object tmp(1)/w1/P2 -warning

```
{/tmp(1)/w1/P2}
```

report_conf_message tmp(1)/w1

```
Report for errors, warnings, infos
  /tmp(1)/w1
    E0) Message1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_conf_message tmp(1)/w1 -hierarchical

```
Report for errors, warnings, infos
  /tmp(1)/w1
    E0) Message1
  /tmp(1)/w1/P2
    W0) Message3
    I0) Message2
```

Related Topics

add_config_message

get_config_messages

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_config_syntax

Context: unspecified, all contexts

Mode: all modes

Reports the legal configuration syntax for a specified configuration object.

Usage

report_config_syntax ***config_or_meta_object_spec***
[-properties_only | -wrappers_only]
[-levels *integer*]

Description

Reports the legal configuration syntax found below the specified config_or_meta_object_spec element.

If you refer to the meta path, you only need to specify the wrapper and property names such as DftSpecification/IjtagNetwork even though the real path name is Wrapper(DftSpecification)/Wrapper(IjtagNetwork). If you have a configuration element such as DftSpecification(corea,rtl)/IjtagNetwork, you can use it too and it looks up the metadata of the element and report its syntax.

Three options limit the amount of syntax displayed to only wrapper, only properties, or to a specified sub-level count.

Arguments

- ***config_or_meta_object_spec***

  A required string that specifies the configuration wrapper for which you want to see the available syntax. If you refer to the meta path, you only need to specify the wrapper and property names such as “DftSpecification/IjtagNetwork” even though the real meta path name is “Wrapper(DftSpecification)/Wrapper(IjtagNetwork)”. If you have a configuration element such as DftSpecification(corea,rtl)/IjtagNetwork, you can use it too and it looks up the metadata of the element and report its syntax.

- -properties_only

  An optional Boolean option that restricts the command to only show the properties at the top level of the specified wrapper.

- -wrappers_only

  An optional Boolean option that restricts the command to only show the wrappers and exclude the properties.

- -level *integer*

  An optional value pair that specifies the number of sub-wrappers to show. When unspecified, all sub-levels are shown.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example uses the report_config_syntax command to show the available syntax, limited to two levels, currently available in the DftSpecification wrapper.

SETUP> report_config_syntax DftSpecification/IjtagNetwork/ -level 1

```
DftSpecification(<module_name>,<id>) {
  IjtagNetwork {
    ImplementationOptions {
      scan_path_retiming : <string> ; // legal  : (latch) flop
    }
    DataInPorts {
      count        : <int>            ; // default: auto
      port_naming  : <port_name>, ... ; // default: ijtag_data_in[%d]
      multiplexing : <boolean>        ; // default: auto
      connection(<range>) : <port_pin_name> ; // repeatable
    }
    DataOutPorts {
      count       : <int>            ; // default: auto
      port_naming : <port_name>, ... ; // default: ijtag_data_out[%d]
      connection(<range>) : <port_pin_name> ; // repeatable
    }
    HostScanInterface(<id>) {
      Interface {
      }
      DesignInstance(<instance_name>) {
      }
      ScanMux(<id>) {
      }
      Sib(<id>) {
      }
      Tdr(<id>) {
      }
      Tap(<id>) {
      }
      TwoPinSerialPort {
      }
      STapPipeline(<id>) {
      }
      STap(<id>) {
      }
      SecondaryHostScanInterface(<id>) {
      }
    }
  }
}
```

Example 2

The second example shows the entire syntax available in the Tdr wrapper.

**report_config_syntax **
DftSpecification/IjtagNetwork/HostScanInterface/Tdr

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(<module_name>,<id>) {
  IjtagNetwork {
    HostScanInterface(<id>) {
      Tdr(<id>) {
        Interface {
          tck            : <port_name>       ; // default: ijtag_tck
          reset          : <port_name>       ; // default: ijtag_reset
          select         : <port_name>       ; // default: ijtag_sel
          shift_en       : <port_name>       ; // default: ijtag_se
          capture_en     : <port_name>       ; // default: ijtag_ce
          update_en      : <port_name>       ; // default: ijtag_ue
          scan_in        : <port_name>       ; // default: ijtag_si
          scan_out       : <port_name>       ; // default: ijtag_so

          reset_polarity : <active_polarity> ; // legal : active_high
(active_low)
        }
        DataInPorts {
          count       : <int>            ; // default: auto
          port_naming : <port_name>, ... ; // default: ijtag_data_in[%d]
          connection(<range>) : <pin_name> ; // repeatable
        }
        DataOutPorts {
          count        : <int>            ; // default: auto
          port_naming  : <port_name>, ... ; // default: ijtag_data_out[%d]
          multiplexing : <boolean>        ; // default: auto
          output_timing(<range>) : <timing_spec> ; // repeatable
          connection(<range>) : <pin_name> ; // repeatable
        }
        parent_instance              : <instance_name>      ;
        leaf_instance_name           : <leaf_instance_name> ;
        keep_active_during_scan_test : <boolean>        ; // default: auto
        length                       : <int>            ; // default: auto
        extra_bits_capture_value     : <enum>       ; // legal : 1 (0) self
        reset_value                  : <binary>     ; // default: auto
        Attributes {
          <attribute> : <string> ; // repeatable
        }
        DecodedSignal(<signal_name>) {
          decode_values : <binary_code>, ... ;
          multiplexing  : <boolean>          ; // default: auto
          output_timing : <timing_spec>; // legal : (auto) unlatched normal
                                         // ijtag_scan_selection
          connection : <port_pin_name> ; // repeatable
          Attributes {
            <attribute> : <string> ; // repeatable
          }
        }
      }
    }
  }
}
```

Related Topics

create_dft_specification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_config_data

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_context

Context: unspecified, all contexts

Mode: all modes

Reports the current context as specified by the set_context command and any inferred subcontexts.

Usage

report_context

Description

Reports the current context as specified by the set_context command and any inferred subcontexts.

This command also provides information about inferred sub-contexts as shown in the examples below.

Arguments

None

Return Values

None

Examples

Example 1

In the following example, the command reports that it ran in uncompressed ATPG mode.

report_context

```
//  Context: patterns -scan -no_rtl
//  Uncompressed ATPG (no EDT) mode active.
```

Example 2

In the following example, the command reports that it is in extraction mode.

report_context

```
// Context: patterns -ijtag -no_rtl
// ICL extraction mode active.
```

Example 3

In the following example, the command reports that it is in compressed ATPG mode.

report_context

```
//  Context: patterns -scan -no_rtl
//  Compressed ATPG (EDT) mode active.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

In the following example, the command reports that it is in Logic BIST mode.

report_context

```
//  Context: patterns -scan -no_rtl
//  Logic BIST mode active.
```

Related Topics

get_context

set_context

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_control_signals

Context: dft -scan

Mode: analysis

Displays the rules checking results for control signals consisting of clocks added with the add_clocks command and pins identified for gating scannable memory elements with test logic.

Usage

report_control_signals {[-All] | {[*pin_pathname*…] [-Clock] [-Set] [-Reset] [-Write] [-Read] [-Tristate_enable]} [-Verbose] [-NOSTABLE_High] [-NOSTABLE_Low]

Description

Displays the rules checking results for control signals consisting of clocks added with the add_clocks command and pins identified for gating scannable memory elements with test logic.

Arguments

- -All

  An optional switch that outputs all control signals to the report. Default setting.

- *pin_pathname*

  An optional, repeatable string that specifies the pathnames of control signals to report on.

- -Clock

  An optional switch that outputs clock control signals in the report.

- -Set

  An optional switch that reports on all set control signals.

- -Reset

  An optional switch that reports on all reset control signals.

- -Write

  An optional switch that reports on all write control signals.

- -Read

  An optional switch that reports on all read control signals.

- -Tristate_enable

  An optional switch that reports on all tristate enable signals.

- -Verbose

  An optional switch that includes all memory elements associated with each control signal in the report.

- -NOSTABLE_High

  An optional switch that disables the report on any stable-high memory elements.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -NOSTABLE_Low

  An optional switch that disables the report on any stable-low memory elements.

Related Topics

add_clocks

delete_clocks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_core_descriptions

Context: all contexts

Mode: all modes

Reports all core description files.

Usage

report_core_descriptions [-all]

Description

Reports all core description files.

The tool uses the information in this file to retarget patterns for instances of the associated core. The report includes the *design_id* value taken from memory or the TSDB container. The core description report is sorted by core name, then the *design_id* value, and finally by mode name.

The *design_id* for TCD instruments is not stored in memory but is included in the instrument's name.

Arguments

- -all

  An optional literal that reports the core descriptions in the open Tessent Shell Data Base (TSDB) and the core descriptions in memory.

Examples

The following example shows the output of the report_core_descriptions command in two modes:

Example 1

This command in setup mode:

report_core_descriptions

```
Name           Mode Name     Type
------------   -----------   -----------
* chip         retargeting   retargeting

Note: * denotes current design
```

Example 2

This command in analysis mode:

report_core_descriptions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Name                Mode                        Instantiations  Chain     Scan
                    Name         Type                           Patterns  Patterns
----------------- ----------- ----------- --------------  --------  --------
* chip              retargeting retargeting
  piccpu_maxlen16 internal     internal     1               8         117

Note: * denotes current design
```

Example 3

In this example, there are two cores inside the opened TSDBs: coreA and coreB. Each core has multiple views specified with mode and design_id values. The report sorts the cores by core name, then by *design_id* value, and finally by the mode name.

```
SETUP> report_core_descriptions -all
Name                          Design ID  Mode Name  Type       In Memory  TCD File
----------------------------  ---------  ---------  ---------  ---------  ----------------------------------------------------------------------------------------
* top                                    unwrapped  unwrapped  Yes
  coreA                       id1        m1         internal   No         tsdbA/logic_test_cores/coreA_id1.logic_test_core/coreA.atpg_mode_m1/coreA_m1.tcd.gz
                              id1        m2         internal   No         tsdbA/logic_test_cores/coreA_id1.logic_test_core/coreA.atpg_mode_m2/coreA_m2.tcd.gz
                              id1        m3         external   No         tsdbA/logic_test_cores/coreA_id1.logic_test_core/coreA.atpg_mode_m3/coreA_m3.tcd.gz
                              id2        m1         internal   No         tsdbA/logic_test_cores/coreA_id2.logic_test_core/coreA.atpg_mode_m1/coreA_m1.tcd.gz
                              id2        m2         internal   No         tsdbA/logic_test_cores/coreA_id2.logic_test_core/coreA.atpg_mode_m2/coreA_m2.tcd.gz
                              id2        m3         external   No         tsdbA/logic_test_cores/coreA_id2.logic_test_core/coreA.atpg_mode_m3/coreA_m3.tcd.gz
  coreA_id1_tessent_edt_blk1                        edt        No         tsdbA/instruments/coreA_id1_edt.instrument/coreA_id1_tessent_edt_blk1.tcd
  coreA_id1_tessent_edt_blk2                        edt        No         tsdbA/instruments/coreA_id1_edt.instrument/coreA_id1_tessent_edt_blk2.tcd
  coreB                       rtl1       bcd        internal   No         tsdbB/logic_test_cores/coreB_rtl1.logic_test_core/coreB.atpg_mode_bcd/coreB_bcd.tcd.gz
                              rtl2       bcd        internal   No         tsdbB/logic_test_cores/coreB_rtl2.logic_test_core/coreB.atpg_mode_bcd/coreB_bcd.tcd.gz
  coreB_id2_tessent_edt                             edt        No         tsdbB/instruments/coreB_id2_edt.instrument/coreB_id2_tessent_edt.tcd

Note: * denotes current design
```

Note

The In Memory column indicates whether or not a TCD was read into memory using the read_core_descriptions command or the add_core_instances command. By default, the report_core_descriptions command only reports TCDs that are in memory. Use the -all switch to report all cores.

Example 4

In this example, a number of TCDs have been loaded into memory for coreA and coreB. Every mode has an associated *design_id* value.

report_core_descriptions

```
Name                            Design ID   Mode Name   Type
-----------------------------   ---------   ---------   ---------
* top                                        unwrapped   unwrapped
  coreA                         id1         m1          internal
                                id1         m2          internal
                                id2         m3          external
  coreA_id1_tessent_edt_blk1                            edt
  coreA_id2_tessent_edt_blk1                            edt
  coreB                         rt12        bcd         internal
  coreB_id2_tessent_edt                                 edt

Note: * denotes current design
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

add_core_instances

delete_core_instances

report_core_instances

write_core_description

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_core_instance_parameters

Context: patterns -scan, patterns -scan_retargeting, patterns -failure_mapping

Mode: all modes

Reports the instrument configuration parameters and their values that were specified when the core instance was added with the add_core_instances commands -parameter_values switch.

Usage

report_core_instance_parameters {**-cores** ***core_names*** | **-modules** ***module_objects*** | **-instances** ***instance_objects*** | **-icl_modules** ***icl_module_objects*** | **-icl_instances** ***icl_instance_objects***} [-silent]

Description

Reports the EDT instrument configuration parameters and their values specified when the core instance was added with the -parameter_values switch of the add_core_instances command. By default, the command reports all parameters when you have no filters defined.

Note

The report_core_instance_parameters command only lists the instruments whose parameters can be modified. Use the report_core_parameters command to show every instrument and the parameters available if an instrument's parameters cannot be modified.

Arguments

- **-cores** ***core_names***

  A required switch and Tcl list that specify the core names of the core descriptions from which all instrument configuration parameters should be reported.

- **-modules** ***module_objects***

  A required switch and string pair that specify a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects. This command reports the instance instrument configuration parameters for each instantiation of those modules in the design.

- **-instances** ***instance_objects***

  A required switch and string pair that specify a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects. This command reports the instance EDT instrument configuration parameters for each instance in the list.

- **-icl_modules** ***icl_module_objects***

  A required switch and string pair that specify a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects. This command reports the instance instrument configuration parameters for each instantiation of those ICL modules in the design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-icl_instances *icl_instance_objects***

  A required switch and string pair that specify a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects. This command reports the ICL instance EDT instrument configuration parameters for each instance in the list.

- -silent

  An optional switch that suppresses warning messages.

Examples

**SETUP> report_core_instance_parameters -core \
{m8051_edt m8051_G_tessent_occ_NX1 m8051_G_tessent_occ_NX2}**

```
Core instance 'm8051_edt_i' (core 'm8051_edt', instrument type 'edt')
---------------------------

Parameter Name           Parameter Value  Legal Override Values  Read Only
-----------------------  ---------------  ----------------------  ---------
edt_bypass                    off                off, on
edt_single_bypass_chain       off                off, on
tessent_chain_masking                                              Yes
tessent_chain_masking         off                off, on
```

Related Topics

add_core_instances

set_core_instance_parameters

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_core_instances

Context: patterns -scan, patterns -scan_retargeting, patterns -failure_mapping

Mode: setup, analysis

Reports the instance bindings created with the add_core_instances command, as well as the pattern count for patterns read in for each core.

Usage

report_core_instances [**-cores *core_names*** | -modules *module_objects* |
-instances *instance_objects* | -icl_modules *icl_module_objects* |
-icl_instances *icl_instance_objects*] [-silent]

Description

Reports the instance bindings created with the add_core_instances command, as well as the pattern count for patterns read in for each core. If this command is invoked in analysis mode, the pattern count for each core is also reported.

You can specify the absolute instance path names or the module or core names. You can either report all core instances that have been added (the default), or filter by modules, instances, or core names.

This command also reports the *design_id* values if they are available. TCD files incoming from TSDBs that have logic_test_core containers created with an empty *design_id* are reported with an empty value. Since it is useful to know the *design_id* for cores, the report includes the *design_id* value taken from memory or the TSDB container. The core description report is sorted by core name, then the *design_id* value, and finally by mode name.

Arguments

- **-cores *core_names***

  A switch and Tcl list that specifies the core names of the core descriptions from which all instance bindings should be reported.

- -modules *module_objects*

  An optional switch and string pair that specifies a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects. This command reports the instance bindings for each instantiation of those modules in the design.

- -instances *instance_objects*

  An optional switch and string pair that specifies a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects. This command reports the instance bindings for each instance in the list.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -icl_modules *icl_module_objects*

  An optional switch and string pair that specifies a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects. This command reports the instance bindings for each instantiation of those ICL modules in the design.

- -icl_instances *icl_instance_objects*

  An optional switch and string pair that specifies a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects. This command reports the instance bindings for each ICL instance in the list.

- -silent

  An optional switch that specifies to suppress warning messages.

Examples

Example 1

The following example reports all existing core instances while in setup mode.

report_core_instances

```
Instance  Core        Mode Name   Type        Module
--------  ---------   ---------   ---------   -----------
core_1    my_core_1   internal    internal    my_module_1
core_2    my_core_2   internal    internal    my_module_2
core_3    my_core_3   internal    internal    my_module_3
```

Example 2

The following example reports all existing core instances while in analysis mode.

report_core_instances

```
Instance  Icl Instance  Core       Mode Name  Type       Module      Chain      Scan
                                                                     Patterns   Patterns
--------  ------------  ---------  ---------  ---------- ----------- --------   --------
core_1    ''            my_core_1  internal   internal   my_module_1 12         10
core_2    ''            my_core_2  internal   internal   my_module_2 12         10
core_3    ''            my_core_3  internal   internal   my_module_3 6          74
```

Example 3

The following example first reports all core instances for modules my_module_1 and my_module_2 while in setup mode; it then reports the same while in analysis mode.

report_core_instances -modules {my_module_1 my_module_2}

```
Instance Core        Mode Name  Type       Module
-------- ---------   ---------  ---------- -------------
core_1   my_core_1   internal   internal   my_module_1
core_2   my_core_2   internal   internal   my_module_2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_system_mode analysis
...
report_core_instances -modules {my_module_1 my_module_2}

```
Instance Icl Instance  Core       Mode Name  Type       Module      Chain      Scan
                                                                    Patterns   Patterns
-------- ------------  ---------  ---------  ---------  ----------- --------   --------
core_1   ''            my_core_1  internal   internal   my_module_1 12         10
core_2   ''            my_core_2  internal   internal   my_module_2 12         10
```

Example 4

The following example reports all EDT IP instances:

report_core_instances

```
Instance     Core       Type  Module
----------   --------   ----  --------
blk1_edt_i   blk1_edt   edt   blk1_edt
blk2_edt_i   blk2_edt   edt   blk2_edt
blk3_edt_i   blk3_edt   edt   blk3_edt
blk4_edt_i   blk4_edt   edt   blk4_edt
```

Example 4

The following example first reports core instances core_1 and core_3 while in setup mode; it then reports the same while in analysis mode.

report_core_instances -instances {core_1 core_3}

```
Instance Core       Mode Name  Type       Module
-------- ---------- ---------  ---------- -----------
core_1   my_core_1  internal   internal   my_module_1
-------- ---------- ---------  ---------- -----------
core_3   my_core_3  internal   internal   my_module_3
```

set_system_mode analysis
...
report_core_instances -instances {core_1 core_3}

```
Instance Core      Mode Name  Type       Module       Chain      Scan
                                                      Patterns   Patterns
-------- --------- ---------  ---------  ------       --------   --------
core_1   my_core_1 internal   internal   my_module_1  12         10
-------- --------- ---------  ---------  ------       --------   --------
core_3   my_core_3 internal   internal   my_module_3  6          74
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 5

In this example, the tool reports the core instances with the *design_id* values.

```
SETUP> report_core_instances
Instance                                        Core                        Design ID  Mode Name  Type      Module
----------------------------------------------  --------------------------  ---------  ---------  --------  -----------
'coreA_inst1'                                   coreA                       id1        m1         internal  'coreA'
|_ 'coreA_inst1/coreA_id1_tessent_edt_blk1_i'   coreA_id1_tessent_edt_blk1                          edt       'coreA_id1_tessent_edt_blk1'
|_ 'coreA_inst1/coreA_id1_tessent_edt_blk2_i'   coreA_id1_tessent_edt_blk2                          edt       'coreA_id1_tessent_edt_blk2'
'coreA_inst2'                                   coreA                       id1        m1         internal  'coreA'
|_ 'coreA_inst2/coreA_id1_tessent_edt_blk1_i'   coreA_id1_tessent_edt_blk1                          edt       'coreA_id1_tessent_edt_blk1'
|_ 'coreA_inst2/coreA_id1_tessent_edt_blk2_i'   coreA_id1_tessent_edt_blk2                          edt       'coreA_id1_tessent_edt_blk2'
'coreB_inst1'                                   coreB                       rtl2       bcd        internal  'coreB'
|_ 'coreB_inst1/coreB_id2_tessent_edt_i'        coreB_id2_tessent_edt                                edt       'coreB_id2_tessent_edt'
```

Related Topics

add_core_instances

delete_core_instances

read_core_descriptions

write_core_description

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_core_parameters

Context: patterns -scan, patterns -scan_retargeting, patterns -failure_mapping

Mode: all modes

Reports all core parameters and their values.

Usage

report_core_parameters {**-cores** ***core_names*** | **-modules** ***module_objects*** | **-instances** ***instance_objects***} [-silent]

Description

Reports all core parameters and their values. By default, the command reports all parameters when you have no filters defined.

Arguments

- **-cores** ***core_names***

  A required switch and Tcl list that reports core parameters for the specified ***core_names***.

- **-modules** ***module_objects***

  A required switch and string pair that reports core parameters for the specified ***module_objects***, which is a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects.

- **-instances** ***instance_objects***

  A required switch and string pair that reports core parameters for the specified ***instance_objects***, which is a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects.

- -silent

  An optional switch that specifies to suppress warning messages.

Examples

SETUP> rep_core_param -cores piccpu_maxlen16_1_edt

```
Core 'piccpu_maxlen16_1_edt' (instrument type 'edt')
----------------------------

Parameter Name              Default Value  Legal Values
-------------------------   -------------  ------------
edt_bypass                       off          off, on
edt_configuration                            high, low
edt_low_power_shift_en            on          off, on
used_input_channels                4            2..4
```

Related Topics

add_core_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_core_instance_parameters

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_current_design

Context: all contexts

Mode: all modes

Generates a report listing design ports that will cause problems during the design flow.

Usage

report_current_design [*object_spec*]
[-columns *column_names*]
[-add_columns *add_columns*]

Description

Use the report_current_design command to list ports that can cause issues in simulation or in the TSDB flow. Most, but not all, of these issues typically arise when port declarations use data types that are not visible to the tool.

Note

The set_current_design command reports a warning about ports that can cause these problems. Use report_current_design to obtain more information about these ports.

Arguments

- *object_spec*

  A list of design objects. Default: the current design.

- -columns *column_names*

  An optional switch and list of strings specifying the data to list. *column_names* can be:

  name — Port name.

  where — The scope of the declaration (local or global).

  flow — The type of problem the port can cause. This is one of SIM (the port is not usable for simulation) or TSDB (problems can occur during the bottom-up flow).

  datatype — A description of the data type.

  By default, all of the above columns are listed in this order.

- -add_columns *add_columns*

  A list of strings specifying additional columns to display following the default columns or those specified by -columns. You can use the same names as those for -columns.

Examples

In this example, set_current_design reports a warning about ports with problematic data type declarations. Next, report_current_design is used to obtain a more comprehensive report about those ports.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
SETUP> set_current_design TOP
//  Note: Design level set to 'physical_block' from previous settings
//  Warning: The current design 'TOP' has ports with 'unpacked struct'/
'enumerate' datatypes declared in Global ($unit) or local scope ('TOP'
scope).
If you try to use the 'read_design' command and elaborate this design
within its parent, you will get an error. Use the 'report_current_design'
command to see the ports declared in global and local scope that will
cause trouble.
SETUP> report_current_design
=====================================================================
The current design 'TOP' has ports with 'unpacked struct'/'enumerate'
datatypes declared in Global ($unit) or local scope ('TOP' scope).
If you try to use the 'read_design' command and elaborate this design
within its parent, you will get an error. The following ports declared in
global and local scope that will cause trouble.
=====================================================================
name            where   flow      datatype
-------------   -----   --------  -----------------------------------
intfc_bisr      global  SIM/TSDB  unpacked struct type defined in global scope
intfc_nbisr     global  SIM/TSDB  unpacked struct type defined in global scope
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_default_ssn_datapath_configuration

Context: all contexts

Mode: all modes

Reports all override settings of an ICL network's default SSN datapath configuration.

Usage

report_default_ssn_datapath_configuration

Description

Displays a report of the override settings of the ICL network.

Refer to the command "set_default_ssn_datapath_configuration" on page 2441 to learn more about the purpose and the use model of the default SSN datapath configuration and how set_default_ssn_datapath_configuration assigns settings to the PDL values of ICL targets to override their default state.

Arguments

None.

Examples

The following example sets two PDL values for two ICL targets and then displays a report of those settings. One of the ICL targets is in a particular design instance.

```
SETUP> set_default_ssn_datapath_configuration -target cfginst.din -value 0b1101xx10
SETUP> set_default_ssn_datapath_configuration -instance tdr_inst1 -target ssnMuxCtrl \
       -value Upper
SETUP> report_default_ssn_datapath_configuration

//  instance   target       value
//  =========  ===========  ==========
//  -          cfginst.din  0b1101xx10
//  tdr_inst1  ssnMuxCtrl   Upper
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_design_partitions

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Reports the partitions in a design.

Usage

report_design_partitions [-all | *partition_name*...]

Description

Reports the partitions created by the add_design_partition command. The report includes the number of gates, number of states, number of RAMs, and the capture power thresholds for switching and rejection.

Arguments

- ***-all***

  An optional switch that reports all partitions. This is the default.

- ***partition_name***

  An optional repeatable string that specifies the names of partitions to report.

Examples

This example shows how to report two partitions named hs_part_1 and hs_part_2:

report_design_partitions hot_spot_partition_1 hot_spot_partition_2

```
                                                Capture Power Thresholds
Name           | # Gates | # States | # RAMs |  Switching   |  Rejection
============== | ======= | ======== | ====== | ============ | ============
hs_part_1        5000       300        0         15%            20%
hs_part_2        300000     1500       2         10%            15%
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_design_sources

Context: all contexts

Mode: all modes

Reports the pathnames or file extensions previously specified with the set_design_sources command.

Usage

report_design_sources

Description

Reports the pathnames or file extensions previously specified with the set_design_sources command.

Arguments

None.

Examples

Example 1

The following example shows a report generated by this command:

set_design_sources -format icl -y ../data/icl_primitives -extension icl

set_design_sources -y ../data/verilog_primitives -extension v

report_design_sources

```
//  ICL search_design_load_path: activated
//  ---------------------------------------------------------
//  format   type  path                          file extensions
//  -------  ----  ----------------------------  ---------------
//  Verilog  dir   '../data/verilog_primitives'  'v v.gz'
//  ICL      dir   '../data/icl_primitives'      'icl icl.gz'
```

Example 2

The following example reports information about all formats:

SETUP> set_design_sources -format tcd_scan -y ../data -extension tcd_scan
SETUP> report_design_sources

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Search design load path settings
// ================================
//  format              value
//  ------------------  -----
//  icl                 on
//  tcd_bscan           on
//  tcd_scan            on
//  tcd_fusebox         on
//  tcd_memory          on
//  tcd_memory_cluster  on
//
// File and directory search path
// ==============================
//  format              type  path     file extensions
//  ------------------  ----  -------  ----------------------------------
//  verilog             dir            v vhd v.gz vhd.gz
//  tcd_scan            dir   ../data  tcd_scan tcd_scan.gz
//  tcd_fusebox         dir            tcd_fbox tcd_fbox.gz
//  tcd_memory          dir            tcd_mem_lib tcd_mem_lib.gz
//  tcd_memory_cluster  dir            tcd_mem_cluster_lib
//                                     tcd_mem_cluster_lib.gz
```

Related Topics

get_design_sources

set_design_sources

set_simulation_library_sources

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_dfm_rules

Context: patterns -scan_diagnosis

Mode: analysis

Returns a tabular report of the DFM rules contained in the layout database.

Usage

report_dfm_rules [-header | -col *columns*...]

Description

Returns a tabular report of the DFM rules contained in the layout database.

By default, all columns are listed. These are: id, rule, layer, and type.

For more information about DFM rules, see "Diagnosis for Design for Manufacturability Analysis."

Arguments

- -header

  An optional switch that returns a list of available column names.

- -col *columns*...

  An optional switch that specifies the column names you want to use for the DFM rule information in the report.

Examples

Example 1

The following example creates a report that contains all available columns.

report_dfm_rules

```
-------------------------------------------------------------
  id   rule                                  layer     type
-------------------------------------------------------------
  1    minimum_space_4_line_route1           route_1   BRIDGE
  2    metal_cross_edge_route1               route_1   OPEN
  3    low_density_min_space_route2          route_2   BRIDGE
  4    half_comb_route3                      route_2   OPEN
  5    metal_cross_edge_route1               route_3   OPEN
  6    corner_to_interior_metal4             route_4   BRIDGE
  7    divergent_line_with_jog_metal5        route_5   BRIDGE
  8    metal_cross_edge_route1               route_5   OPEN
  9    route_5_barbell                       route_5   OPEN
  10   route_2_between_vias                  route_2   OPEN
-------------------------------------------------------------
```

Example 2

The following example creates a report that contains the rule name and the type.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_dfm_rules -col rule type

```
-----------------------------------------
  rule                                type
-----------------------------------------
  minimum_space_4_line_route1       BRIDGE
  metal_cross_edge_route1           OPEN
  low_density_min_space_route2      BRIDGE
  half_comb_route3                  OPEN
  metal_cross_edge_route1           OPEN
  corner_to_interior_metal4         BRIDGE
  divergent_line_with_jog_metal5    BRIDGE
  metal_cross_edge_route1           OPEN
  route_5_barbell                   OPEN
  route_2_between_vias              OPEN
-----------------------------------------
```

Related Topics

get_dfm_rules

delete_dfm

import_dfm

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_dft_cell_selection_mapping

Context: dft

Mode: all

Returns a tabular report of library cell selections, along with their associated power domains.

Usage

report_dft_cell_selection_mappings

Description

Returns a tabular report of library cell selections, along with their associated power domains.

Arguments

None

Related Topics

get_dft_cell_selection_mapping

set_dft_cell_selection_mapping

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_dft_clock_enables

Context: dft, patterns

Mode: all modes

Prerequisites: The current design must be set with the set_current_design command.

Reports the currently defined DFT clock enables.

Usage

report_dft_clock_enables [-post_drc] [-usage func_en | test_en]

Description

Reports the currently defined DFT clock enables.

A DFT clock enable signal is either automatically learned on the ports of clock gating cells, or is specified using the add_dft_clock_enable command on a pin or port of the design.

The -post_drc switch can be specified in analysis mode, when the -usage switch defaults to or is specified as “func_en”, to report the func_en pin that is intercepted to enable the clock gaters that are in the fanin of the memory clocks during memory BIST testing.

Arguments

- -usage func_en | test_en

  An optional switch and value pair that specifies the usage of the clock enable pins and ports to report. When you specify the “func_en” value, the report includes the ports on clock gating cells that have the function attribute set to “func_enable”. The report also includes the ports or pins defined with the add_dft_clock_enables -usage func_en option. When you specify the “test_en” value, the report includes the ports on clock gating cells that have the function attribute set to “test_enable”. The report also includes the ports or pins defined with the add_dft_clock_enables -usage test_en option.

- -post_drc

  An optional Boolean switch that can be specified in analysis mode, when the -usage switch defaults to or is specified as “func_en”, to report the func_en pin that is intercepted to enable the clock gaters that are in the fanin of the memory clocks during memory BIST testing. Without this switch, the command reports the clock enable ports or pins as defined or learned in the cell library or as defined by the “add_dft_clock_enables -usage func_en” command. When the -post_drc switch is specified, the ports are mapped to pins and only those fanning out to memory clock pins are reported.

Examples

The following example uses the add_dft_clock_enables command to identify the en ports of clock gating modules whose module name starts with “my_clock_gate”, and to define the pin on an instance as a clock enable signal with a polarity of 0. The source of the clock enable signal is a special TDR that is scan tested with the rest of the functional logic. The

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_dft_control_points command is used to control the select signal of an existing clock multiplexer; the add_dft_clock_enable command is not used to control that signal because the clock multiplexer must be controlled in scan test mode too.

The report_dft_clock_enables command is used in analysis mode with and without the -post_drc option to show the specified clock enables and those extracted during DRC. The specified clock enables can exist on pins or ports of the current design or on ports of any sub-module. During DRC, the clock enables specified on the ports of sub-modules are mapped to pins. Only the clock enable signals that affect the memory clocking are kept.

```
set_context dft -rtl
# read design command here
set_current_design my_design
set_design_level physical_block
set_dft_specification_requirements -memory_test on

add_clocks clka -period 6.7ns

set mods [get_modules my_clock_gate*]
add_dft_clock_enable [get_ports en -of_modules $mods] -usage func_en
add_dft_clock_enable clock_ctrl/disable -usage func_en -active_polarity 0

add_dft_control_point clock_mux/select


check_design_rules
report_dft_clock_enable
```

```
// Dft clock enable for usage 'func_en'
// ====================================
// ---------------------------------  --------
// Node                               polarity
// ---------------------------------  --------
// Port 'en' of 'my_clock_gate_and'      1
// Port 'en' of 'my_clock_gate_or'       1
// Pin  '/clock_ctrl/disable'            0
```

```
report_dft_clock_enable -post_drc
```

```
// Dft clock enable for usage 'func_en'
// ====================================
// ---------------------------------  --------
// Node                               polarity
// ---------------------------------  --------
// '/clock_ctl/master_clock_gate/en'     1
// '/uart/mem_clk_gate1/en'              1
// '/clock_ctrl/disable'                 0
```

Related Topics

add_dft_clock_enables

delete_dft_clock_enables

read_cell_library

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_dft_clock_muxes

Context: dft, patterns

Mode: all modes

Prerequisites: The current design must be set with the set_current_design command.

Reports the currently defined DFT clock muxes that were previously added using the add_dft_clock_muxes command.

Usage

report_dft_clock_muxes

Description

Reports the currently defined DFT clock muxes that were previously added using the add_dft_clock_mux command.

Arguments

None

Examples

The following example adds three DFT clock muxes and then reports them.

```
add_dft_clock_muxes clkb -test_clock_source mem1/clk  -dft_signal_source_name all_test
register_static_dft_signal_names pll_bypass
add_dft_clock_muxes pll1/vco -test_clock_source pll1/ref \
   -dft_signal_source_name pll_bypass
add_dft_clock_muxes pll1/ref -test_clock_source clkb -dft_signal_source_name alt_ref
report_dft_clock_muxes
```

```
// Dft clock muxes
// ===============
// ----------  -----------------  -------------------
//    Node     Test clock source  Control source name
// ----------  -----------------  -------------------
// 'clkb'      mem1/clk           all_test
// 'pll1/vco'  pll1/ref           pll_bypass
// 'pll1/ref'  clkb               alt_ref
//
```

Related Topics

add_dft_clock_mux

delete_dft_clock_muxes

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_dft_control_points

Context: dft, patterns

Mode: all modes

Prerequisites: The current design must be set with the set_current_design command.

Reports the currently defined DFT control points that were previously added using the add_dft_control_points command.

Usage

report_dft_control_points

Description

Reports the currently defined DFT control points that were previously added using the add_dft_control_points command.

Arguments

None

Examples

Example 1

The following example adds three DFT control points and then reports them.

add_dft_control_point portb -dft_signal_source_name all_test
register_static_dft_signal_names pll_bypass
add_dft_control_point pll1_mux/s -dft_signal_source_name pll_bypass
add_dft_control_point ref_mux/s -dft_signal_source_name alt_ref
report_dft_control_points

```
// DFT Control points
// ===============
// ------------ -------------------
//    Node      Control source name
// ------------ -------------------
// 'portb'      all_test
// 'pll1_mux/s' pll_bypass
// 'ref_mux/s'  alt_ref
//
```

Example 2

If you issue the report_dft_control_points command in analysis mode, the report shows if the added add_dft_control_points -type AsyncSetReset nodes are ignored or reached by the DFT_C9 DRC as follows:

The report shows the following *before* analysis:

```
// '<node>'   async_set_reset   scan_en   User Added
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The report shows the following *after* analysis:

```
// '<node>'   async_set_reset   scan_en   User Added, Ignored
```

when the node was not reached by DFT_C9.

When the node was reached by DFT_C9, then the report shows the following:

```
// '<node>'   async_set_reset   scan_en   User Added, Used by DFT_C9,
Active High
```

-or-

```
// '<node>'   async_set_reset   scan_en   User Added, Used by DFT_C9,
Active Low
```

The reported polarity is the polarity that activates the reset/set at the flops.

The DFT control logic uses scan_en to force the node inactive during shift mode.

Related Topics

add_dft_control_points

delete_dft_control_points

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_dft_modal_connections

A command to report the cumulative effect of the previously invoked add_dft_modal_connections and delete_dft_modal_connections commands.

Usage

report_dft_modal_connections

Description

This command reports the cumulative effect of the previously invoked add_dft_modal_connections and delete_dft_modal_connections commands. The report has four optional sections. If called before any DFT modal connections are specified, the command echoes:

```
SETUP> report_dft_modal_connections
There are no DFT modal connections to report
```

Once there are at least one of input or output DFT modal connections specified, the report includes the “Modes” section, which lists all the used modes and the ports/auxiliary data pins used by each mode. It also includes the “Input connections” or “Output connections” section, or both, showing the destination/source of the port or auxiliary data pins in each mode. Finally, if the specification implies the presence of multiplexing at one or more input data destination nodes, the “Input destinations” section is added and the multiplexing at the destination nodes that include multiplexing are shown. See the example below showing the four sections in the report.

Arguments

None

Examples

The following example shows the four sections of the report representing the effect of the used add_dft_modal_connections commands.

```
set mode edt_mode
add_dft_modal_connections -ports gpio[1:0] \
    -input_data_destination_nodes * \
    -enable_dft_signal $mode -pipeline_stages 2
add_dft_modal_connections -ports gpio[2] \
    -output_data_source_nodes     * \
    -enable_dft_signal $mode -pipeline_stages 3
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set mode retargeting1_mode
add_dft_modal_connections -ports gpio[1:0] \
    -input_data_destination_nodes corea_i1/edt_channels_in[1:0] \
    -enable_dft_signal $mode -pipeline_stages 1
add_dft_modal_connections -ports gpio[2] \
    -output_data_source_nodes     corea_i1/edt_channels_out[0] \
    -enable_dft_signal $mode -pipeline_stages 4

set mode retargeting2_mode
add_dft_modal_connections -ports gpio[1:0] \
    -input_data_destination_nodes corea_i2/edt_channels_in[1:0] \
    -enable_dft_signal $mode -pipeline_stages 2
add_dft_modal_connections -ports gpio[2] \
    -output_data_source_nodes     corea_i2/edt_channels_out[0] \
    -enable_dft_signal $mode -pipeline_stages 5

set mode retargeting3_mode
add_dft_modal_connections -ports gpio[1:0] \
    -input_data_destination_nodes corea_i1/edt_channels_in[1:0] \
    -enable_dft_signal $mode -pipeline_stages 1
add_dft_modal_connections -ports gpio[2] \
    -output_data_source_nodes     corea_i1/edt_channels_out[0] \
    -enable_dft_signal $mode -pipeline_stages 4
add_dft_modal_connections -ports gpio[1:0] \
    -input_data_destination_nodes corea_i2/edt_channels_in[1:0] \
    -enable_dft_signal $mode -pipeline_stages 1
add_dft_modal_connections -ports gpio[3] \
    -output_data_source_nodes     corea_i2/edt_channels_out[0] \
    -enable_dft_signal $mode -pipeline_stages 4

set mode retargeting4_mode
add_dft_modal_connections -ports gpio[1:0] \
    -input_data_destination_nodes corea_i1/edt_channels_in[1:0] \
    -enable_dft_signal $mode -pipeline_stages 1
add_dft_modal_connections -ports gpio[4] \
    -output_data_source_nodes     corea_i1/edt_channels_out[0] \
    -enable_dft_signal $mode -pipeline_stages 4
add_dft_modal_connections -ports gpio[2:3] \
    -input_data_destination_nodes corea_i2/edt_channels_in[1:0] \
    -enable_dft_signal $mode -pipeline_stages 2
add_dft_modal_connections -ports gpio[5] \
    -output_data_source_nodes     corea_i2/edt_channels_out[0] \
    -enable_dft_signal $mode -pipeline_stages 5

SETUP> report_dft_modal_connections
DFT Modal Connections
=====================

Modes
-----
edt_mode           : DFT signal with usage scan_mode(unwrapped)
  Input connections (2):
    Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
    Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
Output connection (1):
    Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
retargeting1_mode : DFT signal with usage scan_mode(retargeting)
  Input connections (2):
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
      Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
      Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
    Output connection (1):
      Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
  retargeting2_mode : DFT signal with usage scan_mode(retargeting)
    Input connections (2):
      Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
      Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
    Output connection (1):
      Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
  retargeting3_mode : DFT signal with usage scan_mode(retargeting)
    Input connections (2):
      Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
      Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
    Output connections (2):
      Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
      Port 'gpio[3]' through auxiliary output pin 'DEF_inst/gpio_3_AuxOut'
  retargeting4_mode : DFT signal with usage scan_mode(retargeting)
    Input connections (4):
      Port 'gpio[1]' through auxiliary output pin 'DEF_inst/gpio_1_AuxIn'
      Port 'gpio[0]' through auxiliary output pin 'DEF_inst/gpio_0_AuxIn'
      Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxIn'
      Port 'gpio[3]' through auxiliary output pin 'DEF_inst/gpio_3_AuxIn'
    Output connections (2):
      Port 'gpio[4]' through auxiliary output pin 'DEF_inst/gpio_4_AuxOut'
      Port 'gpio[5]' through auxiliary output pin 'DEF_inst/gpio_5_AuxOut'
  Input connections
  -----------------
  1) Port 'gpio[1]' through auxiliary input pin 'DEF_inst/gpio_1_AuxIn'
     edt_mode          : P2 (retimed) <Existing destination>
     retargeting1_mode : P1 (retimed) corea_i1/edt_channels_in[1]
     retargeting2_mode : P2 (retimed) corea_i2/edt_channels_in[1]
     retargeting3_mode : P1 (retimed) corea_i1/edt_channels_in[1]
                       : P1 (retimed) corea_i2/edt_channels_in[1]
  2) Port 'gpio[0]' through auxiliary input pin 'DEF_inst/gpio_0_AuxIn'
     edt_mode          : P2 (retimed) <Existing destination>
     retargeting1_mode : P1 (retimed) corea_i1/edt_channels_in[0]
     retargeting2_mode : P2 (retimed) corea_i2/edt_channels_in[0]
     retargeting3_mode : P1 (retimed) corea_i1/edt_channels_in[0]
                       : P1 (retimed) corea_i2/edt_channels_in[0]
  3) Port 'gpio[2]' through auxiliary input pin 'DEF_inst/gpio_2_AuxIn'
     retargeting4_mode : P2 (retimed) corea_i2/edt_channels_in[1]
  4) Port 'gpio[3]' through auxiliary input pin 'DEF_inst/gpio_3_AuxIn'
     retargeting4_mode : P2 (retimed) corea_i2/edt_channels_in[0]

  Output connections
  ------------------
  1) Port 'gpio[2]' through auxiliary output pin 'DEF_inst/gpio_2_AuxOut'
     edt_mode          : P3 (retimed) <Existing source>
     retargeting1_mode : P4 (retimed) corea_i1/edt_channels_out[0]
     retargeting2_mode : P5 (retimed) corea_i2/edt_channels_out[0]
     retargeting3_mode : P4 (retimed) corea_i1/edt_channels_out[0]2) Port
  'gpio[3]' through auxiliary output pin 'DEF_inst/gpio_3_AuxOut'
     retargeting3_mode : P4 (retimed) corea_i2/edt_channels_out[0]
  3) Port 'gpio[4]' through auxiliary output pin 'DEF_inst/gpio_4_AuxOut'
     retargeting4_mode : P4 (retimed) corea_i1/edt_channels_out[0]
  4) Port 'gpio[5]' through auxiliary output pin 'DEF_inst/gpio_5_AuxOut'
     retargeting4_mode : P5 (retimed) corea_i2/edt_channels_out[0]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Input destinations muxing
-------------------------
1) Destination input pin 'corea_i1/edt_channels_in[1]'
   retargeting1_mode : P1 (retimed) DEF_inst/gpio_1_AuxIn
   retargeting3_mode : P1 (retimed) DEF_inst/gpio_1_AuxIn
2) Destination input pin 'corea_i2/edt_channels_in[1]'
   retargeting2_mode : P2 (retimed) DEF_inst/gpio_1_AuxIn
   retargeting3_mode : P1 (retimed) DEF_inst/gpio_1_AuxIn
   retargeting4_mode : P2 (retimed) DEF_inst/gpio_2_AuxIn
3) Destination input pin 'corea_i1/edt_channels_in[0]'
   retargeting1_mode : P1 (retimed) DEF_inst/gpio_0_AuxIn
   retargeting3_mode : P1 (retimed) DEF_inst/gpio_0_AuxIn
4) Destination input pin 'corea_i2/edt_channels_in[0]'
   retargeting2_mode : P2 (retimed) DEF_inst/gpio_0_AuxIn
   retargeting3_mode : P1 (retimed) DEF_inst/gpio_0_AuxIn
   retargeting4_mode : P2 (retimed) DEF_inst/gpio_3_AuxIn
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_dft_signal_names

Context: unspecified, all contexts

Mode: all modes

Reports all registered DFT signal names and identifies their attributes as well as what was the source of registration.

Usage

report_dft_signal_names

Description

Reports all registered DFT signal names and identifies their attributes as well as what was the source of registration. The report separates the static and dynamic DFT signals into two tables. The table for the static DFT signals includes six columns that show the specific information about each signal. The table for the dynamic DFT signals includes two columns that show the specific information about each signal.

Arguments

None

Examples

The command has no arguments, and creates two tables showing the static and the dynamic DFT signals. The static DFT signals table has six columns, which makes the table wider than this page. For an example of the actual report, issue the report_dft_signal_names command in any context. The list of built-in DFT signals shows the complete variation of the report. The actual report generated by the command is not shown here because it is too wide to fit in this narrow page.

report_dft_signal_names

Related Topics

register_static_dft_signal_names

report_dft_signals

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_dft_signals

Context: dft, patterns

Mode: setup, analysis, insertion

Reports all the DFT signals added or to-be-created in the design.

Usage

report_dft_signals

Description

Reports all the DFT signals added or existing in the design. The report is a table with five columns:

- The first column is the name of the DFT signal.
- The second column uses either "To be created" or "Existing" based on whether the DFT signal is scheduled to be created or already exists.
- The third column reports one of the following origin values:
  - **User added** — The DFT signal exists because the add_dft_signals command was issued for the signal in the current insertion pass.
  - **Imported from DesignInfo** — The DFT signal was added on a Port or is a Dynamic DFT signal. During the insertion pass that added the signal, the information was stored in the *.tcd* file located in the dft_inserted_designs directory inside the Tessent Shell Data Base (TSDB) directory and was automatically re-imported when using read_design for the next insertion pass.
  - **ICL attributes** — The signal is a static DFT signal defined as an attributed DataOutPort of an ICL module.
  - **Tool inferred** — The DFT signal was inferred by the tool. You cannot delete the signal with the delete_dft_signals command.
- The fourth and fifth columns report the type and location, respectively.

See "Example 2" on page 231 of the add_dft_signals command description to see the attributes that define static DFT signals.

Arguments

None

Examples

The following example shows the report generated when the command is issued after certain DFT signals were defined using the add_dft_signals command, but prior to issuing the process_dft_specification command. Rerunning the report_dft_signals command after issuing

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

the process_dft_specification command changes the status to “Existing”, and the null locations become the pin on the TDR that sources the DFT signal.

```
add_dft_signals ltest_en int_ltest_en ext_ltest_en
add_dft_signals scan_en test_clock edt_update \
      -source_nodes {my_scan_en my_test_clock my_edt_update}
add_dft_signals edt_clock shift_capture_clock \
      -create_from_other_signals
report_dft_signals
```

```
// Existing and to be created DFT signals
// ======================================
// -------------------  -------------  ----------  --------------  -------------------------
// Name                 Status         Origin      Type            Location
// -------------------  -------------  ----------  --------------  -------------------------
// edt_clock            To be created  User added  ClockGateAnd    here/
//<persistent_cell_prefix>edt_clock
// edt_update           To be created  User added  Port            my_edt_update
// ext_ltest_en         To be created  User added  IJTAG register
// int_ltest_en         To be created  User added  IJTAG register
// ltest_en             To be created  User added  IJTAG register
// scan_en              To be created  User added  Port            my_scan_en
// shift_capture_clock  To be created  User added  ClockGateAnd    here/
//                                                 <persistent_cell_prefix>shift_capture_clock
// test_clock           To be created  User added  Port            my_test_clock
```

Related Topics

add_dft_signals

report_dft_signal_names

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_diagnosis

Context: patterns -scan_diagnosis

Mode: analysis

Writes the scan diagnosis report to standard I/O.

Usage

report_diagnosis [-ENCoded] [-SHORT]

Description

Writes the scan diagnosis report to standard I/O.

Arguments

- -ENCoded

  An optional switch encoding the suspect's pin pathname, net name, and cell name in the diagnosis report.

- -SHORT

  An optional switch that suppresses the display of layout-aware diagnosis information in the diagnosis report.

Related Topics

write_diagnosis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_drc_rules

Context: all contexts

Mode: all modes

Displays either a summary of DRC violations (fails) or violation occurrence messages.

Usage

report_drc_rules [-Fails_summary | -Summary | *rule_id*… | *rule_id-occurence#*… | -All_fails]
[-Verbose] [-SCAn_cells] [-NON_scan_cells]
[{> | >>} *file_pathname*]

C1 Usage:

report_drc_rules **C1** [-EXcluded]

C27 Usage:

report_drc_rules { **C27** | **C27-occurrence#** } [-verbose]

D5 Usage:

report_drc_rules **D5**
[{ [-TYpe {I0 | I1 | IX | T0 | T1 | TX | TLA}…]
[-NOType {I0 | I1 | IX | T0 | T1 | TX | TLA}…]
[-EDge_triggered | -LEvel_sensitive] } | -Summary]
[{> | >>} *file_pathname*]

E5 Usage:

report_drc_rules **E5** [-TIMing_exceptions]

Description

Displays either a summary of DRC violations (fails) or violation occurrence messages. This command can display a report in one of two formats:

- **Summary Report** — Lists for each reported design rule, one line of data per rule, the current number of DRC violations (fails), the violation handling, and a brief description of the rule. When you request a summary of D5 violations, the report lists the number of D5 violations of each type (INIT-0, INIT-1, INIT-X, TIE-0, TIE-1, TIE-X, TLA).
- **Occurrence Report** — Lists one or more violation occurrence messages that give details of specific DRC violations

Table 6-9 provides a summary of the available information displayed and the arguments you use to obtain them. Refer to the Arguments subsection for complete details about the arguments.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 6-9. Available Information Displayed and Arguments

Desired Display	Rules/Occurrences Covered	Argument
Summary report	Design rules that resulted in violations (fails) during DRC	-Fails_summary
	All design rules	-Summary
Occurrence report	Specific rule, specific occurrence	*rule_id-occurence#*
	Specific rule, all occurrences[1]	*rule_id*
	All occurrences	-All_fails

1. Additional D5-specific arguments enable you to further customize the D5 information display.

You can use the set_drc_handling command to change the handling of many of the A (RAM), C (clock), D (data), E (extra), K (EDT), T (trace), and W (timing) rules.

For more information about the design rules, refer to "Design Rule Checking" on page 3167.

Arguments

- -Fails_Summary

  A switch that specifies to display the following for each user-controllable rule that resulted in a violation (fail) during DRC:

  - Rule identification (ID)
  - Number of failures of the rule
  - Current handling status of the rule
  - Brief description of the rule

  This is the default.

  Note

  This switch does not display anything if there are no rule violations or the tool has not yet performed DRC.

- -Summary

  A switch that specifies to display a summary report that varies depending on whether you are reporting on all design rules or just on the D5 rule. (The latter reporting is available only in Tessent FastScan and Tessent TestKompress.)

  When you report on all rules (report_drc_rules -Summary), the following is reported for each user-controllable rule, whether or not it resulted in a violation (fail) during DRC:

  - Rule identification (ID)
  - Number of failures of the rule

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

  - Current handling status of the rule
  - Brief description of the rule

When you report on just the D5 rule (report_drc_rules D5 -Summary) in Tessent FastScan or Tessent TestKompress, the tool displays the number of non-scan memory elements of each type (INIT-0, INIT-1, INIT-X, TIE-0, TIE-1, TIE-X or TLA) that resulted in a D5 violation during DRC. Refer to the description of the -Type switch for the meaning of the types. See also the "D5" section for detailed information about the D5 rule.

- *rule_id*

  A repeatable string that specifies the identification literal (ID) of a particular design rule for which you want to display the violations. For information about the design rules and their ID, refer to the appropriate section:

  - RAM Rules (A Rules)
  - BIST Rules (B Rules)
  - Clock Rules (C Rules)
  - Scan Cell Data Rules (D Rules)
  - Extra Rules (E Rules)
  - EDT Finder Rules (F Rules)
  - Flattening Rules (FN, FP, and FG Rules)
  - General Rules (G Rules)
  - EDT Rules (K Rules)
  - Procedure Rules (P Rules)
  - Scannability Rules (S Rules)
  - Scan Chain Trace Rules (T Rules)
  - Power-Aware Rules (V Rules)
  - Timing Rules (W Rules)
  - ICL Extraction Rules (I Rules)
  - ICL Semantic Rules (ICL Rules)
  - Core Mapping for ATPG and Scan Pattern Retargeting Rules (R Rules)

- *rule_id-occurrence#*

  A repeatable string that specifies the identification literal (ID) of a particular design rule and the violation occurrence for which you want to display the occurrence message. This argument must include the specific design rule ID (*rule_id*), the specific occurrence number of the violation, and the hyphen between them. For example, you can analyze the second

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

violation occurrence of the C3 rule by specifying C3-2. Numbers to occurrences of rule violations are assigned as they are encountered; you cannot change the number assigned to a specific occurrence.

- -All_Fails

  A switch that specifies to display all occurrence messages for all occurrences of rule violations. The displayed information can be lengthy, as it is the same information you would get if you consecutively entered a "report_drc_rules <*rule_id*>" command for each rule that had a violation. Use this switch to output a report of all violation occurrences (most likely to a log file) for later analysis.

- -Verbose

  A switch that specifies to display any additional information available for the specified DRC rule violation. This switch displays the following specific information for these DRC rule violations:

  D12 — The specific cells that failed the DRC check.

- -SCAn_cells

  An optional switch that restricts the report to DRC violations only for scan cells. Use this for DRC checks that apply to both scan and non-scan cells, such as C3, C4, C6, and C7.

- -NON_scan_cells

  An optional switch that restricts the report to DRC violations only for non-scan cells. Use this for DRC checks that apply to both scan and non-scan cells, such as C3, C4, C6, and C7.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

C1-Only Arguments

- **C1**

  A required literal that specifies reporting C1 DRC rule violations.

- -EXcluded

  An optional switch for use only with a C1 violation. Specifying this switch reports the C1 violations that have been excluded from the default C1 list because these C1 violations can be handled by the tool without causing a potential mismatch.

D5-Only Arguments

- -TYpe I0 | I1 |IX | T0 | T1 | TX | TLA

  An optional switch and repeatable literal that displays D5 occurrence messages for only the specified type(s) of non-scan sequential elements. The literal choices for the type of element

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

are as follows (the term you see in occurrence messages for each type is shown in parentheses):

I0 — If the element is at 0 at the beginning of the first capture cycle and may go to any state during capture. (INIT-0)

I1 — If the element is at 1 at the beginning of the first capture cycle and may go to any state during capture. (INIT-1)

IX — If the element's state is unknown at the beginning of the first capture cycle and may go to any state during capture. (INIT-X)

T0 — If the element is always at 0 during capture. (TIE-0)

T1 — If the element is always at 1 during capture. (TIE-1)

TX — If the element is always at an unknown state during capture. (TIE-X)

TLA — If the element is always transparent when its clock is at its off state. (TLA)

Tip

Except for TLAs, you can also direct the tool to display information for only those D5 elements that are edge-triggered or level-sensitive. See the -Edge_triggered and -Level_sensitive switch descriptions for details.

- -NOType I0 | I1 |IX | T0 | T1 | TX | TLA

  An optional switch and repeatable literal that specify not to display occurrence messages for the particular type(s) of D5 violations. See the description of the -Type switch for the meaning of the literal choices.

- -EDge_triggered | -LEvel_sensitive

  Optional switches that specify to display D5 occurrence messages either for edge-triggered or level-sensitive elements only. The default (when neither option is specified) is to display information for both edge-triggered and level-sensitive elements.

E5-Only Arguments

- -TIMing_exceptions

  An optional switch that specifies to report only on E5 violations due to timing exceptions. In order to use this switch, you must first issue a "set_drc_handling -timing_exception_check ON" command in setup mode (see "Example 8" on page 1782).

Examples

Example 1

The following example displays a summary of the rules that resulted in violations during an attempted ATPG run:

report_drc_rules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
C3: #fails=2 handling=warning (clock may capture data affected by its
    captured data)
C5: #fails=1 handling=error (clock is connected to multiple ports of same
    latch)
C7: #fails=19 handling=warning (scan cell capture ability check)
C8: #fails=2 handling=warning (PO connected to a clock line)
C9: #fails=2 handling=warning (PO connected to a clock line gated by scan
    cell that uses same clock)
D5: #fails=23 handling=warning (non-scan memory element)
D6: #fails=22 handling=warning (non-transparent non-scan latches)
D7: #fails=22 handling=warning (stable high edge-triggered clock ports)
```

Example 2

The following example displays the occurrence message for the two C3 violations, and the first C7, and first D7 violation reported in the preceding example:

report_drc_rules c3 c7-1 d7-1

```
// Warning: Clock /sgst failed rule C3 on input 3 of /sp_b_i0.latch
   (1501). (C3-1)// Source of violation: input 3 of /sp_c_i1.slave (1482).
// Warning: Clock /sgst failed rule C3 on input 3 of /sp_c_i0.latch
   (1527). (C3-2)// Source of violation: input 3 of /sp_c_i1.slave (1482).
// Warning: Clock input 5 of /c8.master (1495) cannot capture data
   with single clock on. (C7-1)
// Warning: Flipflop /FF1 (103) has clock port set to stable high. (D7-1)
```

Example 3

The following example changes the handling of data rule 7 (D7) from warning to error and also specifies execution of a full test generation analysis when performing the rules checking for the clock (C) rules:

set_drc_handling d7 error atpg_analysis
set_system_mode analysis

```
//-------------------------------------------------------------
//Begin scan chain identification process, memory elements=8.
//-------------------------------------------------------------
// Reading group test procedure file /user/design/tpf.
// Simulating load/unload procedure in g1 test procedure file.
// Chain = c1 successfully traced with scan_cells = 8.
// Error: Flipflop /FF1 (103) has clock port set to stable high.(D7-1)
// Error: Rules checking unsuccessful, cannot exit SETUP mode.
```

Example 4

The following example displays the occurrence message for the D7 violation:

report_drc_rules d7-1

```
//Error: Flipflop /FF1 (103) has clock port set to stable high. (D7-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 5

The following example displays a summary of the non-scan memory elements that resulted in D5 rule violations:

report_drc_rules d5 -summary

```
//  ---------------------------------------------------------------------
//  44 non-scan memory elements are identified.
//  ---------------------------------------------------------------------
//    9 non-scan memory elements are identified as TIE-0.  (D5)
//    2 non-scan memory elements are identified as TIE-1.  (D5)
//    5 non-scan memory elements are identified as TIE-X.  (D5)
//    3 non-scan memory elements are identified as INIT-0. (D5)
//   11 non-scan memory elements are identified as INIT-1. (D5)
//    4 non-scan memory elements are identified as INIT-X. (D5)
//   10 non-scan memory elements are identified as TLA.    (D5)
//  ---------------------------------------------------------------------
```

Example 6

The following example displays details about the preceding example's D5 violations that occurred on non-scan memory elements the tool identified as type INIT-X:

report_drc_rules d5 -type ix

```
//  Warning: /U1 (78) is a non-scan flip-flop identified as INIT-x.(D5-1)
//  Warning: /U6 (56) is a non-scan flip-flop identified as INIT-x.(D5-7)
//  Warning: /U7 (82) is a non-scan flip-flop identified as INIT-x.(D5-8)
//  Warning: /U9 (11) is a non-scan latch identified as INIT-x.(D5-9)
//  Number of non-scan memory elements reported = 4.
```

Example 7

The following example displays the information for just the INIT-X non-scan latch:

report_drc_rules d5 -type ix -level_sensitive

```
//  Warning: /U9 (11) is a non-scan latch identified as INIT-x.(D5-9)
//  Number of non-scan memory elements reported = 1.
```

You could obtain the same information using the -Notype switch:

report_drc_rules d5 -notype i1 i0 tx t1 t0 tla -level_sensitive

```
//  Warning: /U9 (11) is a non-scan latch identified as INIT-x.(D5-9)
//  Number of non-scan memory elements reported = 1.
```

Example 8

The following example reads timing exception paths and enables the timing exception check for E5. Next, the example enters analysis mode and reports E5 due to timing exceptions.

SETUP> read_sdc my_design.sdc
SETUP> set_drc_handling E5 -timing_exception_check on
SETUP> set_system_mode analysis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
…
//  9 gates may have an observable X-state, 2 of which are due to timing
//  exception paths. (E5)
```

ANALYSIS> report_drc_rules e5 -timing_exceptions

```
//  DFF gate /M3S004FP/VALIDDEVCHIRPK_REG (59193) may have an observable
//  X-state due to timing exception paths. (E5-8)
//  DFF gate /M3S014FP0/TXVALIDH_REG (59220) may have an observable
//  X-state due to timing exception paths. (E5-9)
```

Example 9

The following example shows how to enable the C27 design rule check as a warning. It displays the warning summary message from the system mode transition. The example reports the C27 violation using the -verbose option in analysis mode. The verbose report displays the internal pin generating the glitch and the sequential elements impacted.

SETUP> set_drc_handling C27 warning
SETUP> set_system_mode analysis

```
…
// Warning: There are 2 internal pins that may generate glitches that
propagate to set/reset ports of sequential elements causing simulation
mismatches (C27).
```

ANALYSIS> report_drc_rules C27-1 -verbose

```
// Warning: Internal pin '/inst1/udp/A (567)' may generate a glitch that
propagates to reset ports of 4 sequential elements causing simulation
mismatches, which are
//          '/inst/sdff1 (880)'.
//          '/inst/sdff2 (881)'.
//          '/inst/sdff3 (882)'.
//          '/inst/sdff4 (883)'. (C27-1).
```

Related Topics

set_drc_handling

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_edt_abort_analysis

Context: dft -edt, patterns -scan (EDT On)

Mode: analysis

Reports the results of EDT Aborted fault analysis.

Usage

report_edt_abort_analysis [-block *block_name*]
[-from *from_cube_id*] [-to *to_cube_id*]
[-statistics [-cells_most_commonly_specified *number_of_cells*]
[-shift_positions_most_commonly_specified *number_of_shift_positions*]]
[-cubes]

Description

Reports the results of EDT Aborted (EAB) fault analysis. Without any arguments, the report_edt_abort_analysis command generates a summary report listing the cells most commonly specified and the shift positions most commonly specified. The classification of an EAB test cube is meant to identify the EDT block in which the encoding first failed.

See "EDT Aborted Fault Analysis" in the *Tessent TestKompress User's Manual* for a complete discussion.

Arguments

- -block *block_name*

  An optional switch and string pair that specifies the name of an EDT block on which to perform the analysis.

- -from *from_cube_id*

  An optional switch and positive integer pair that specifies which EAB test cube the tool should analyze first. The default is 1.

  This option is only meaningful when either the -block switch pair is used, or all EAB test cubes are from one EDT block. If multiple EDT blocks contain EAB test cubes and -block is not used, the tool produces an error message.

  Used in conjunction with the complementary -to *to_cube_id* switch and positive integer pair.

- -to *to_cube_id*

  An optional switch and positive integer that specifies where to end the EAB analysis. By default, *to_cube_id* is the total number of collected EAB test cubes.

  This option is only meaningful when either the -block switch pair is used, or all EAB test cubes are from one EDT block. If multiple EDT blocks contain EAB test cubes and -block is not used, the tool produces an error message.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Used in conjunction with the complementary -from *from_cube_id* switch and positive integer pair.

- -statistics

  An optional switch that reports the statistics for the selected block and EAB cubes. This is the default unless you specify the -cubes keyword.

  The tool-generated report shows statistics for each EDT block that has EAB cubes. For each such EDT block, the report contains the following sections:

  - EAB cube size distribution
  - Most specified scan cells by EAB test cubes
  - Most specified shift positions by EAB test cubes

  Optionally, you can control the number of displayed scan cells and shift positions by using the -cells_most_commonly_specified and -shift_positions_most_commonly_specified options.

- -cells_most_commonly_specified *number_of_cells*

  An optional switch and positive integer pair that specifies the number of cells to report. Used in conjunction with the -statistics switch.

- -shift_positions_most_commonly_specified *number_of_shift_positions*

  An optional switch and positive integer pair that specifies the number of shift positions to report. Used in conjunction with the -statistics switch.

- -cubes

  An optional literal that reports the smallest test cubes for the selected block and test cubes.

Examples

Example 1

The following example shows the default report.

In this example, two EDT blocks contain EAB cubes. The first one contains 3 EAB cubes and the second one contains 49. In the first one, none of the EAB cubes in the block are “oversized” (>100 percent).

Note

In this example, the “Gate” column entries are shortened with /gate_path, which represents the following:

```
/edt_top_core2_maxlen90/piccpu_maxlen90_i
```

This is done for presentation purposes.

ANALYSIS> create_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

……

ANALYSIS> report_edt_abort_analysis

```
//EAB test cube analysis results for EDT block 'piccpu_90':
//EDT configurations:
//Number of channels            = 1
//Number of scan chains         = 5
//Number of scan cells          = 217
//Longest scan chain length     = 90
//Approximate encoding capacity = 94
//
//EAB test cube size distribution:
//Specified bits  Specified bits / Capacity (%)  EAB cubes
//--------------  -----------------------------  ---------
//28 - 37         30.00% - 40.00%                2
//37 - 47         40.00% - 50.00%                1
//Total                                          3
//

//Analyzed a total of 3 EAB cubes.
//Top specified scan cells are:
//ID Specified  Gate                                  Chain             Shift   Constraint
//   times                                                              postion
//-- ---------  ----------------------------------  -----             ------- -----------
//1  2(66.67%)  '/gate_path/i/phase_reg_0/Q' (31096) piccpu_90_chain1  16
//2  2(66.67%)  '/gate_path/inst_reg_11/' (31093)    piccpu_90_chain1  13
//3  2(66.67%)  '/gate_path/inst_reg_10/' (31092)    piccpu_90_chain1  12
//4  2(66.67%)  '/gate_path/w_reg_2/' (31076)        piccpu_90_chain5  22
//5  2(66.67%)  '/gate_path/portb_reg_5/' (31006)    piccpu_90_chain5  7
//6  2(66.67%)  '/gate_path/inst_reg_7/' (31089)     piccpu_90_chain1  9
//7  2(66.67%)  '/gate_path/inst_reg_6/' (31088)     piccpu_90_chain1  8
//8  2(66.67%)  '/gate_path/status_reg_6/' (31056)   piccpu_90_chain5  17
//9  2(66.67%)  '/gate_path/inst_reg_3/' (31085)     piccpu_90_chain1  5
//10 2(66.67%)  '/gate_path/inst_reg_2/' (31084)     piccpu_90_chain1  4
//

//Top specified shift positions are:
//Shift position  Specified times  Average specified times per cube
//--------------  ---------------  --------------------------------
//4               4                1
//5               4                1
//12              4                1
//16              4                1
//3               3                1
//22              3                1
//7               3                1
//8               3                1
//9               3                1
//13              3                1
```

In the next report block, 8 out of 49 EAB cubes are beyond the encoding capacity. This means that the specified bits of these 8 cubes are over the encoding capacity of the block. It is more or less expected that they cannot be compressed. The other 41 EAB cubes are within normal cube size range.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

In this example, the “Gate” column entries are shortened with /gate_path, which represents the following:

```
/m8051_5channels/m8051_i
```

This is done for presentation purposes.

```
//  EAB test cube analysis results for EDT block 'm8051_5ch':
//  EDT configurations:
//  Number of channels            = 5
//  Number of scan chains         = 48
//  Number of scan cells          = 498
//  Longest scan chain length     = 11
//  Approximate encoding capacity = 63
//
//  EAB test cube size distribution:
//   Specified bits  Specified bits / Capacity (%)  EAB cubes
//   --------------  ----------------------------  ---------
//   37 - 44         60.00% - 70.00%               1
//   44 - 50         70.00% - 80.00%               8
//   50 - 56         80.00% - 90.00%               16
//   56 - 63         90.00% - 100.00%              16
//    > 63           > 100%                        8
//   Total                                         49
//

//  Analyzed a total of 49 EAB cubes.
//  Top specified scan cells are:
//ID Specified  Gate                                 Chain             Shift   Constraint
//   times                                                             postion
//-- ---------  ----------------------------------  -----             ------- -----------
//1  29(59.18%)  '/gate_path/u14/OPC_reg_4_/' (31458)m8051_5ch_chain26 5
//2  27(55.10%)  '/gate_path/u7/INDIRECT_ADDR_reg_5_/' (31349) m8051_5ch_chain1 4 Condition
//                                                                                 of NCP
//                                                                   user_capture_proc_22
//3  22(44.90%)  '/gate_path/u14/OPC_reg_6_/' (31456) m8051_5ch_chain26 7
//4  20(40.82%)  '/gate_path/u14/OPC_reg_5_/' (31457) m8051_5ch_chain26 6
//5  20(40.82%)  '/gate_path/u14/OPC_reg_1_/' (31461) m8051_5ch_chain26 2
//6  19(38.78%)  '/gate_path/u14/OPC_reg_3_/' (31459) m8051_5ch_chain26 4
//7  19(38.78%)  '/gate_path/u7/INDIRECT_ADDR_reg_3_/' (31351) m8051_5ch_chain1 2  Cell
//                                                                            constraint
//8  18(36.73%)  '/gate_path//u7/INDIRECT_ADDR_reg_4_/' (31350) m8051_5ch_chain1 3  Cell
//                                                                            constraint
//9  18(36.73%)  '/gate_path//u1/Q4_reg/' (31164)  m8051_5ch_chain23  6
//10 17(34.69%)  '/m8051_5channels/m8051_i/u5/TMPDAT_reg_6_/' (31492)  m8051_5ch_chain27 7
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//Top specified shift positions are:
//Shift position  Specified times  Average specified times per cube
//--------------  ---------------  --------------------------------
//7               130              2
//5               119              2
//6               108              2
//4               104              2
//3               83               1
//2               62               1
//8               58               1
//1               20               0
//9               10               0
//0               3                0
//
//  Total analysis time = 0.09 sec.
```

Example 2

The following example shows a more in depth analysis of the second block in Example 1, which had some incompressible EAB blocks.

Note

In this example, the “Gate” column entries are shortened with /gate_path, which represents the following:

```
/m8051_5channels/m8051_i
```

This is done for presentation purposes.

**ANALYSIS> report_edt_abort_analysis -block m8051_5ch -from 1 -to 41 -statistics \
-cells_most_commonly_specified 30 -shift_positions_most_commonly_specified 5**

```
// EAB test cube analysis results for EDT block 'm8051_5ch':
// EDT configurations:
// Number of channels       = 5
// Number of scan chains     = 48
// Number of scan cells      = 498
// Longest scan chain length   = 11
// Approximate encoding capacity = 63
//
//  EAB test cube size distribution:
//   Specified bits  Specified bits / Capacity (%)  EAB cubes
//   --------------  ----------------------------  ---------
//   37 - 44         60.00% - 70.00%               1
//   44 - 50         70.00% - 80.00%               8
//   50 - 56         80.00% - 90.00%               16
//   56 - 63         90.00% - 100.00%              16
//   Total                                         41
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Analyzed a total of 41 EAB cubes.
//  Top specified scan cells are:
//ID Specified  Gate                                  Chain             Shift   Constraint
//   times                                                              postion
//-- ---------  ----------------------------------  -----             ------- -----------
//1  23(56.10%)  '/gate_path/u14/OPC_reg_4_/' (31458) m8051_5ch_chain26 5
//2  21(51.22%)  '/gate_path/u7/INDIRECT_ADDR_reg_5_/' (31349) m8051_5ch_chain1 4 Condition
//                                                                                  of NCP
//                                                                          user_capture_proc_22
//3  19(46.34%)  '/gate_path/u14/OPC_reg_6_/' (31456) m8051_5ch_chain26 7
//4  18(43.90%)  '/gate_path/u7/INDIRECT_ADDR_reg_3_/' (31351)  m8051_5ch_chain1 2  Cell
//                                                                                  constraint
//5  18(43.90%)  '/gate_path/u14/OPC_reg_1_/' (31461) m8051_5ch_chain26  2
//6  17(41.46%)  '/gate_path/u14/OPC_reg_3_/' (31459) m8051_5ch_chain26  4
//7  17(41.46%)  '/gate_path/u14/OPC_reg_5_/' (31457) m8051_5ch_chain26  6
//8  16(39.02%)  '/gate_path/u15/u10_DAT_reg_3_/' (31265)  m8051_5ch_chain40  5  ATPG
//                                                                          constraint on
//                                                     '//gate_path/u15/u10_DAT_reg_3_/Q' (1457)
//9  16(39.02%)  '/gate_path/u7/INDIRECT_ADDR_reg_4_/' (31350)  m8051_5ch_chain1  3  Cell
//                                                                                  constraint
//10  15(36.59%)  '/gate_path/u5/TMPDAT_reg_6_/' (31492) m8051_5ch_chain27 7
//11  14(34.15%)  '/gate_path/u1/Q4_reg/' (31164) m8051_5ch_chain23  6
//12  14(34.15%)  '/gate_path/u14/OPC_reg_2_/' (31460) m8051_5ch_chain26  3
//13  14(34.15%)  '/gate_path/u4/ACLDAT_reg_3_/' (31486)  m8051_5ch_chain32  5
//14  14(34.15%)  '/gate_path/u5/TMPDAT_reg_3_/' (31495)  m8051_5ch_chain27  4
//15  13(31.71%)  '/gate_path/u5/TMPDAT_reg_4_/' (31494)  m8051_5ch_chain27  5
//16  12(29.27%)  '/gate_path/u14/OPC_reg_7_/' (31455)  m8051_5ch_chain26  8
//17  12(29.27%)  '/gate_path/u1/Q5_reg/' (31158)       m8051_5ch_chain23 7
//18  12(29.27%)  '/gate_path/u15/u10_DAT_reg_4_/' (31266)  m8051_5ch_chain40 4  ATPG
//                                                                          constraint on
//                                                      '/gate_path/u15/u10_DAT_reg_4_/Q' (1458)
//19  11(26.83%)  '/gate_path/u4/L_ACCDAT_reg_7_/' (31464)  m8051_5ch_chain30  7
//20  11(26.83%)  '/gate_path/u15/u10_DAT_reg_5_/' (31267)  m8051_5ch_chain40  3 ATPG
//                                                                          constraint on
//                                                                  '/gate_path/u15/U314/Z' (10147)
//21  10(24.39%)  '/gate_path/u4/L_ACCDAT_reg_6_/' (31465)  m8051_5ch_chain30  6
//22  10(24.39%)  '/gate_path/u7/INDIRECT_ADDR_reg_7_/' (31347)  m8051_5ch_chain1  6 Cell
//                                                                                  constraint
//23  10(24.39%)  '/gate_path/u5/TMPDAT_reg_7_/' (31491)  m8051_5ch_chain27 8
//24  10(24.39%)  '/gate_path/u5/TMPDAT_reg_5_/' (31493)  m8051_5ch_chain27  6
//25  9(21.95%)  '/gate_path/u4/L_ACCDAT_reg_5_/' (31466)  m8051_5ch_chain30  5
//26  9(21.95%)  '/gate_path/u4/L_ACCDAT_reg_4_/' (31467)  m8051_5ch_chain30  4
//27  9(21.95%)  '/gate_path/u15/u10_DAT_reg_1_/' (31263)  m8051_5ch_chain40  7  ATPG
//                                                                          constraint on
//                                                      '/gate_path/u15/u10_DAT_reg_1_/Q' (1455)
//28  9(21.95%)  '/gate_path/u5/TMPDAT_reg_2_/' (31496)  m8051_5ch_chain27 3
//29  8(19.51%)  '/gate_path/u7/L_FA_reg_4_/'(31343)  m8051_5ch_chain5  7
//30  8(19.51%)  '/gate_path/u7/INDIRECT_ADDR_reg_6_/' (31348)  m8051_5ch_chain1  5  Cell
//                                                                                  constraint
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//Top specified shift positions are:
//Shift position  Specified times  Average specified times per cube
//--------------  ---------------  --------------------------------
//7               106              2
//5               96               2
//6               91               2
//4               87               2
//3               68               1
//
//  Total analysis time = 0.00 sec.
```

The preceding example focuses on the cubes whose size is below or near encoding capacity. As noted earlier, EAB cubes are sorted in ascending order of their sizes in the failing block. The tool listed 30 top specified scan cells. For each scan cell, the tool reports its specified times among the selected EAB cubes, gate name and id, scan chain, shift position counted from scan input, block name, and any known constraint on the gate.

The constraint column shows that a number of top specified bits have constraints on them. For example, the fourth scan cell has a cell constraint, the second one is a condition bit of an NCP, and so on. This column may include the following constraint types:

- **Cell Constraint** — A C0/C1 cell constraint is on this scan cell.
- **ATPG Constraint** — The scan cell's value is determined due to some added ATPG constraints. ATPG constraints may not necessarily be applied directly to the cell. For example, if an AND gate is constrained to 1, both of its inputs must be 1. If one of the inputs is a scan cell, it has this constraint.
- **Condition Bit of NCP** — This scan cell must load a predetermined value for the NCP.
- **Condition Bit of a Clock Under Control** — The bit is part of the clock control definition of an internal clock.
- **Equivalent Scan Cell** — The scan cell is equivalent to another scan cell that is constrained.
- **Design Constraint** — The scan cell does not have a "hard" constraint. However, when you try to satisfy all the ATPG constraints and prevent all the bus contentions in the design, this scan cell is part of the solution. If this type of scan cell is in the top list, the encoding problem may come from buses or ATPG constraints in the design.

Example 3

The following example shows a design that suffers from excessive EAB faults due to clock control condition bits.

Note

In this example, the "Gate" column entries are shortened with /gate_path, which represents the following:

```
/core_0/dig_core_0/wrap_0/test_clock_ctrl_test_clk_bp_0
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This is done for presentation purposes.

ANALYSIS> report_edt_abort_analysis

```
//  EAB test cube analysis results:
//  EDT configurations:
//  Number of channels            = 10
//  Number of scan chains         = 800
//  Number of scan cells          = 90766
//  Longest scan chain length     = 183
//  Approximate encoding capacity = 1387
//
//  EAB test cube size distribution:
//   Specified bits  Specified bits / Capacity (%)  EAB cubes
//   --------------  -----------------------------  ---------
//   277 - 416       20.00% - 30.00%                81
//    > 1387          > 100%                        19
//   Total                                          100
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Analyzed a total of 100 EAB cubes.
//  Top specified scan cells are:
//   ID  Specified times  Gate
//ID Specified  Gate                                  Chain             Shift   Constraint
//   times                                                              postion
//-- ---------  ----------------------------------  -----             ------- -----------
//1  81(81.00%) '/gate_path/ShiftReg/FF_reg[3]/' (4622541)  chain772  0       Condition of
//                                                                        Clock Control for
//                                                                  '/hsp_test_clk_bp_int'(117)
//2  80(80.00%) '/gate_path/ShiftReg/FF_reg[2]/' (4595006)  chain769   1       Condition of
//                                                                        Clock Control for
//                                                                     '/test_clk_bp_int' (112)
//3  79(79.00%) '/gate_path/ShiftReg/FF_reg[1]/' (4595005)  chain769  2  Condition of Clock
//                                                                              Control for
//                                                                     '/test_clk_bp_int' (112)
//4  77(77.00%) '/gate_path/ShiftReg/FF_reg[0]/' (4597960)   chain775 3   Condition of Clock
//                                                                               Control for
//                                                                     '/test_clk_bp_int' (110)
//5  75(75.00%) '/gate_path/ShiftReg/FF_reg[0]/' (4542701)  chain774  3  ATPG constraint on
//                                                 '/gate_path/ShiftReg/FF_reg[0]/Q' (273991)
//6  74(74.00%) '/gate_path/ShiftReg/FF_reg[2]/' (4598252)  chain770  1  Condition of Clock
//                                                                              Control for
//                                                                   '/test_clk_bp_int' (111)
//7  65(65.00%) '/gate_path/ShiftReg/FF_reg[3]/' (4599800)  chain767  0  Condition of Clock
//                                                                               Control for
//                                                                     '/test_clk_bp_int' (109)
//8  63(63.00%) '/gate_path/ShiftReg/FF_reg[1]/' (4542702)  chain774 2    ATPG constraint on
//                                                 '/gate_path/ShiftReg/FF_reg[1]/Q' (273992)
//9  63(63.00%) '/gate_path/ShiftReg/FF_reg[0]/' (4623366)  chain765 3  Condition of Clock
//                                                                               Control for
//                                                                  '/p_test_clk_bp_int' (108)
//10  60(60.00%) '/gate_path/ShiftReg/FF_reg[1]/' (4598251)  chain770 2   Condition of Clock
//                                                                               Control for
//                                                                     '/test_clk_bp_int' (111)
//
//  Top specified shift positions are:
//   Shift position  Specified times  Average specified times per cube
//   --------------  ---------------  --------------------------------
//   3               592              5
//   0               511              5
//   1               491              4
//   2               425              4
//   4               22               0
//   5               17               0
//   15              16               0
//   7               16               0
//   16              15               0
//   6               15               0
//
//  Total analysis time = 2.24 sec.
```

Example 4

Another usage of the command is to inspect specific test cubes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

In this example, the “Gate” column entries are shortened with /gate_path, which represents the following:

```
/m8051_5channels/m8051_i
```

This is done for presentation purposes.

ATPG> report_edt_abort_analysis -block m8051_5ch -from 1 -to 3 -cubes

```
//EAB test cube #1:
//Gate                                          Value Chain               Shift  Constraint
                                                                          pos
                                                                          (from sci)
//---------------------------------------------- ----- ---------------- ---------- ---------
//’/gate_path/u7/INDIRECT_ADDR_reg_4_/' (31350) 1     m8051_5ch_chain1    3        Cell
//                                                                          constraint
//’/gate_path/u14/OPC_reg_2_/' (31460)          0     m8051_5ch_chain26   3
//’/gate_path/u14/OPC_reg_4_/' (31458)          1     m8051_5ch_chain26   5
//’/gate_path/u4/ACLDAT_reg_6_/' (31483)        1     m8051_5ch_chain32   8
//’/gate_path/u5/TMPDAT_reg_6_/' (31492)        1     m8051_5ch_chain27   7
//’/gate_path/u7/L_FA_reg_4_/' (31343)          1     m8051_5ch_chain5    7
//’/gate_path/u4/L_ACCDAT_reg_4_/' (31467)      0     m8051_5ch_chain30   4
//’/gate_path/u4/L_ACCDAT_reg_7_/' (31464)      0     m8051_5ch_chain30   7
// Total bits specified = 8
//

// EAB test cube #2:
//Gate                                           Value  Chain              Shift Constraint
//                                                                          pos
//                                                                         (from
//                                                                          sci)
//---------------------------------------------  -----  -----              ----- ----------
//’/gate_path/u15/u10_DAT_reg_4_/' (31266)       0      m8051_5ch_chain40  4   ATPG
//                                                                            constraint on
//                                                 ’/gate_path/u15/u10_DAT_reg_4_/Q' (1458)
//’/gate_path/u15/u10_DAT_reg_6_/'(31268)        0      m8051_5ch_chain40  2   ATPG
//                                                                            constraint on
//                                                 ’/gate_path/u15/u10_DAT_reg_6_/Q' (1460)
//’/gate_path/u15/u10_DAT_reg_5_/' (31267)       1      m8051_5ch_chain40  3   ATPG
//                                                                            constraint on
//                                                     ’/gate_path/u15/U314/Z' (10147)
//’/gate_path/u7/INDIRECT_ADDR_reg_2_/' (31352)  1      m8051_5ch_chain1   1     Cell
//                                                                                constraint
//’/gate_path/u4/L_ACCDAT_reg_3_/' (31468)       1      m8051_5ch_chain30  3
//’/gate_path/u14/OPC_reg_0_/' (31462)           0      m8051_5ch_chain26  1
//’/gate_path/u14/OPC_reg_4_/' (31458)           1      m8051_5ch_chain26  5
//’/gate_path/u5/TMPDAT_reg_2_/' (31496)         1      m8051_5ch_chain27  3
//’/gate_path/u4/ACLDAT_reg_2_/' (31487)         0      m8051_5ch_chain32  4
//  Total bits specified = 9
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//EAB test cube #3:
//Gate                                          Value  Chain              Shift  Constraint
//                                                                        pos
//                                                                       (from
//                                                                        sci)
//-------------------------------------------  -----  ----------------  -----  ----------
//'/gate_path/u15/u10_DAT_reg_3_/' (31265)      0      m8051_5ch_chain40  5      ATPG
//                                                                               constraint
//                                                                              on
//                                                 '/gate_path/u15/u10_DAT_reg_3_/Q' (1457)
//'/gate_path/u7/INDIRECT_ADDR_reg_4_/' (31350) 1      m8051_5ch_chain1   3     Cell
//                                                                              constraint
//'/gate_path/u14/OPC_reg_2_/' (31460)          0     m8051_5ch_chain26   3
//'/gate_path/u14/OPC_reg_4_/' (31458)          1     m8051_5ch_chain26   5
//'/gate_path/u14/OPC_reg_5_/' (31457)          0     m8051_5ch_chain26   6
//'/gate_path/u5/TMPDAT_reg_6_/' (31492)        1     m8051_5ch_chain27   7
//'/gate_path/u5/TMPDAT_reg_5_/' (31493)        0     m8051_5ch_chain27   6
//'/gate_path/u4/L_ACCDAT_reg_5_/' (31466)      1     m8051_5ch_chain30   5
//'/gate_path/u1/Q4_reg/' (31164)               0     m8051_5ch_chain23   6
//'/gate_path/u4/L_ACCDAT_reg_6_/' (31465)      0     m8051_5ch_chain30   6
//'/gate_path/u4/L_ACCDAT_reg_4_/' (31467)      1     m8051_5ch_chain30   4
//  Total bits specified = 11
//
//  Total analysis time = 0.00 sec.
```

The test cubes are represented in a similar way as the cells in the top specified cell list. A little different is that the bit in an EAB cube has a value field. We can see that the number of EAB cubes is pretty small. For example, the first reported EAB cube contains only 8 bits. In certain situations such as linear dependency, it may not be possible to encode these 8 bits even though the encoding capacity of the block is over 60. If detection of a fault requires specification of these 8 bits, the fault may be classified as EAB.

Example 5

When multiple EDT blocks share the same EDT channel input pins, analysis of EAB cubes must consider such blocks together.

Note

In this example, the “Gate” column entries are shortened with /gate, which represents the following:

```
/EngineX4_i
```

This is done for presentation purposes.

The generated report displays such blocks as a composite EDT block named COMPOSITE_BLOCK_*N*. The following example shows EAB analysis for 6 EDT blocks that share EDT input channels and are reported as COMPOSITE_BLOCK_0 and another 2 blocks reported as COMPOSITE_BLOCK_1:

ANALYSIS> report_edt_abort_analysis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  EAB test cube analysis results for composite EDT block 'COMPOSITE_BLOCK_0':
//  This composite EDT block contains the following EDT blocks:
//  blk1
//  blk2
//  blk7
//  blk8
//  blk3
//  blk4
//  EDT configurations:
//  Number of channels            = 13
//  Number of scan chains         = 600
//  Number of scan cells          = 18899
//  Longest scan chain length     = 32
//  Approximate encoding capacity = 310
//
//  EAB test cube size distribution:
//   Specified bits  Specified bits / Capacity (%)  EAB cubes
//   --------------  ----------------------------  ---------
//   31 - 62         10.00% - 20.00%               3
//   62 - 93         20.00% - 30.00%               35
//   93 - 124        30.00% - 40.00%               5
//   124 - 155       40.00% - 50.00%               3
//   Total                                         46
//

//  Analyzed a total of 46 EAB cubes.
//  Top specified scan cells are:
//ID Specified  Gate                                                        Chain Shift
//   times                                                                        position
//-- ---------  ------------------------------------------------------------  ----- --------
//1  4 (8.70%)  '/gate/Engine_Instance_3/SE_EX0/PX_COO0_reg[29]5/' (574905)  387   30
//2  4 (8.70%)  '/gate/Engine_Instance_4/SE_PE0/V2Y_reg[8]/' (584285)        81    26
//3  3 (6.52%)  '/gate/Engine_Instance_3/SE_EX0/PX_COO0_reg[0]5/' (574913)   387   22
//4  3 (6.52%)  '/gate/Engine_Instance_4/SE_PE0/V2Y_reg[22]/' (582973)       62    28
//5  3 (6.52%)  '/gate/Engine_Instance_2/SE_EX0/COO_REG_OUT_SEL_reg/' (568317) 628 17
//6  3 (6.52%)  '/gate/Engine_Instance_3/SE_EX0/PX_COO0_reg[2]5/' (574906)   387   29
//7  3 (6.52%)  '/gate/Engine_Instance_3/SE_EX0/PX_COO0_reg[22]5/' (574907)  387   28
//8  3 (6.52%)  '/gate/Engine_Instance_4/SE_PE0/V1Y_reg[1]/' (583376)        68    26
//9  3 (6.52%)  '/gate/Engine_Instance_4/SE_PE0/V1Y_reg[0]/' (584345)        82    26
//10 3(6.52%)   '/gate/Engine_Instance_3/SE_FMUL00/OP_IN_REG_SEL_reg[4]/Q' (579371)  366  3
//
//  Top specified shift positions are:
//   Shift position  Specified times  Average specified times per cube
//   --------------  ---------------  --------------------------------
//   26              73               1
//   28              54               1
//   27              48               1
//   21              38               0
//   30              36               0
//   18              35               0
//   17              34               0
//   25              33               0
//   23              32               0
//   20              31               0
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  EAB test cube analysis results for composite EDT block 'COMPOSITE_BLOCK_1':
//  This composite EDT block contains the following EDT blocks:
//  blk5
//  blk6
//  EDT configurations:
//  Number of channels            = 3
//  Number of scan chains         = 200
//  Number of scan cells          = 6253
//  Longest scan chain length     = 32
//  Approximate encoding capacity = 89
//
//  EAB test cube size distribution:
//   Specified bits  Specified bits / Capacity (%)  EAB cubes
//   --------------  -----------------------------  ---------
//   35 - 44         40.00% - 50.00%               1
//   71 - 80         80.00% - 90.00%               1
//   80 - 89         90.00% - 100.00%              1
//    > 89            > 100%                       6
//   Total                                          9
//

//Analyzed a total of 9 EAB cubes.
//Top specified scan cells are:
//ID Specified  Gate                                                        Chain Shift
//   times                                                                        position
//-- ---------  ------------------------------------------------------------ ----- --------
//1  3(33.33%)  '/gate/Engine_Instance_3/SE_FMUL00/Z_reg[70]1/Q' (579854)    437   10
//2  3(33.33%)  '/gate/Engine_Instance_3/SE_FMUL00/Z_reg[97]1/Q' (579826)    436   7
//3  2(22.22%)  '/gate/Engine_Instance_2/SE_PE0/ZSS_P_reg[10]3/Q' (571404)   505   26
//4  2(22.22%)  '/gate/Engine_Instance_3/SE_FMUL00/Z_reg[33]1/Q' (579889)    438   7
//5  2(22.22%)  '/gate/Engine_Instance_3/SE_FMUL00/Z_reg[28]6/Q' (579885)    438   11
//6  2(22.22%)  '/gate/Engine_Instance_2/SE_PE0/N1523_reg/Q' (570880)        488   15
//7  2(22.22%)  '/gate/Engine_Instance_3/SE_FMUL00/Z_reg[57]1/Q' (579839)    437   25
//8  2(22.22%)  '/gate/Engine_Instance_3/SE_FMUL00/Z_reg[25]6/Q' (579882)    438   14
//9  2(22.22%)  '/gate/Engine_Instance_3/SE_FMUL00/Z_reg[31]6/Q' (579881)    438   15
//10 2(22.22%)  '/gate/Engine_Instance_3/SE_FMUL00/Z_reg[6]6/Q' (579880)     438   16
//
//  Top specified shift positions are:
//   Shift position  Specified times  Average specified times per cube
//   --------------  ---------------  --------------------------------
//   26              17               1
//   11              12               1
//   28              11               1
//   7               9                1
//   16              9                1
//   25              8                0
//   10              8                0
//   21              8                0
//   24              7                0
//   15              7                0
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  EAB test cube analysis results for EDT block 'blk9':
//  EDT configurations:
//  Number of channels            = 1
//  Number of scan chains         = 100
//  Number of scan cells          = 3084
//  Longest scan chain length     = 32
//  Approximate encoding capacity = 42
//
//  EAB test cube size distribution:
//   Specified bits  Specified bits / Capacity (%)  EAB cubes
//   --------------  -----------------------------  ---------
//   33 - 37         80.00% - 90.00%               3
//   37 - 42         90.00% - 100.00%              5
//    > 42            > 100%                        97
//   Total                                          105
//

//  Analyzed a total of 105 EAB cubes.
//  Top specified scan cells are:
//   ID Specified  Gate                                                      Chain Shift
//      times                                                                     position
//-- ---------  ---------------------------------------------------------   ----- --------
//1  14(13.33%) '/gate/Engine_Instance_1/SE_FMUL00/Z_reg[14]1/Q' (565444)    880   18
//2  14(13.33%) '/gate/Engine_Instance_1/SE_FMUL00/OP5_L2_reg[4]/Q' (565169) 814   18
//3  14(13.33%) '/gate/Engine_Instance_1/SE_FMUL00/OP5_L2_reg[3]/Q' (565104) 812   22
//4  12(11.43%) '/gate/Engine_Instance_1/SE_EX0/PX_COO1_reg[16]8/' (561265)  854   10
//5  12(11.43%) '/gate/Engine_Instance_1/SE_FMUL00/OP5_L2_reg[25]/Q' (565260)817   18
//6  11(10.48%) '/gate/Engine_Instance_1/SE_FMUL00/OP5_L2_reg[13]/Q' (565077)811   19
//7  10(9.52%)  '/gate/Engine_Instance_1/SE_EX0/PX_COO1_reg[28]8/' (561266)  854   9
//8  10(9.52%)  '/gate/Engine_Instance_1/SE_FMUL00/OP5_L2_reg[21]/Q' (565223)816   25
//9  10(9.52%)  '/gate/Engine_Instance_1/SE_EX0/COO_REG_OUT_SEL_reg/' (561258)854  17
//10 9(8.57%)   '/gate/Engine_Instance_1/SE_FMUL00/OP5_L2_reg[2]/Q' (565196)  815  21
//
//  Top specified shift positions are:
//   Shift position  Specified times  Average specified times per cube
//   --------------  ---------------  --------------------------------
//   19              94               0
//   18              84               0
//   15              81               0
//   22              75               0
//   21              72               0
//   25              68               0
//   10              68               0
//   13              66               0
//   17              66               0
//   16              65               0
//
//  Total analysis time = 0.38 sec.
```

Example 6

The following example reports EAB statistics for COMPOSITE_BLOCK_0 from the previous example.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

In this example, the “Gate” column entries are shortened with /gate, which represents the following:

```
/EngineX4_i
```

This is done for presentation purposes.

ANALYSIS> report_edt_abort_analysis -block COMPOSITE_BLOCK_0 -cubes -from 1 -to 2

```
//EAB test cube #1:
//Gate                                                                  Value  Chain  Shift
//                                                                               pos
//                                                                              (from sci)
//-------------------------------------------------------------        -----  -----  --------
//'/gate/Engine_Instance_3/SE_EX0/REG_OUT_SEL_reg25/Q' (574882)        0      386    22
//'/gate/Engine_Instance_3/SE_EX0/PX_COO0_reg[18]5/' (574910)          1      387    25
//'/gate/Engine_Instance_3/SE_EX0/PX_COO0_reg[27]5/' (574849)          1      385    23
//'/gate/Engine_Instance_3/SE_EX0/PX_COO0_reg[26]5/' (574915)          0      387    20
//'/gate/Engine_Instance_3/SE_FMUL00/OP_IN_REG_SEL_reg[3]/Q' (579096)0      314    26
//'/gate/Engine_Instance_3/SE_FMUL00/C_FMUL_reg[3]/Q' (579320)         0      365    22
//'/gate/Engine_Instance_3/SE_FMUL00/C_FMUL_reg[0]/Q' (579252)         0      363    27
//'/gate/Engine_Instance_3/SE_EX0/PX_COO0_reg[24]2/' (573291)          0      320    27
//'/gate/Engine_Instance_3/SE_EX0/PX_COO0_reg[8]5/' (574852)           1      385    20
//Total bits specified = 9
//
//EAB test cube #2:
//Gate                                                                  Value  Chain  Shift
//                                                                               pos
//                                                                              (from sci)
//-------------------------------------------------------------        -----  -----  --------
//'/gate/Engine_Instance_3/SE_PE0/DX23_reg[9]/' (576871)               0      300    30
//'/gate/Engine_Instance_3/SE_EX0/PX_COO0_reg[20]5/' (574908)          1      387    27
//'/gate/Engine_Instance_3/SE_EX0/PX_COO0_reg[0]5/' (574913)           1      387    22
//'/gate/Engine_Instance_3/SE_EX0/PX_COO0_reg[14]5/' (574850)          1      385    22
//'/gate/Engine_Instance_3/SE_EX0/PX_COO0_reg[29]5/' (574905)          0      387    30
//'/gate/Engine_Instance_3/SE_EX0/PX_COO0_reg[18]5/' (574910)          1      387    25
//'/gate/Engine_Instance_3/SE_EX0/PX_COO0_reg[22]5/' (574907)          0      387    28
//'/gate/Engine_Instance_3/SE_EX0/REG_OUT_SEL_reg28/Q' (575097)        1      393    28
//'/gate/Engine_Instance_3/SE_FMUL00/OP_IN_REG_SEL_reg[0]/Q' (579345)1      366    29
//Total bits specified = 9
//
//  Total analysis time = 0.00 sec.
```

Related Topics

set_edt_abort_analysis_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_edt_blocks

Context: dft -edt, patterns -scan (EDT On), patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays a list of the current user-defined EDT block names (also referred to as block tags) and indicates which one, if any, represents the current block.

Usage

report_edt_blocks

Description

Displays a list of the current user-defined EDT block names (also referred to as block tags) and indicates which one, if any, represents the current block.

When the display shows “current block” by a name, it indicates relevant tool commands apply only to the EDT block associated with that name.

You can transfer this restriction (also referred to as the EDT context) to another EDT block with the set_current_edt_block or add_edt_blocks commands.

Arguments

None

Examples

The following example lists currently defined EDT block names:

report_edt_blocks

```
//  my_core1_block
//  my_core2_block
//  my_core3_block
//  my_core4_block   (current block)
```

Related Topics

add_edt_blocks

delete_edt_blocks

set_current_edt_block

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_edt_configurations

Context: dft (with no subcontext), dft -edt, patterns -scan (EDT On), patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays configuration information for the current compression logic.

Usage

report_edt_configurations [-ALl_blocks] [{> | >>} *file_pathname*]

Description

Displays configuration information for the current compression logic.

Among the items reported are the version, compactor type, power controller status, low-power pipeline stages, number of scan channels and related information, multiple compression configurations, and whether the tool is set up to insert bypass logic and lockup cells. This command only displays a partial report from Setup mode. To produce a complete report, you must exit Setup mode to run DRC and flatten the model before issuing this command.

Note

The report_edt_configurations command does not list the names of EDT pins. To display the names of EDT pins, use the report_edt_pins command.

The -all_blocks switch is useful where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the EDT configuration information as described above, but only for the EDT block you have specified as the current EDT block. If you include the -all_blocks switch, the command displays configuration information for every EDT block. You set the current EDT block using either the add_edt_blocks or set_current_edt_block command. “EDT block” is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the Modular Compressed ATPG chapter of the *Tessent TestKompress User’s Manual*.

You can use the report_edt_configurations command in the dft context with no subcontext, together with the “process_dft_specification -no_insertion” command, to review the potential configuration of EDT IP created with the current DftSpecification EDT settings.

Arguments

- -ALl_blocks

  An optional switch that displays EDT configuration information for all EDT blocks. Without this switch, only information for the current EDT block displays. If a current EDT block is not defined, configuration information for all EDT blocks displays.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of file_pathname.

Examples

Example 1

The following example reports configuration details for a non-modular design with eight scan chains and one scan channel; first while in Setup mode, then again after DRC:

report_edt_configurations

```
// IP version:                 1
// External scan channels:     1
// Bypass logic:               On
// Lockup cells:               On
// Clocking:                   edge-sensitive
```

set_system_mode analysis

```
...
# -----------------------------------------------------
# Begin EDT rules checking.
# -----------------------------------------------------
# Running EDT logic creation Phase.
# 5 pin(s) will be added to the EDT wrapper. (K11)
# EDT rules checking completed, CPU time=0.01 sec.
# All scan input pins were forced to TIE-X.
# All scan output pins were masked.
# -----------------------------------------------------
...
```

report_edt_configurations

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// IP version:                  1
// Shift cycles:                381, 373 (internal scan length)
//                              + 8 (additional cycles)
// External scan channels:      1
// Internal scan chains:        8
// Masking bits:                2
// Longest chain range:         2 - 373
// Decompressor size:           32
// Shift power controller:      Enabled and inactive(shift power control
//                              is off)
// Min switching threshold:     25%
// Compactor type:              Xpress
// Scan cells:                  18650
// Compression per pattern:     16.01x (ATPG bypass = 3 x 6341, EDT = 3 x
//                              396)
// Clocking:                    edge-sensitive
// Compactor pipelining:        Off
```

Example 2

The following example defines two EDT blocks in the patterns -scan context, then reports configuration details first for the current block and then for all blocks:

```
# Define the block name "my_block1" for an EDT block
#   and automatically set it as the current EDT block.
```

add_edt_blocks my_block1

```
# Define the location (instance name) of the top-level EDT logic for this
#   block; also the decompressor and compactor.
```

set_edt_instances -edt_logic_top core1_edt_i
set_edt_instances -decompressor core1_edt_decompressor_i
set_edt_instances -compactor core1_edt_compactor_i

add_scan_chains...
add_scan_chains...

set_edt_options -channels 3

```
# Specify the top-level control signals and channel inputs and outputs for
#   this block (required).
```

set_edt_pins...

```
# Repeat the preceding procedure for another block.
```

add_edt_block my_block2

set_edt_instances -edt_logic_top core2_edt_i
set_edt_instances -decompressor core2_edt_decompressor_i
set_edt_instances -compactor core2_edt_compactor_i

add_scan_chains...
add_scan_chains...

set_edt_options -channels 1

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_edt_pins...

report_edt_configurations

```
//  EDT block "my_block2"
//  --------------------
//  IP version:                     3
//  External scan channels:         1
//  Decompressor size:              10
//  Compactor type:                 Xpress
//  Clocking:                       edge-sensitive
//
//  Note: Only current EDT block "my_block2" is reported. Use "report edt
//  configuration -all_blocks" to report the configurations of all blocks.
```

report_edt_configurations -all_blocks

```
//  EDT block "my_block1"
//  --------------------
//  IP version:                     3
//  External scan channels:         3
//  Decompressor size:              12
//  Compactor type:                 Xpress
//  Clocking:                       edge-sensitive
//
//  EDT block "my_block2"
//  --------------------
//  IP version:                     3
//  External scan channels:         1
//  Decompressor size:              10
//  Compactor type:                 Xpress
//  Clocking:                       edge-sensitive
```

Example 3

The following example reports all EDT blocks. Channel sharing information is shown in red font. Note that "// Blocks sharing input channels:" means the blocks having at least one input channel that is shared with this channel.

report_edt_configurations -all

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  EDT block "B1"
//  --------------
//  IP version:                    4
//  Shift cycles:            53,  19 (internal scan length) + 11
// (additional cycles) + 23 (pad cycles due to blocks with longer chains)
//  External scan channels:        2 (1 control input channel)
//  Internal scan chains:          32
//  Masking bits:                  25
//  Longest chain range:           2 - 19
//  Decompressor size:             16
//  Compactor type:                Xpress
//  Blocks sharing input channels: B2
//  Scan cells:                    573
//  Clocking:                      edge-sensitive
//  Compactor pipelining:          0 stages
//
//  EDT block "B2"
//  --------------
//  IP version:                    4
//  Shift cycles:             53,  2 (internal scan length) + 13
// (additional cycles) + 38 (pad cycles due to blocks with longer chains)
//  External input channels:       3 (1 control input channel)
//  External output channels:      2
//  Internal scan chains:          4
//  Masking bits:                  5
//  Longest chain range:           2 - 2
//  Decompressor size:             8
//  Compactor type:                Xpress
//  Blocks sharing input channels: B1, B3
//  Scan cells:                    8
//  Clocking:                      edge-sensitive
//  Compactor pipelining:          0 stages
//
//  EDT block "B3"
//  --------------
//  IP version:                    4
//  Shift cycles:             53,  2 (internal scan length) + 13
// (additional cycles) + 38 (pad cycles due to blocks with longer chains)
//  External scan channels:        2 (1 control input channel)
//  Internal scan chains:          4
//  Masking bits:                  5
//  Longest chain range:           2 - 2
//  Decompressor size:             8
//  Compactor type:                Xpress
//  Blocks sharing input channels: B2
//  Scan cells:                    8
//  Clocking:                      edge-sensitive
//  Compactor pipelining:          0 stages
```

Example 4

In this example, the tool reports while in non-setup mode if there is an EDT low-power controller:

report_edt_configurations

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//
//  IP version:                    6
//  Shift cycles:                  148,  143 (internal scan length) + 5 (additional cycles)
//  External scan channels:        23 (4 control input channels)
//  Internal scan chains:          1827
//  Masking bits:                  254
//  Decompressor size:             92 (4 segments)
//  Scan cells:                    258561
//  Compression per pattern:       76.33x (ATPG bypass = 23 x 11297, EDT = 23 x 148)
//  Bypass logic:                  On
//  Lockup cells:                  On
//  Clocking:                      edge-sensitive
//  Compactor pipelining:          Off
//  Low power shift controller:    Enabled and active
//    Min switching threshold:     25%
//    Low power bits:                 30
```

Related Topics

add_edt_configurations

delete_edt_configurations

report_edt_pins

report_scan_volume

set_edt_options

set_edt_pins

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_edt_finder

Context: patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Reports information about the EDT logic contained in a design.

Usage

To report decompressors

report_edt_finder **-Decompressors** [-Type {ACtive | Inactive | ALl}]
[-Id ***decompressor_id#***] [-Filter ***instance_name_wildcard***]
[-Verbose [instances]] [{> | >>} ***file_pathname***]

To report scan chain segments

report_edt_finder **-Chain_segments**
[-Id ***decompressor_id#***] [-Type {ALl | ACcessible [COMpressed | Uncompressed | CONtrol] | Inaccessible}] [-MIn_length ***integer***] [-MAx_length ***integer***] [-Bins ***integer***]
[-Endpoints {***instance_name_wildcard*** | ***gate_id#***}]
[-Details {SUmmary | Endpoints | Full}] [{> | >>} ***file_pathname***]

To report masking logic

report_edt_finder **-Masking_logic** [-Id ***decompressor_id#***] [-Filter ***instance_name_wildcard***]
[-Verbose [instances]] [{> | >>} ***file_pathname***]

To report low-power shift controller logic

report_edt_finder **-Low_power_shift_controller** [-Id ***decompressor_id#***]
[-Filter ***instance_name_wildcard***] [-Verbose [instances]] [{> | >>} ***file_pathname***]

Description

Reports information about the EDT logic contained in a design. Depending on the specified switches, the following columns can be reported:

- Length — Specifies the length of the identified scan chains. For example, (16-101) means the smallest scan chain has 16 cells, and the longest scan chain has 101 cells.
- Count — Specifies the number of scan chains. For example, the first line of this report shows there are 40 chains that are compressed with lengths 16-101.

```
// Type         Length     Count
// ------------ -------- -------
// compressed   16-101     40
// compressed   102-188    50
// inaccessible 1-94       7520
// inaccessible 95-189     1160
```

- Type — Specifies the type of chain segment as defined by the Type argument description.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Begin/End — Specifies the first scan cell or last scan cell of the scan chain segment. Defined by the -Details Endpoints switch.

Arguments

- **-Decompressors**

  A required switch that reports on decompressors found in the design.

- [-Type {ACtive| Inactive | ALl}]

  An optional switch and literal pair that specifies which decompressors to report on. Options include:

  ACtive — Reports on all active decompressors identified. This is the invocation default.

  Inactive — Reports on all inactive decompressors identified (decompressors that do not reach channel outputs).

  ALl — Reports all decompressors identified.

- **-Chain_segments**

  A required switch that reports on scan chain segments found in the design.

- -Type {ALl | ACcessible [Compressed | Uncompressed | CONtrol] | Inaccessible}}

  An optional switch and literal pair that specifies the type of chain segments to report. Literal options include:

  ALl — Reports all chain segments found. This is the invocation default.

  ACcessible — Reports all scan chain segments accessible via identified EDT logic or primary input/output pins. You can also enter an additional literal option to further sort on the accessible scan chains returned. Literal sort options include:

  COMpressed — Reports compressed chains, such as chains accessible via identified EDT logic.

  Uncompressed — Reports uncompressed chains such as chains accessible via primary input/output pins.

  CONtrol — Reports chain segments of the EDT control logic such as the mask shift registers of Xpress compactors.

  Inaccessible — Reports chain segments that are not accessible via identified EDT logic or primary input/output pins.

- -MIn_length ***integer***

  An optional switch and integer pair that specifies the minimum length of chain segments to display. The default is 1.

- -MAx_length ***integer***

  An optional switch and integer pair that specifies the maximum length of chain segments to display. The default is unlimited.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Endpoints {***instance_name_wildcard*** | ***gate_id#***}

  An optional switch and string or integer that specifies the endpoints of the chain segments to display. The string may include asterisk (*) and question mark (?) wildcard characters.

- -Details {SUmmary | Endpoints | Full}

  An optional switch and literal pair that specifies the details to display. Options include:

  SUmmary — Reports only a summary such as the number of chains that belong to a certain category. This is the invocation default.

  Endpoints — Reports the endpoints of the chain segments.

  Full — Reports all available information about the chain segments, including the instance name of each chain element.

- -Bins ***integer***

  An optional switch and integer pair that specifies the maximum number of bins that scan chain data is sorted by in the summary report.

  Each type (accessible/compressed/uncompressed/control/inaccessible) of scan chain reported is sorted by scan chain lengths determined by the number of bins specified with this switch. By default, 10 bins are used.

- **-Masking_logic**

  A required switch that reports details about specified Xpress mask registers identified by EDT Finder.

- **-Low_power_shift_controller**

  A required switch that reports details about low-power shift and hold registers identified by EDT Finder.

- -Id ***decompressor_id#***

  An optional switch and integer pair that specifies the identification number of the decompressor to report. The value of the *decompressor_id#* is the unique identification number automatically assigned to every decompressor by EDT Finder.

- -Filter ***instance_name_wildcard***

  An optional switch and string pair that specifies the instance name of the decompressors or masking logic to report. You can use regular expressions that include asterisk (*) and question mark (?) wildcard characters.

- -Verbose [instances]

  An optional switch and literal that transcribes informational messages about the EDT logic found in a design. Depending on the EDT logic specified, this switch reports informational messages about decompressors, low power registers, and masking logic found in the design. If the instances literal is specified, only the names of instances are reported. By default, these messages are disabled.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- >***file_pathname***

  An optional redirection operator and pathname pair that creates or replaces the contents of *file_pathname* with the output.

- >> ***file_pathname***

  An optional redirection operator and pathname pair that appends the output to the contents of *file_pathname*.

Examples

Example 1

The following example displays a summary of decompressors found in a design:

report_edt_finder -decompressors

```
// Decomp. Id# #Bits #Inputs #Chains  Block      Type
// ----------- ----- ------- -------  ---------  -----
//  1            16      4      28     m1_28x16   active
//  2            16      4       4     m2_4x32    active
//  3            16      4      50     m3_50x187  active
//  4            10      1       8     m4_8x16    active
```

Example 2

The following example displays a summary of all the masking logic found:

report_edt_finder -masking_logic

```
// Decomp. Id# Length Type     Block
// ----------- ------ ------   --------------
// 1           9      xpress   piccpu_40
// 2           31     xpress   my_core_5ch
// 3           31     xpress   my_core_5ch_2
```

Example 3

This example displays a summary of all found chain segments grouped into 2 bins per type:

report_edt_finder -chain_segments -bins 2

```
// Type          Length    Count
// ------------  --------  -------
// compressed    16-101    40
// compressed    102-188   50
// inaccessible  1-94      7520
// inaccessible  95-189    1160
```

Example 4

The following example displays the scan inputs and outputs of the accessible chain segments with a minimum length of 50:

report_edt_finder -chain_segments -type accessible compressed -min_length 50

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  EDT Control         Length  Type          Name
//  ----------------    ------  ------------  ------------------
//  edt controllable    50      compressed    piccpu_90_chain2
//  edt controllable    90      compressed    piccpu_90_chain3
//  edt controllable    50      compressed    piccpu_90_2_chain2
//  edt controllable    90      compressed    piccpu_90_2_chain3
```

Example 5

The following example displays a summary of the low-power shift controller logic in blk1:

report_edt_finder -low_power_shift_controller -id 2 -verbose instances

```
// Decomp. Id # #Registers     Block
// ------------ -------------  ---------------------
// 2            5              blk1
// Shift reg.   /top/blk1_i/.../low_power_shift_reg_0_reg[4]/UD1 (16171)
// Hold reg.    /top/blk1_i/.../control_hold_reg_0_reg[4]/DFF1 (16166)
// Shift reg.   /top/blk1_i/.../low_power_shift_reg_0_reg[3]/UD1 (16172)
// Hold reg.    /top/blk1_i/.../low_power_hold_reg_0_reg[3]/DFF1 (16167)
// Shift reg.   /top/blk1_i/.../low_power_shift_reg_0_reg[2]/UD1 (16173)
// Hold reg.    /top/blk1_i/.../low_power_hold_reg_0_reg[2]/DFF1 (16168)
// Shift reg.   /top/blk1_i/.../low_power_shift_reg_0_reg[1]/UD1 (16174)
// Hold reg.    /top/blk1_i/.../low_power_hold_reg_0_reg[1]/DFF1 (16169)
// Shift reg.   /top/blk1_i/.../low_power_shift_reg_0_reg[0]/UD1 (16175)
// Hold reg.    /top/blk1_i/.../low_power_hold_reg_0_reg[0]/DFF1 (16170)
```

Related Topics

set_edt_finder

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_edt_instances

Context: dft -edt, patterns -scan (EDT On), patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays the instance pathnames of the top-level EDT logic, decompressor, and compactor.

Usage

report_edt_instances [-All_blocks] [-Errors_only] [{> | >>} ***file_pathname***]

Description

Displays the instance pathnames of the top-level EDT logic, decompressor, and compactor.

Note

For modular Tessent TestKompress, the report_edt_instances command operates only on the current EDT block (the EDT block that is the current target of EDT-specific commands). This is the EDT block specified by the last add_edt_blocks or last set_current_edt_block command, whichever you issued most recently.

The report_edt_instances command lists the names of the instances where the tool expects to find the top-level EDT logic, decompressor, and compactor. It also reports the pathnames of these instances when known. If the display lists any instance paths as "NOT FOUND," the tool is not able to provide the maximum amount of detail in the debugging information for K19-K22 DRC violations. You can use the set_edt_instances command to update the instance information.

The -All_blocks switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the information described above, but only for the EDT block you have specified as the current EDT block. If you include the -All_blocks switch, the command displays this information for every EDT block. You set the current EDT block using either the add_edt_blocks or set_current_edt_block command. "EDT block" is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the "Modular Compressed ATPG" chapter of the *Tessent TestKompress User's Manual*.

Arguments

- -All_blocks

  Note

  This switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. If you are using a standard (non-modular) Tessent TestKompress flow, you do not need this switch.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

An optional switch that specifies to display instance information for all EDT blocks. Without this switch, the tool displays information for only the current EDT block, if it is defined. If a current EDT block is not defined, the tool displays the information for all EDT blocks.

- -Errors_only

  An optional switch that specifies to display instance inform &‡n only if the tool is unable to positively identify at least one of the instance names. This could be because the tool either cannot find the instance name/path, or the instance name is not unique.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following example displays the current EDT instance data for a design, updates the information, then displays the updates:

report_edt_instances

```
// edt_logic_top
//    Setting: edt_i
//    Path   : NOT FOUND
//
// decompressor
//    Setting: edt_decompressor_i
//    Path   : NOT FOUND
//
// compactor
//    Setting: edt_compactor_i
//    Path   : NOT FOUND
```

**set_edt_instances -edt_logic_top my_core_edt_i **
**-decompressor my_core_edt_decompressor_i **
-compactor my_core_edt_compactor_i

```
// Found edt_logic_top at path: "/my_core_edt_top/my_core_edt_i".
// Found compactor at path:
//    "/my_core_edt_top/my_core_edt_i/my_core_edt_compactor_i".
// Found decompressor at path:
//    "/my_core_edt_top/my_core_edt_i/my_core_edt_decompressor_i".
```

report_edt_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// edt_logic_top
//    Setting: my_core_edt_i
//    Path   : /my_core_edt_top/my_core_edt_i
//
// decompressor
//    Setting: my_core_edt_decompressor_i
//    Path   : /my_core_edt_top/my_core_edt_i/my_core_edt_decompressor_i
//
// compactor
//    Setting: my_core_edt_compactor_i
//    Path   : /my_core_edt_top/my_core_edt_i/my_core_edt_compactor_i
```

Related Topics

add_edt_blocks

set_current_edt_block

set_edt_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_edt_lockup_cells

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Displays a tabular report of the lockup cells in the EDT logic.

Usage

report_edt_lockup_cells [-All_blocks] [{> | >>} *file_pathname*]

Description

Displays a tabular report of the lockup cells in the EDT logic.

The report_edt_lockup_cells command reports details about the lockup cells in the EDT logic.

The command only reports lockup cells that are included in the generated IP. Specifically, when lockup cell insertion is disabled (which is not recommended), the report does not include any lockup cells.

Note

During pattern generation, the command reports the expected lockup cells and not the actual lockup cells in the design.

The reported information is arranged in tables that cover the following:

Note

If you invoke this command in Setup mode because EDT DRC failed, some data is not available, and hence, is not reported.

- Scan clocking — Details about the first and last scan cell in each scan chain. For each flip-flop (FF), the report lists whether it is leading edge triggered (LE) or trailing edge triggered (TE). For each latch, the report lists whether it is active high (HI) or active low (LO). The names of the first and last cell's clocks and their capture timing are also provided. The capture time is listed in the "Cap" column for the first cell, and in the "Chg" column for the last cell.
- Location — Each lockup cell is listed under one of the following categories depending on its location in the IP:
  - Decompressor — Lockup cells between the decompressor and the scan chain inputs
  - Compactor — Lockup cells between the scan chain outputs and the compactor
  - Bypass — Lockup cells in the bypass circuitry

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Type of lockup cell — Flip-flop (FF) or latch (LA). For each flip-flop, the report lists whether it is leading edge triggered (LE) or trailing edge triggered (TE). For each latch, the report lists whether it is active high (HI) or active low (LO).
- Clock — Name of the clock for each lockup cell
- Relevant Clocks — Summary of the off state and timing of all clocks listed elsewhere in the report. The timing is based on the shift procedure in the test procedure file.

For more information, see "Understanding Lockup Cells" in the Tessent TestKompress User's Manual.

The -All_blocks switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the lockup cell information described above, but only for the EDT block you have specified as the current EDT block. If you include the -All_blocks switch, the command displays this information for every EDT block. You set the current EDT block using either the add_edt_blocks or set_current_edt_block command. "EDT block" is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the "Modular Compressed ATPG" chapter of the *Tessent TestKompress User's Manual*.

Arguments

- -All_blocks

  Note

  This switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. If you are using a standard (non-modular) Tessent TestKompress flow, you do not need this switch.

  An optional switch that specifies to display the lockup cell information for all EDT blocks. Without this switch, the tool displays information for only the current EDT block, if it is defined. If a current EDT block is not defined, the tool displays the information for all EDT blocks.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example reports lockup details for a design with eight scan chains and one scan channel. The EDT logic is FF-based and includes bypass circuitry:

set_system_mode analysis
report_edt_lockup_cells

```
// SCAN Clocking
//----------+-----------------------+----------------------
//          |        First Cell     |       Last Cell
// Chain (#)|Gate (id) Type Cap Clock|Gate (id) Type Chg Clock
//----------+-----------------------+----------------------
// chain1(1)|/i7/ (92) FF-TE 20 /clk |/rg0/ (33) FF-TE 20 /clk
   ...
// chain8(8)|/i5/ (48) FF-TE 20 /clk |/ix8/ (89) FF-TE 20 /clk
//
// DECOMPRESSOR Lockup Cells
//----------+------------------+----------------
//          | First cell       | Second cell
// Chain (#)| Type  Clock      | Type  Clock
//----------+------------------+----------------
// chain1(1)| FF-TE /edt_clock | ----- ----------
// chain2(2)| FF-TE /edt_clock | FF-LE /edt_clock
   ...
// chain8(8)| FF-TE /edt_clock | ----- ----------
//
// COMPACTOR Lockup Cells
//----------+----------------
//          | Lockup cell
// Chain (#)| Type  Clock
//----------+----------------
// chain1(1)| FF-TE /edt_clock
   ...
// chain8(8)| FF-TE /edt_clock
//
// BYPASS Lockup Cells
//----------------------------+-----------
//                             |Lockup cell
// Chain  (#)  ->  Chain  (#)  |Type  Clock
//-----Channel 1--------------+-----------
// chain1 (1)  ->  chain2 (2)  |FF-TE /clk
// chain2 (2)  ->  chain3 (3)  |----- -----
// chain3 (3)  ->  chain4 (4)  |----- -----
// chain4 (4)  ->  chain5 (5)  |----- -----
// chain5 (5)  ->  chain6 (6)  |----- -----
// chain6 (6)  ->  chain7 (7)  |----- -----
// chain7 (7)  ->  chain8 (8)  |----- -----
//
// RELEVANT Clocks
//------------+-------+-----
//            |  Off  | Time
// Name       | State |LE TE
//------------+-------+-----
// /edt_clock |   0   |20 30
// /clk       |   0   |20 30
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following is the bypass portion of a report for a design with eight scan chains and two scan channels. Notice that in addition to listing the bypass lockups, the bypass tables show the chain connections to the channels.

```
// BYPASS Lockup Cells
//---------------------------------+-----------
//                                 |Lockup cell
// Chain    (#)  ->  Chain    (#)  |Type  Clock
//-----Channel 1-------------------+-----------
// chain1   (1)  ->  chain2   (2)  |----- -----
// chain2   (2)  ->  chain3   (3)  |----- -----
// chain3   (3)  ->  chain4   (4)  |----- -----
// chain4   (4)  ->  chain5   (5)  |----- -----
// chain5   (5)  ->  chain6   (6)  |----- -----
//-----Channel 2-------------------+-----------
// chain7   (7)  ->  chain8   (8)  |----- -----
// chain8   (8)  ->  chain9   (9)  |----- -----
// chain9   (9)  ->  chain11 (10)  |FF-TE /NX1
// chain11 (10)  ->  chain10 (11)  |FF-TE /NX2
```

Example 2

The following example shows a design with eight scan chains; four of the scan chains are clocked by LE clk1 and the other four scan chains are clocked by TE clk2. The report_edt_lockup_cells output that follows shows the inserted lockup cells at the scan chain inputs and outputs.

```
set_edt_options -lockup on -retime_chain_boundaries on
set_system_mode analysis
report_edt_lockup_cells
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//
//  SCAN Clocking
//  -----------+---------------------------------------+-------------------------------------
//             |                 First cell            |                Last cell
//  Chain  (#) | Gate              (id) Type  Cap Clock | Gate             (id) Type  Chg Clock
//  -----------+---------------------------------------+-------------------------------------
//  chain1 (1) | /chain1_cell19/Q (234) FF-LE  40 /clk1 | /chain1_cell0/Q (215) FF-LE  40 /clk1
//  chain2 (2) | /chain2_cell19/Q (254) FF-LE  40 /clk1 | /chain2_cell0/Q (235) FF-LE  40 /clk1
//  chain3 (3) | /chain3_cell19/Q (274) FF-LE  40 /clk1 | /chain3_cell0/Q (255) FF-LE  40 /clk1
//  chain4 (4) | /chain4_cell19/Q (294) FF-LE  40 /clk1 | /chain4_cell0/Q (275) FF-LE  40 /clk1
//  chain5 (5) | /chain5_cell19/Q (314) FF-TE  50 /clk2 | /chain5_cell0/Q (295) FF-TE  50 /clk2
//  chain6 (6) | /chain6_cell19/Q (334) FF-TE  50 /clk2 | /chain6_cell0/Q (315) FF-TE  50 /clk2
//  chain7 (7) | /chain7_cell19/Q (354) FF-TE  50 /clk2 | /chain7_cell0/Q (335) FF-TE  50 /clk2
//  chain8 (8) | /chain8_cell19/Q (374) FF-TE  50 /clk2 | /chain8_cell0/Q (355) FF-TE  50 /clk2
//
//  ------  ---  ----------------  -----------  ------------
//                 Decompressor    Chain Input  Chain Output
//                    Lockup          Lockup       Lockup
//  Chain   (#)       Type  Clock  Type  Clock  Type  Clock
//  ------  ---  ----------------  -----------  ------------
//  chain1  (1)  FF-TE /edt_clock  ----- -----  FF-TE /clk1
//  chain2  (2)  FF-TE /edt_clock  ----- -----  FF-TE /clk1
//  chain3  (3)  FF-TE /edt_clock  ----- -----  FF-TE /clk1
//  chain4  (4)  FF-TE /edt_clock  ----- -----  FF-TE /clk1
//  chain5  (5)  FF-TE /edt_clock  FF-LE /clk2  ----- -----
//  chain6  (6)  FF-TE /edt_clock  FF-LE /clk2  ----- -----
//  chain7  (7)  FF-TE /edt_clock  FF-LE /clk2  ----- -----
//  chain8  (8)  FF-TE /edt_clock  FF-LE /clk2  ----- -----
//
//  RELEVANT Clocks
//  -----------+-------+---------------+------
//             |  Off  |               |  Time
//  Name       | state | Type          | LE TE
//  -----------+-------+---------------+------
//  /edt_clock |   0   | EDT           | 40 50
//  /clk1      |   0   | Shift  MASTER | 40 50
//  /clk2      |   0   | System MASTER | 40 50
//
```

Related Topics

report_edt_configurations

set_edt_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_edt_pins

Context: dft -edt, patterns -scan (EDT On), patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays the currently defined names and inversion status of EDT channel and control pins.

Usage

report_edt_pins [-All_blocks] [-Group_by_pin_name]
[{> | >>} *file_pathname*]

Description

Displays the currently defined names and inversion status of EDT channel and control pins. The report_edt_pins command displays the names of all EDT channel and control pins and whether the pin signals are inverted between the chip input pin and the EDT logic.

The -All_blocks switch is only for the top-level Pattern Generation phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the information described above, but only for the EDT block you have specified as the current EDT block. If you include the -All_blocks switch, the command displays this information for every EDT block. You set the current EDT block using either the add_edt_blocks or set_current_edt_block command. “EDT block” is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the “Modular Compressed ATPG” chapter of the *Tessent TestKompress User’s Manual*.

Arguments

- -All_blocks

  Note

  This switch is only for the top-level Pattern Generation phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. If you are using a standard (non-modular) Tessent TestKompress flow, you do not need this switch.

  An optional switch that specifies to display the EDT channel and control pin information for all EDT blocks. Without this switch, the tool displays information for only the current EDT block, if it is defined. If a current EDT block is not defined, the tool displays configuration information for all EDT blocks.

- -Group_by_pin_name

  An optional switch that specifies to display the pin name and all of the blocks that share that pin.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- *>> file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

Example 1

Assume the tool is in setup mode. The following example reports, for a design with two scan channels and no shared EDT pins, the names and inversion status of the EDT pins. The example then shares the EDT clock pin with a functional pin called “a1”. Following that, it renames the dedicated EDT bypass pin from the default name to “my_bypass” and specifies in the same command that the bypass signal is inverted between the chip input pin and the EDT logic. Afterward, the example again reports the EDT pins.

report_edt_pins

```
//
//    Pin description         Pin name           Inversion
//    ---------------         --------           ---------
//    Clock                   edt_clock              -
//    Update                  edt_update             -
//    Bypass mode             edt_bypass             -
//    Scan channel 1 input    edt_channels_in1       -
//     "      "    " output   edt_channels_out1      -
//    Scan channel 2 input    edt_channels_in2       -
//     "      "    " output   edt_channels_out2      -
//
```

set_edt_pins clock a1
set_edt_pins bypass my_bypass -inv
report_edt_pins

```
//
//    Pin description         Pin name           Inversion
//    ---------------         --------           ---------
//    Clock                   a1                     -
//    Update                  edt_update             -
//    Bypass mode             my_bypass              inv
//    Scan channel 1 input    edt_channels_in1       -
//     "      "    " output   edt_channels_out1      -
//    Scan channel 2 input    edt_channels_in2       -
//     "      "    " output   edt_channels_out2      -
//
```

Example 2

The following example reports each pin name and lists all of the blocks that share that pin. Channel sharing information is shown in red font.

report_edt_pins -all -group_by_pin_name

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Pin name      Block  Pin                 Inversion  Channel pipeline stages
                     Description                    (mask register)
=========================================================================
edt_clock     A      Clock               -          -
              B      Clock               -          -
              C      Clock               -          -
edt_update    A      Update              -          -
              B      Update              -          -
              C      Update              -          -
channel_in_A  A      Scan channel 1      -          7
                     input(control)
channel_in_B  B      Scan channel 1      -          "
                     input(control)
channel_in_C  C      Scan channel 1      -          "
                     input(control)
channel_in1   A      Scan channel 2      -          7
                     input
              B      Scan channel 2      -          "
                     input
              C      Scan channel 2      -          "
                     input
channel_in2   A      Scan channel 3      -          7
                     input
              B      Scan channel 3      -          "
                     input
              C      Scan channel 3      -          "
                     input
channel_outA  A      Scan channel 1      -          0
                     Output
channel_outB  B      Scan channel 1      -          "
                     Output
channel_outC  C      Scan channel 1      -          "
                     Output
```

Related Topics

- report_drc_rules
- report_edt_configurations
- report_scan_volume
- set_edt_options
- set_edt_pins
- write_edt_files

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_environment

Context: unspecified, all contexts

Mode: all modes

Displays the current value(s) of many of the most common “set_...” commands.

Usage

report_environment [{> | >>} *file_pathname*]

Description

Displays the current value(s) of many of the most common “set_...” commands.

When you first invoke the tool, the values displayed are the default settings for these commands.

This command shows the set_gate_report command settings in a field titled “gate report”. The tool reports global on/off options only when the options are enabled. The global on options are reported in a semi-colon separated list in parenthesis. The tool reports clock_cone clock names in single quotes. The tool only reports the value of time values and not the units. The -max_fanouts and -clock_domains_in_capture_cycles options of set_gate_report do not impact the report_environment output.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of file_pathname.

Examples

The following example reports the current conditions under which the tool tests the circuit, then performs an ATPG run:

```
set_system_mode analysis
add_faults -all
report_environment
create_patterns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The output from the report_environment command may look like the following:

```
abort limit =              30
atpg compression =         OFF
atpg limits =              none
bist initialization =      X
bus simulation method =    global
capture clock =            none
checkpoint =               OFF
clock restriction =        domain_clocks (same-edge interaction)
clock_off simulation =     OFF
contention check =         ON, mode = bus, handling = warning
DRC transient detection = ON -Verbose
fails report =             OFF
fault mode =               uncollapsed
fault type =               stuck
gate level =               design
gate report =              normal
iddq checks =              none, handling = warning
iddq strobe =              label
learn reporting =          OFF
logfile handling =         OFF
net dominance =            wired-gate
net resolution =           wired-gate
observation point =        master
pattern classification =   ON
pattern source =           internal
possible credit =          50%
pulse generators =         ON
RAM initialization =       uninitialized
RAM test mode =            static_pass_thru
random atpg =              ON
random clocks =            none
random patterns =          1024
screen display =           ON
shadow checking =          ON
simulation mode =          combinational    depth = 0
skew load =                OFF
split capture cycle =      OFF
stability check =          ON
system mode =              analysis
tester cycles =            PLL_capture 10 timeplate_1
TLA loop handling =        OFF
trace report =             OFF
Z handling =               int=X  ext=X
Zhold behavior =           ON
```

Related Topics

report_pattern_filtering

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_external_simulator

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays the shell command the tool uses to invoke an external simulator for verifying patterns.

Usage

report_external_simulator [{> | >>} *file_pathname*]

Description

Displays the shell command the tool uses to invoke an external simulator for verifying patterns.

The report_external_simulator command displays the current setting of the set_external_simulator command. You use the latter to specify the shell command the tool should use to invoke an external time-based simulator for verifying patterns.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following example checks the current external simulator setting, then specifies for the tool to invoke Questa SIM in batch mode (-c) with loading messages disabled (-quiet), when verifying simulation results:

report_external_simulator

```
.../vsim -c
```

set_external_simulator vsim -c -quiet
report_external_simulator

```
.../vsim -c -quiet
```

Related Topics

set_external_simulator

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_failures

Context: dft -edt, patterns -scan, patterns -scan_diagnosis, patterns -failure_mapping

Mode: analysis

Simulates the current design and test patterns and reports the failures in a pattern-based format.

Usage

report_failures [{*pin_pathname* -stuck_at {0 | 1}}…] [-pdet | -exact] [-max *integer*] [-cycle]
[-measure_scan_at {channels | chains}]
[-return_status]
[-exclude_channels_or_chains {channel_name|channel_ouput_pin_name|
chain_name | chain_output_pin_name}...]
[{> | >>} *file_pathname*]

For “patterns -failure_mapping”

report_failures -unmapped

Description

Simulates the current design and test patterns and reports the failures in a pattern-based format.

For failure mapping, reports the unmapped top-level failures.

Arguments

- *pin_pathname* -stuck_at {0 | 1}

  An optional, repeatable string, switch and literal triplet that specifies both the location and the value of a stuck-at fault to check for failing patterns. A fault simulation is run on the specified fault and the results are reported. Argument options include:

  pin_pathname — A string that specifies the pin pathname of the fault whose failing patterns you want to identify.

  -stuck_at 0 | 1 — A switch and literal that specify the stuck-at fault value to simulate. The stuck-at options include:

  0 — stuck-at-0

  1 — stuck-at-1

- -pdet

  An optional switch that directs the tool to report a failure when an X is simulated where a binary value is expected. By default, only the binary detections are reported.

- -exact

  An optional switch directs the tool to report a failure when an X is simulated where a binary value is expected (like the -Pdet switch) and also when a binary value is simulated where an X is expected. By default, only the binary detections are reported.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -max *integer*

  An optional switch and integer pair that specifies a maximum number of failing patterns before the command stops the simulation. By default, all failing patterns are simulated.

- -cycle

  An optional switch that reports the failure file in a cycle-based format. By default, a pattern-based format is output. For more format information, see the "Input File Requirements" chapter in the *Tessent Diagnosis User's Manual*.

- -measure_scan_at {channels | chains}

  An optional switch and literal pair that instructs the tool to report the failing bits of an EDT-enabled design at the scan elements; normally, the tool reports the failing bits at the EDT external channels. For an EDT-enabled design, this switch is only valid for external pattern sets in ASCII or binary format; STIL and WGL formats are not supported.

  If EDT is enabled (On), by default, the tool measures at the external channels so that all failing bits are compressed. If you specify the chains option in this case, the tool reports the failing bits at the scan chain cell_number without considering the output compaction logic.

  If EDT is disabled (Off), the tool always measures at the scan cells. In this case, specifying the channels option creates an error condition.

  Reporting failures using the chains option generates the following information:

  pattern_id — pattern index where the failure occurs.

  chain/PO_name — scan chain or the primary output name where the failure occurs.

  cell_number — scan cell index where the failure occurs. This field is empty if the failure occurs at a primary output.

  expected_value — expected value observed at the scan chain output pin from the external pattern set for the failing point.

  simulated_value — the simulated value from the internal simulation for the failing point.

  cell_path_name — cell or primary output name of the failing point.

- -exclude_channels_or_chains {channel_name | channel_ouput_pin_name | chain_name | chain_output_pin_name} ...

  An optional switch and repeatable string pair that specifies the EDT channels or scan chains to exclude from reporting.

  channel_name — the name of an EDT channel to exclude

  channel_ouput_pin_name — the output pin name of an EDT channels to exclude

  chain_name — the name of a scan chain to exclude

  chain_ouput_pin_name — the output pin name of a scan chain to exclude

- -return_status

  An optional switch to return the status when pattern failures are detected.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

- -unmapped

  A required switch that specifies to report all failures that Tessent Diagnosis could not map to core instances.

Examples

Example 1

The following example loads a test pattern source, runs simulation, and displays the pattern-based failure information.

set_system_mode analysis
read_patterns file1
report_failures

```
// failing_patterns=8 simulated_patterns=36 simulation_time=0.00 sec
```

The pattern-based failure data is written in columns from left to right: pattern_id, chain/PO_name, cell_number, expected_value, simulated_value, cell_path_name as shown in the following example.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
chain_test
0 chain5   0   H  L  //  uUART/reg_MODE0_CLK
0 chain5   1   H  L  //  uCNTR/reg_PORT_3_SAMPLE_P1_2_
0 chain5   4   H  L  //  uCNTR/reg_PORT_3_SAMPLE_P1_5_
0 chain5   5   H  L  //  uINTR/ix546
0 chain5   8   H  L  //  uINTR/ix476
0 chain5   9   H  L  //  uINTR/ix446
0 chain5   12  H  L  //  uINTR/ix356
0 chain5   13  H  L  //  uINTR/ix326
0 chain5   16  H  L  //  uINTR/ix236
0 chain5   88  H  L  //  padding_cycle
0 chain5   100 H  L  //  padding_cycle
last_pattern_tested  0
scan_test
0 chain5   2   H  L  //  uCNTR/reg_PORT_3_SAMPLE_P1_3_
0 chain5   4   H  L  //  uCNTR/reg_PORT_3_SAMPLE_P1_5_
0 chain5   6   H  L  //  uINTR/ix106
0 chain5   8   H  L  //  uINTR/ix476
0 chain5   9   H  L  //  uINTR/ix446
0 chain5   11  H  L  //  uINTR/ix386
0 chain5   12  H  L  //  uINTR/ix356
0 chain5   13  H  L  //  uINTR/ix326
0 chain5   28  H  L  //  uSTM/reg_WAIT_SYNC/SFFR
0 chain5   100 H  L  //  padding_cycle
last_pattern_tested  2
```

Example 2

The following example shows the output of the report_failures command when the “-measure_scan_at chains’ switch and option combination is specified for an EDT-enabled design:

FAULT> read_patterns patterns.ascii
FAULT> report_failures -measure_scan_at chains

```
// pattern_id chain/    cell    expected simulated cell_path_name
//            PO_name   number  value    value
   0          chain1    2       H        L         // all_rams_i/sf_0003_1
   0          chain4    3       H        L         // all_rams_i/sf_0028_1
   0          chain10   1       L        H         // all_rams_i/lf_0055_1
   0          chain11   5       L        H         // all_rams_i/lf_0043_1
   1          chain7    4       H        L         // all_rams_i/sf_0053_1

// incorrect_patterns=2, simulated_patterns=61, simulation_time=0.04sec.
```

Example 3

The following example returns a list of the top-level failures for core instance logic_die_BOT_inst__logic_die__internal.

report_failures -unmapped

```
//Note: TestSuite[1] '../data/top_flog1',  #fail_bits=1Cycle  Pin
             Expected  Actual688    bc_2_core_2_q1       L         H
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

The design in the following example contains two cores, coreA and coreB. CoreA includes edt_channel1 with 8 chains: chainA, chainB, chainC, chainD, chainE, chainF, chainG, and chainH. CoreB includes edt_channel2 with 8 chains: chainI, chainJ, chainK, chainL, chainM, chainN, chainO, and chainP. For this example, six mismatches appear in the failure report:

ANALYSIS> report_failures -measure_scan_at chains -exact

```
//pattern_id chain/PO_name cell_number expected_value simulated_value cell_path_name
0            chainA         13           H               L                //  cellA2
0            chainP         15           H               L                //  cellP0
1            chainC         4            L               H                //  cellC11
1            chainO         4            L               H                //  cellO11
2            chainB         4            X               H                //  cellB11
2            chainP         0            X               L                //  cellP15
//  incorrect_patterns=3, simulated_patterns=29, simulation_time=0.01sec.
```

The following command excludes the mismatches in edt_channel2 and chainC:

**report_failures -measure_scan_at chains -exact \
-exclude_channels_or_chains edt_channel2 chainC**

```
//pattern_id  chain/PO_name  cell_number expected_value  simulated_value  cell_path_name
0             chainA         13           H               L                //  cellA2
2             chainB         4            X               H                //  cellB11
//  incorrect_patterns=2, simulated_patterns=29,
simToulation_time=0.01sec.
```

The following command reports failures in channel cycle format and excludes the cells in edt_channel2:

**report_failures -measure_scan_at channels -exact \
-exclude_channels_or_chains edt_channel2**

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//pattern_id channel/PO_name channel_cycle_id expected_value simulated_value
                                                          chain---cell_id---cell_path_name
0             edt_channel1     13                H               L
                                                                  //  chainA---13---cellA2
                                                                  //  chainB---13---cellB2
                                                                  //  chainC---13---cellC2
                                                                  //  chainD---13---cellD2
                                                                  //  chainE---13---cellE2
                                                                  //  chainF---13---cellF2
                                                                  //  chainG---13---cellG2
                                                                  //  chainH---13---cellH2
1             edt_channel1     4                 L               H
                                                                  //  chainA---4---cellA11
                                                                  //  chainB---4---cellB11
                                                                  //  chainC---4---cellC11
                                                                  //  chainD---4---cellD11
                                                                  //  chainE---4---cellE11
                                                                  //  chainF---4---cellF11
                                                                  //  chainG---4---cellG11
                                                                  //  chainH---4---cellH11
2             edt_channel1     4                 X               H
                                                                  //  chainA---4---cellA11
                                                                  //  chainB---4---cellB11
                                                                  //  chainC---4---cellC11
                                                                  //  chainD---4---cellD11
                                                                  //  chainE---4---cellE11
                                                                  //  chainF---4---cellF11
                                                                  //  chainG---4---cellG11
                                                                  //  chainH---4---cellH11
//  incorrect_patterns=3, simulated_patterns=29, simulation_time=0.02sec.
```

Related Topics

diagnose_failures

read_failures

read_patterns

write_failures

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_false_paths

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays false paths previously defined with the add_false_paths command.

Usage

report_false_paths [-All] [-Verbose] [-VIolations [-EXclude invalid_design_objects]]
{ [-From *source_node…* | -FROM_Clock *clock_signal_name…*]
[-TO *sink_node…* | -TO_Clock *clock_signal_name…*]
[{-THrough *through_node…*}…] }
| [-GAte {*gate_id#* | *pin_pathname*}...]
| [-X_Statistics [-path_count {*int* | all}] [-path_x_threshold *int*]
[-to_gate_count {*int* | all}] [-to_gate_x_threshold *int*]]
| [-DRC {C6 | C6-*n* | C22 | C22-*n*}]
[{> | >>} *file_pathname*]

Description

Displays false paths previously defined with the add_false_paths command.

Note that you can display SDC filenames and line numbers by first issuing the "set_timing_exceptions_handling -line_no" command. To analyze if a pin or a gate is affected by a false path, you can use "set_gate_report -timing_exceptions on" and report_gates commands.

If a specified starting point, end point, source node, clock signal name, sink node, or through node does not translate to an internal gate, zero (0) false paths are reported.

Arguments

When using the following arguments, you can include one or more asterisks (*) in pin or instance pathnames. An asterisk is treated as a wildcard character, enabling you to use it to match many pathnames in the design.

- -All

  An optional switch that displays all previously-defined false paths and multicycle paths.

- -Verbose

  An optional switch that displays all the gate information in the from, to, and through gate lists.

- -VIolations

  An optional switch that displays only the false paths that have violations, along with their respective violation messages.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -EXclude invalid_design_objects

  An optional switch and keyword pair to exclude the display of violation paths due to invalid objects.

- -From *source_node…*

  An optional switch and repeatable string pair that specifies the starting point of a false path. A valid starting point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, all the output pins of the instance are considered to be starting points. For a clock PI, the outputs of all the non-transparent latches, flip-flops, and RAMs associated with the clock PI are considered to be starting points. A pin pathname is considered to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and the internal clock pin is handled the same as a clock PI.

- -FROM_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the starting point of a false path by its associated clock pin pathname. Any pin pathname you specify with this argument is interpreted to be the source of a clock signal, and the outputs of all non-transparent latches, flip-flops, and RAMs associated with this clock signal are considered to be starting points.

- -TO *sink_node…*

  An optional switch and repeatable string pair that specifies the end point of a false path. A valid end point is a pin pathname, an instance pathname, a clock primary input (PI), or an internal clock pin name. For an instance pathname, all the input pins of the instance are considered to be end points. For a clock PI, the data inputs of all non-transparent latches, flip-flops, and RAMs associated with the clock PI are considered to be end points. A pin pathname is considered to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and the internal clock pin is handled the same as a clock PI.

  Note

  For a 2-port latch with ports: Port1 = (D1, CLK1), Port2 = (D2, CLK2), and primary input clocks CLK1 and CLK2, the command “report_false_paths... -to CLK2” would report false paths with an end point at D2.

- -TO_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the end point of a false path by its associated clock pin pathname. Any pin pathname you specify with this argument is interpreted to be the source of an internal clock signal. The data inputs of all non-transparent latches, flip-flops, and RAMs associated with this clock are considered to be end points.

- {-THrough *through_node…*}…

  An optional, repeatable switch and repeatable string pair that specifies circuit node(s) through which the false path must pass. A valid through node is a pin or instance pathname. Clock PIs are not supported as through nodes and no attempt is made to determine if a pin pathname is that of an internal clock pin.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When you use multiple -Through arguments, their order, left-to-right, in the command string is considered to be the order in which a signal would pass through the nodes on the specified path. For example, the following false path specification:

```
add_false_paths -from A -through B C -through D E -to  F
```

specifies all paths that start from A, pass through either B or C, then pass through D or E, and end at F. If you do not use the -through argument when specifying a path, all paths from the specified starting points to the specified end points are considered to be false paths.

- -GAte *gate-id#* | *pin_pathname*

  An optional repeatable switch and repeatable integer or string that filters the report results based on gates or pins that you specify. The tool reports the false paths and multicycle paths that include the specified gates or pins as the definition points (-from, -to, or -through). If no such path is found, it locates one false path that includes *gate-id#* in either its intersection cone or effect cone. You can use multiple -Gate switches to specify additional pins or gates in a single command invocation.

  gate-id# — A repeatable integer that specifies the gate identification number. When you specify multiple gate ids, the command reports on any path that includes any of those gates as definition points.

  pin_pathname — A repeatable string that specifies the name of a pin within the design.When you specify multiple pin_pathnames, the command reports on any path that includes any of those pins as definition points.

- -X_Statistics [-path_count *int* | all] [-path_x_threshold *int*] [-to_gate_count {*int* | all}] [-to_gate_x_threshold *int*]

  An optional switch that reports the false or multicycle paths that have the most X masking from create_patterns or simulate_patterns after you enable the collection of X statistics with the set_timing_exceptions_handling command. The command reports the full path only when the path is not associated with the SDC file. Otherwise, the command reports only the SDC filename and line number of the path, as well as the number of cumulative Xs.

  -path_count {*int* | all} — An optional switch and integer or literal pair that specifies the number of paths to be reported. The default is to report up to 10 false or multicycle paths that have the greatest number of masking effects propagated to some to-points of the timing exception paths.

  int — An optional positive integer that specifies the number of paths for which the tool reports X statistics.

  all — An optional literal that specifies the tool reports X statistics for all paths that have X statistics.

  -path_x_threshold *int* — An optional switch and positive integer pair that specifies the minimum number of accumulated Xs needed for the command to report the path.

  -to_gate_count {*int* | all}— An optional switch and integer or literal pair that specifies that the tool reports some or all of the to-gates' X statistics, immediately following each false path, in order from the gate with the greatest number of maskings to the least. The to-gates are the false path defined points from using the -to option. When

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

the -to option is not specified, path to-gates are derived from the last -through option or the -from option, when the -through option is not specified.

int — An optional positive integer that specifies the number of to-gates for which the tool reports X statistics.

all — An optional switch that specifies the tool reports all to-gates with at least one masking.

-to_gate_x_threshold *int* — An optional switch and integer pair that specifies the minimum number of accumulated Xs needed for the command to report to-gates. When the -to_gate_x_threshold option is specified, and the -to_gate_count is not specified, the default is 10 gates.

By default, if the -to_gate_count option and the -to_gate_x_threshold option are both unspecified, the tool does not report any to-gates.

The -to_gate_count and the -to_gate_x_threshold reporting criteria is the AND operation of both options. For example, if "-to_gate_count 5" and "-to_gate_x_threshold 8" are specified, the tool reports the 5 to-gates that have at least 8 accumulated Xs.

- -DRC [C6 | C6-*n* | C22 | C22-*n*]

  An optional switch that reports false paths that are automatically added by the tool to handle DRC violations. The only DRC errors that this applies to are C6 and C22, for which the tool adds false paths to prevent simulation mismatches.

  - The "-drc c6" switch reports all false paths that result from C6.
  - The "-drc c6-1" switch reports false paths due only to C6-1.
  - The "-drc c22" switch reports all false paths that result from C22.
  - The "-drc c22-1" switch reports false paths due only to C22-1.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

Example 1

The following example displays all previously-defined false paths.

```
report_false_paths -all
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// False Path -from LINESTATE[1] LINESTATE[0] -to_clock MCLK
// False Path -from_clock UTMICLK[0] -to_clock MCLK
//
// ---------------------------------------------
// The false path below has an error.
// ---------------------------------------------
// False Path -from name1 -to name2
//
// ---------------------------------------------
// The false paths below have warnings.
// ---------------------------------------------
// False Path -from_clock S -to_clock s
// False Path -through I1 I2 I3 I4 I5 SIN CLK1 SEN
// False Path -from CLK1 -through U7/CP CLK1
//
// Total paths reported = 6
// Warning: The last 4 paths reported have 1 error and 3 warnings.
//          Use the command "report_false_paths -debug_error" to report
//          the causes of the errors/warnings.
// Warning: False path defined through all primary inputs, including all
//          clocks, which prohibits at-speed testing. Deleted clocks from
//          this list.
// Warning: There exists a false path that prohibits at-speed testing of
//          clock domain /CLK1.
//          This may result in lower test coverage.
```

Example 2

The following example displays the false paths with violations.

report_false_paths -violations

```
// Warning: False Path -to_clock clk5 -setup -hold prohibits at-speed \
        testing of clock domains /clk5
// Warning: False Path -from_clock aa -to_clock bb -setup -hold did not \
        find gates aa bb
// Warning: False Path -from_clock op[1] -to_clock op[2] -setup -hold \
        did not find gates op[1] op[2]
// Warning: False Path -from_clock ix1565/A -to_clock \
        result_add_add_0_ix139/Y -setup -hold has no sensitizable \
        combinational path between start and end points.
// False path debug summary: 0 errors, 4 warnings
```

Example 3

The following example displays the false paths with violations excluding those due to invalid design objects.

report_false_paths -violations -exclude invalid_design_objects

```
// Warning: False Path -to_clock clk5 -setup -hold prohibits at-speed \
        testing of clock domains /clk5
// Warning: False Path -from_clock ix1565/A -to_clock \
        result_add_add_0_ix139/Y -setup -hold has no sensitizable \
        combinational path between start and end points.
// False path debug summary: 0 errors, 2 warnings
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

The following example displays the false paths that start at /my_design/d_in[6]. The second path defined by the user results in an error because there is no sensitized path between the gates. Using the -debug_error option provides additional information about the error.

report_false_paths -verbose -from /my_design/d_in[6]

```
// False path -from /my_design/d_in[6] ( /reg_d_1_7_/Q (81)
      /reg_d_1_6_/Q (73) ) -to /my_design/u25 ( /my_design/u25/Y (130) )
//
// ----------------------------------------
// The false path below has an error.
// ----------------------------------------
// Error: False path -from /my_design/d_in[6]( ) -to /my_design/xray[5]( )
//
// Total reported false paths = 2
// Warning: The last 1 path reported has an error.
//          Use the command "report_false_paths -debug_error" to report
//          the cause of the error.
```

report_false_paths -verbose -from /my_design/d_in[6] -debug_error

```
//  Error: False Path -from /my_design/d_in[6] -to /my_design/xray[5] \
      has no sensitizable combinational path between start and end points.
//  False path debug summary: 1 errors, 0 warnings
```

Example 5

The following example first shows that the pin "/top/inst1/A" is in a timing exception intersection cone. It then displays one false paths or multicycle path whose intersection cone includes the gate top/inst1/A:

set_gate_report -timing_exceptions on

report_gate top/inst1/A

```
//  /top/inst1 (46)  NOR
//       A     I  (In/2)  42-/top/G5/Z
//       B     I  (In/2)  41-/top/G6/Z
//       Z     O  (In/2)  49-/top/G13/A
```

report_false_paths -gate top/inst1/A

```
// pin 'top/inst1/A' maps to gate 'top/inst1' (46)
// Multicycle Path -from_clock ipg_clk_source -to_clock adc_clk -start
      -cycles 2 -setup
```

Example 6

The following shows false path reporting after turning on reporting of SDC filenames and line numbers:

read_sdc timing.sdc
set_timing_exceptions_handling -line_no on
report_false_paths

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  False Path -from_clock cmem_fclki cmem_fclkin -to_clock p_clk
//    spm_pci_clk -setup  (filename = timing.sdc, line 406)
//  False Path -from_clock p_clk spm_pci_clk -to_clock cmem_fclki
//    cmem_fclkin -setup  (filename = timing.sdc, line 408)
...
//  Total reported false paths = 177
```

Example 7

The following shows the commands to enable X statistics functionality and create ATPG patterns. It reports the 3 false paths with the greatest masking effect, in order starting with the path that has the most Xs. For each path, the tool reports the X statistics for the top 5 top-gates, which observe the most masking, in order starting from the gate that has the most Xs.

read_sdc mod.sdc
set_timing_exceptions_handling -x_statistics on
create_patterns
report_false_paths -x_statistics -path_count 3 -to_gate_count 5

```
//  Total 249 paths out of 350 paths produced Xs.
//  ----  ----------  ------------  --------------------
//  Path  Type        Number of Xs  File and line number
//  ----  ----------  ------------  --------------------
//     1  false path       1572811  mod.sdc (line 32761)
//  ----  ----------  ------------  --------------------
//      ----------------------------------------------------  ------------
//      Gate (Id)                                             Number of Xs
//      ----------------------------------------------------  ------------
//      '/u_lspipe/dcu_valid_dc3_reg/' (253337)                      3162
//      '/clk_gate_list_phys_id_q_reg_0_/' (218650)                  2656
//      '/clk_gate_list_phys_id_q_reg_3_/' (218653)                  2684
//      '/clk_gate_list_phys_id_q_reg_2_/' (218652)                  1886
//      '/clk_gate_list_phys_id_q_reg_1_/' (218651)                  1006
//     Only the first 5 to-gates with most masking are reported (21524 out
//     of total 26000 to-gates captured X).

//  ----  ----------  ------------  --------------------
//     2  false path       1107367  mod.sdc (line 32761)
//  ----  ----------  ------------  --------------------
//      ----------------------------------------------------  ------------
//      Gate (Id)                                             Number of Xs
//      ----------------------------------------------------  ------------
//      '/clk_gate_list_phys_id_q_reg_0_/' (218650)                  2656
//      '/clk_gate_list_phys_id_q_reg_1_/' (218651)                  2312
//      '/clk_gate_list_phys_id_q_reg_3_/' (218653)                  2110
//      '/clk_gate_list_phys_id_q_reg_5_/' (218655)                  1118
//      '/clk_gate_list_phys_id_q_reg_2_/' (218652)                  1092
//     Only the first 5 to-gates with most masking are reported (14547 out
//     of 16000 to-gates captured X).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  ----  ----------  ------------  --------------------
//     3  false path        113454  mod.sdc (line 30390)
//  ----  ----------  ------------  --------------------
//      ---------------------------------------------------  ------------
//      Gate (Id)                                            Number of Xs
//      ---------------------------------------------------  ------------
//      '/u_coreoverpowerdown/cluster_id_reg_reg_3_/' (134183)      779
//      '/fwd_data_sync_reg_40_/' (170295)                          654
//      '/clken_reg/' (230954)                                      548
//      'gen_resync_0__gen_resync_r_u_resync/' (180823)             545
//      '/state_reg_0_/' (238777)                                   520
//      Only the first 5 to-gates with most masking are reported (1359 out
//      of 1800 to-gates captured X).
//  ----  ----------  ------------  --------------------
//  Only the first 3 paths producing the most Xs were reported.
```

Example 8

In the following example, the tool has automatically added false paths due to reconverging combinational logic that can cause a C6 race condition.

set_simulation_options -C6_mask_races on

```
//  2 false paths are added for C6 violation.
```

report_false_paths

```
//  False Path -from CLK -to 36 -hold -DRC
//  False Path -from CLK -to 35 -hold -DRC
//
//  Total reported false paths = 2
```

Example 9

The following shows a C22-1 violation and then reports, using verbose mode, the false path resulting from the C22-1 violation:

report_drc_rule c22-1

```
//  Warning: Pulse-always clock '/clk2' (2) interacts with clock domain \
   '/clk1' (1) and may cause simulation mismatches. Use \
   "report_false_paths -verbose -drc C22-1" to see the clock \
   interactions. (C22-1)
```

report_false_path -verbose -drc c22-1

```
//  False Path -from clk1 ( '/cellB0/' (60) '/U_0/' (57) '/U_1/' (58) )
   -to clk2 ( '/cellB2/' (41) '/cellB1/' (50) ) -setup -hold -auto -DRC
```

Related Topics

add_false_paths

delete_false_paths

delete_multicycle_paths

read_sdc

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_multicycle_paths

set_timing_exceptions_handling

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_fault_sites

Context: dft -edt, patterns -scan

Mode: analysis

Depending on the fault model, displays path definitions, bridge entries, or UDFM entries from the internal list.

Usage

UDFM Usage

report_fault_sites {[-ALl | instance_name] [-Udfm_type *name*] [-CEll *name*]
[-MOdule *name*] [-FAult *name*]}
| {[-All | *instance_name*] [-UNDEFined_cells] | [-DEFined_cells]}
[{> | >>} *file_pathname*]

Bridge Usage

report_fault_sites
{[-All | *net_pathname_pair* | *bridge_name…* | *net_pathname…* | *instance_name…*]
[-NAME | -SINGle] [-NET_pathname | -PIN_pathname] [-NOEQ] [-ATtribute] }
| {[-Histogram Weight [-Sets *number*]]}
| {[-Histogram {Layer | Type}]
[{> | >>} *file_pathname*]

Path Delay Usage

report_fault_sites [-All] [*path_name*] [-Path *gate_id_begin gate_id_end*]
[{> | >>} *file_pathname*]

Description

Depending on the fault model, displays path definitions, bridge entries, or UDFM entries from the internal list.

The report_fault_sites command displays the definitions in the tool's internal list for the specified faults. Fault definitions are loaded into the internal list with the read_fault_sites command.

For more information about bridge faults, refer to "The Static Bridge Fault Model" in the *Tessent Scan and ATPG User's Manual*.

For more information about path delay faults, refer to "Path Delay Test Set Creation" in the *Tessent Scan and ATPG User's Manual*.

For more information about UDFM faults, refer to "About User-Defined Fault Modeling" in the *Tessent Scan and ATPG User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

UDFM Specifics

The report_fault_sites command does not report restricted information about UDFM models generated by Tessent CellModelGen, which protects the IP inside of those models.

Arguments

- -All

  An optional switch that displays all fault definitions for the currently loaded paths, bridge, or UDFM entries. This is the default.

- *path_name*

  An optional string that specifies the name of a path whose definition you want to display. The path must be one defined in a path definition file that you previously loaded using the read_fault_sites command.

- -Path *gate_id_begin gate_id_end*

  An optional switch and two-integer triplet that specifies a particular path or portion of a path whose definition you want to display. Use this argument to report on paths that were not defined in a path definition file, and therefore were not loaded using the read_fault_sites command.

  The two integers specify two gate identification numbers that indicate the beginning and ending of the path. The path begins at *gate_id_begin* and ends at *gate_id_end*.

  The value of *gate_id_begin* or *gate_id_end* is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.

- *net_pathname_pair*

  A string pair that specifies the pathname to the nets or pins in a bridge entry. The first string specifies NET1, and the second string specifies NET2.

- *bridge_name*

  A repeatable string that specifies a bridge name.

- *net_pathname*

  A repeatable string that specifies a single net or pin pathname.

- *instance_name*

  A repeatable string that specifies an instance pathname.

- -ATtribute

  An optional switch that displays the attributes of the reported bridge entries. Attributes may include distance, weight, layer, and parallel_run.

- -NAME

  An optional switch that specifies all entered strings are bridge names.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -SINGle

  An optional switch that specifies all the entered strings are either a single net or instance. The tool searches for a net first. If a corresponding net cannot be found, the tool searches for a corresponding instance.

  If the string is an instance that contains a non-dominant net from a bridge entry, a fault is reported for each bridge entry associated with the instance.

- -NET_pathname

  An optional switch that reports net pathnames for bridge entries. This is the default.

- -PIN_pathname

  An optional switch that reports pin pathnames for bridge entries instead of net pathnames.

- -NOEQ

  An optional switch that reports only the representative fault information: the net pair and fault categories.

- -Histogram {Layer | Type | {Weight [-Sets *number*]}

  An optional switch and literal pair that displays a histogram of the number of equivalent classes of bridges based on either their layer, type, or weight attribute. When you specify a weight histogram, you can optionally include the -Sets switch to specify a number of sets into which to distribute the report data. Specifying "-sets 0" forces the tool to display an individual line for each bridge of different weight.

  The display for a weight histogram consists of one weight histogram per type of bridge fault, whereas a layer or type histogram is a single histogram.

- -Udfm_type *name*

  An optional switch and string pair that specifies the name of a UDFM fault type. Use this option to display the faults associated with the specified UDFM fault type. The name string is case insensitive and can include any number of asterisk (*) and question mark (?) wildcard characters.

- -CEll *name*

  An optional, repeatable switch and string pair that specifies the name of a library cell. Use this option to display all faults associated with a specified library cell. The name string is case insensitive and can include any number of asterisk (*) and question mark (?) wildcard characters.

- -MOdule *name*

  An optional switch and string pair that specifies the name of a module. Use this option to display all faults associated with the specified module. The name string is case insensitive and can include any number of asterisk (*) and question mark (?) wildcard characters.

- -FAult *name*

  An optional switch and string pair that specifies the name of a UDFM-defined fault. Use this option to display all faults associated with the specified name. The name string is case

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

insensitive and can include any number of asterisk (*) and question mark (?) wildcard characters.

- -UNDEFined_cells

  An optional switch that reports cells without a valid UDFM definition that were added with a previous read_fault_sites or add_fault_sites command.

- -DEFined_cells

  An optional switch that reports cells with a valid UDFM definition that were added with a previous read_fault_sites or add_fault_sites command.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

The following path delay example reads in (loads) the path information and then displays that data:

set_fault_type path_delay
read_fault_sites /user/design/pathfile
report_fault_sites

```
PATH "path0" =
   PIN /I$6/Q + ;
   PIN /I$35/B0 + ;
   PIN /I$35/C0 + ;
   PIN /I$1/I$650/IN + ;
   PIN /I$1/I$650/OUT - ;
   PIN /A_EQ_B + ;
END ;
PATH "path1" =
   PIN /I$6/Q + ;
   PIN /I$35/B1 + ;
   PIN /I$35/C1 + ;
   PIN /I$1/I$649/IN + ;
   PIN /I$1/I$649/OUT - ;
   PIN /I$5/D - ;
END ;
```

Example 2

The following bridge example reads the bridge information in the bridge fault definition file, */user/design/bridgefile*, then displays the data for bridge entries added to the fault site list:

set_fault_type bridge read_fault_sites /user/design/bridgefile

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//
//  Load bridge entries from file "/user/design/bridgefile".
//
//  5 bridge entries were read from file "/user/design/bridgefile".
//  2 bridge entries were skipped due to unknown net name.
//  3 bridge entries were added to the fault site list.
```

report_fault_sites

```
BRIDGE {
  NET1 = "/G5";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  NAME = "U7.BRIDGE";
}
BRIDGE {
  NET1 = "/G4";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  NAME = "U5.BRIDGE";
}
BRIDGE {
  NET1 = "/G11";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  NAME = "U0.BRIDGE";
}
```

Example 3

The following example displays the attribute data for the “U0.BRIDGE” bridge entry:

report_fault_sites U0.BRIDGE -name -attribute

```
BRIDGE {
  NET1 = "/G11";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  WEIGHT = 1.0000;
  DISTANCE = 4.93827000e-06 um;
  PARALLEL_run = 6.07500000e-05 um;
  X_COORDINATE = 152500;
  Y_COORDINATE = 739500;
  LAYER = "metal1";
  NAME = "U0.BRIDGE";
}
```

Example 4

The following example displays the UDFM fault definition f01 defined in UDFM type my-udfm-sa-faults of instance /top/inst1:

report_fault_sites /top/inst1 -udfmtype my-udfm-sa-faults -fault f01

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
UDFM {
   Version:1;
   UdfmType("my-udfm-sa-faults") {
      Instance("/top/inst1") {
         // Source reference:
         // File: /home/JohnDoe/myadded.udfm, Line: 145
         Fault("f01") {
            Test { StaticFault{"Z":0;} Conditions{"D":0;"E":1; "F":-;} }
            Test { StaticFault{"Z":1;} Conditions{"D":1;"E":-; "F":0;} }
         }
      }
   }
}
```

If there is more than one UDFM types used, the following command reports all f01 faults on instance /top/inst1:

report_fault_sites /top/inst1 -Fault f01

```
UDFM {
   Version:1;
   UdfmType("my-udfm-sa-faults") {
      Instance("/top/inst1") {
         // Source reference:
         // File: /home/JohnDoe/myadded.udfm, Line: 145
         Fault("f01") {
            Test { StaticFault{"Z":0;} Conditions{"D":0;"E":1; "F":-;} }
            Test { StaticFault{"Z":1;} Conditions{"D":1;"E":-; "F":0;} }
         }
      }
   }
   UDFMType("intra-cell") {
      Instance("/top/inst1") {
         // Source reference:
         // File: /home/JohnDoe/myadded.udfm, Line: 418
         Fault("f01") {
            Test { StaticFault{"Z":0;} Conditions{"D":0;"E":1; "F":-;} }
            Test { StaticFault{"Z":1;} Conditions{"D":1;"E":-; "F":0;} }
         }
      }
   }
}
```

report_fault_sites -all

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
UDFM {
   Version:1;
   UdfmType("my-udfm-sa-faults") {
      Cell("AOX") {
         // Source reference:
         // File: /home/JohnDoe/myadded.udfm, Line: 145
         Fault("f01") {
            Test { StaticFault{"Z":0;} Conditions{"D":0;"E":1; "F":-;} }
            Test { StaticFault{"Z":1;} Conditions{"D":1;"E":-; "F":0;} }
         }
      }
   }
}
```

Example 5

The following example shows how to display a list of cells both with and without valid UDFM definitions:

report_fault_sites -defined_cells

```
//  Report of cells defined by UDFM
//
//  Cell Name  Instances
//  ---------  ---------
//  XNOR2X18         518
//  XNOR2X3          139
//  XOR2X17          387
//  AND2X35          913
//  AO112X27           4
//  AO12X27          203
//  AO22X27          219
//  AO312X27          39
//  AO31X27            5
//  ---------  ---------
```

report_fault_sites -undefined_cells

```
//  Report of cells not defined by UDFM
//
//  Cell Name  Instances
//  ---------  ---------
//  cnhlsx4v          52
//  df1qx1v          834
//  df4sqx1v           7
//  u_noti            68
//  u_mux2          7261
//  u_mux3            20
//  u_mux4           398
//  ---------  ---------
```

Related Topics

add_fault_sites

add_faults

delete_fault_sites

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_fault_sites

set_fault_type

write_fault_sites

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_faults

Context: dft -edt, dft -test_points, patterns -scan

Mode: analysis

Displays fault information from the current fault list.

Usage

Stuck/Transition Usage:

```
report_faults [-Stuck_at {01 | 0 | 1}] [-NO_Subclass]
    {[-Class fault_class[.fault_subclass]] [{pin_pathname | gate_id | combined} ...] [-NOEQ]
    [instance_name ...] [-Hierarchy integer] [-Min_count integer] [-VErilog | -VHdl]
     [-CEll_name]
    [-CLOCK_Domains {ALL | clock_pathname ...} [-NO_EQUivalent_clocks]
      [-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]]
    [-CAPture_procedures {ALL | capture_procedure_name ...}] [-DElay_data]
    [-TIMING_CRITICAL]}
    | {[-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls]}
    | {-UNlisted }
    [-FRom pin ...][-THrough pin ...][-TO pin ...][-FROM_Clock pin ...][-TO_Clock pin ...]
    [-stop_at {sequential_elements | scan_cells | ports_only}]
    [-fanout_off_path_stop {on | off }] [-fanin_off_path_stop { on | off }]
    [(> | >>) filename]
```

UDFM Usage:

```
report_faults [-NO_Subclass]
    {[-Class fault_class[.fault_subclass]] [instance_pathname] [-Hierarchy integer] [-NOEQ]
    [-DETAILS]
      [-CLOCK_Domains {ALL | clock_pathname ...} [-NO_EQUivalent_clocks]
        [-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]]
    [-DElay_data] [-TIMING_CRITICAL]}
    | {[-UDFM_Type name]
    { [-bridge { -id bridge_id | -net1 net_pathname [-net2 net_pathname] }]
    | [-INstance name] [-CELL name] [-MODule name] [-SOurce]}}
    | {-UNlisted}
    [-FRom pin ...][-THrough pin ...][-TO pin ...][-FROM_Clock pin ...][-TO_Clock pin ...]
    [-stop_at [ sequential_elements | scan_cells | ports_only }]
    [-fanout_off_path_stop {on | off }] [-fanin_off_path_stop { on | off }]
    [(> | >>) filename]
```

Bridge Usage:

```
report_faults {[-Histogram Weight [-Sets number]]}
    | {[-Histogram {Layer | Type}]}
    | {[net_pathname_pair | {bridge_name ... -NAME}
      | {net_pathname ... -SINGle} | {instance ... -SINGle}]
    [-Class fault_class[.fault_subclass]] [-NET_pathname | -PIN_pathname] [-NOEQ]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

[-NO_Subclass] [-VErilog | -VHdl] [-ATtribute] [-SCAN_Enable] [-CLOCK_Cones]
[-IO] [-ASYnchronous_controls]}
| {[-CLOCK_Domains {ALL | *clock_pathname* ...} [-NO_EQUivalent_clocks]
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]]
[-CAPture_procedures {ALL | *capture_procedure_name* ...}]}
[(> | >>) *filename*]

Toggle/Iddq Usage:

report_faults [-Stuck_at {01 | 0 | 1}] [-NO_Subclass]
{[-Class *fault_class*[.*fault_subclass*]] [{*pin_pathname* | *gate_id* | combined} ...] [-NOEQ]
[*instance_name* ...] [-Hierarchy *integer*] [-Min_count *integer*] [-VErilog | -VHdl]
[-CEll_name]
[-CLOCK_Domains {ALL | *clock_pathname* ...} [-NO_EQUivalent_clocks]]
[-CAPture_procedures {ALL | *capture_procedure_name* ...}]}
| {[-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls]}
| {-UNlisted}
[-FRom *pin* ...][-THrough *pin* ...][-TO *pin* ...][-FROM_Clock *pin* ...][-TO_Clock *pin* ...]
[-stop_at {sequential_elements | scan_cells | ports_only}]
[-fanout_off_path_stop {on | off }] [-fanin_off_path_stop { on | off }]
[(> | >>) *filename*]

Path Delay Usage:

report_faults [-Both | -Rise | -Fall] [-VErilog | -VHdl]
[-Class *fault_class*[.*fault_subclass*]] [*object_pathname*] [-NO_Subclass]

Power-Aware Options (applicable only after you have loaded CPF/UPF power data):

[[ON_domains] | [-OFf_domains] | [-POWer_domains {*domain_name* ...}]]
[-ISolation_cells] [-LEvel_shifters] [-REtention_cells]

Description

The report_faults command displays all the faults you added to the fault list using the add_faults or read_faults command. You can use the optional arguments to narrow the focus of the report to only specific stuck-at or transition faults that occur on a specific object in a specific class. If you do not specify any arguments, report_faults displays information on all the known faults.

By default, the report_faults command ignores the constraint values implied in the data path when reporting faults in the data path. However, the tool still considers the constraint values for clock path faults.

For more information about UDFM faults, refer to "About User-Defined Fault Modeling" in the *Tessent Scan and ATPG User's Manual*.

The power-aware options report_faults based on power domains, power features, or both, after you have loaded a CPF or UPF file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The report_faults command displays the following columns of information for each fault:

- **fault value** — The fault value may be either 0 (for stuck-at-0 or "slow-to-rise" transition faults) or 1 (for stuck-at-1 or "slow-to-fall" transition faults).
- **fault code** — A code name that indicates the lowest level fault class assigned to the fault (see Table 6-10). This also includes a fault sub-class code when available (see Fault Sub-Classes in the *Tessent Scan and ATPG User's Manual*).
- **fault site** — The pin pathname of the fault site.
- **cell name** (optional) — The name of the cell corresponding to the fault (displayed only if you include the -Cell_name argument).

Use the -Hierarchy option to display a hierarchical summary of the selected faults. The summary identifies the number of faults in each level of hierarchy whose level does not exceed the specified level number. You can further specify the hierarchical summary by using the -Min_count option, which specifies the minimum number of faults that must be in a hierarchical level before they are displayed.

You can choose to display either collapsed or uncollapsed faults by using the set_fault_mode command. Also, if your fault data is large, it may be more appropriate to use the write_faults command and then read the file contents.

Arguments

- -Class *fault_class*[.*fault_subclass*]

  An optional switch and literal pair that specifies the fault class or fault class/sub-class to display. The following table lists the valid fault class codes and their associated fault class names. "Fault Sub-Classes" in the *Tessent Scan and ATPG User's Manual* lists the valid fault sub-class names. Use the code when specifying a fault class or sub-class.

  When you have previously issued the "report_statistics -detailed_analysis" command, report_faults can report specific AU.TC and AU.PC faults by pin pathname or by gate ID. For more information, refer to "Example 9" on page 1861.

Table 6-10. Fault Class Codes and Names

Fault Class Code	Fault Class Name	Fault Class Coverage
FU	Full	TE+UT
TE	TEstable	DT+PD+AU+UD
DT	DETEcted	DS+DI+DR
DS	DET_Simulation	
DI	DET_Implication	
DR	DET_Robust (Path Delay Testing only)	

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 6-10. Fault Class Codes and Names (cont.)

Fault Class Code	Fault Class Name	Fault Class Coverage
DF	DET_Functional (Path Delay Testing only)	
PD	POSDET	PT+PU
PU	POSDET_Untestable	
PT	POSDET_Testable	
AU	Atpg_untestable	
UD	UNDetected	UC+UO
UC	UNControlled	
UO	UNObserved	
UT	UNTestable	UU+TI+BL+RE
UU	UNUsed	
TI	TIed	
BL	Blocked	
RE	Redundant	

- -NO_Subclass

  An optional switch that prevents the display of fault sub-classes after the fault code. The default is to display fault sub-classes when the tool has analyzed them.

- -UNlisted

  An optional switch that displays unlisted fault data. If there are multiple grayboxed instances in the design, this switch reports accumulated fault numbers.

- -FRom *pin* …

  An optional switch and repeatable string pair that sets the start pin or pins of the cone of logic the tool uses to select faults to report. When specified, the tool performs forward cone tracing from the specified pin or pins until it encounters a stopping point. You can specify one or more instance names rather than pin names. When you specify an instance name with -from, the tool uses its output pins when defining the cone.

- -THrough *pin* …

  An optional switch and repeatable string pair that sets the through pin or pins of the cone of logic the tool uses to select faults to report. When specified, the tool performs forward and backward cone tracing from the specified pin or pins until it encounters a stopping point in both directions. You can specify one or more instance names rather than pin names. When you specify an instance name with -through, the tool uses its output pins when defining the cone.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -TO *pin* …

  An optional switch and repeatable string pair that sets the end pin or pins of the cone of logic the tool uses to select faults to report. When specified, the tool performs backward cone tracing from the specified pin or pins until it encounters a stopping point. You can specify one or more instance names rather than pin names. When you specify an instance name with -to, the tool uses its input pins when defining the cone.

  Note

  When you use -from, -to, and -through with other switches, the selected faults are the intersection of the cones traced for all specified switches.

- -FROM_Clock *pin* ...

  An optional switch and repeatable string pair that sets the launch clock pin or pins of the cone of logic the tool uses to select faults to report. When specified, the tool performs forward cone tracing from the output ports of the state elements launched by the specified clock pin or pins until it encounters a stopping point.

- -TO_Clock *pin* ...

  An optional switch and repeatable string pair that sets the capture clock pin or pins of the cone of logic the tool uses to select faults to report. When specified, the tool performs backward cone tracing from the data ports of the state elements captured by the specified clock pin or pins until it encounters a stopping point.

- -stop_at {sequential_elements | scan_cells | ports_only}

  An optional switch and literal pair that sets the stop condition for tracing of the cone of logic the tool uses to select faults to report. By default, the traced logic cone includes only combinational logic, so cone tracing stops at any sequential elements. When you specify “scan_cells”, the cone includes non-scan cells and tracing stops only at scan cells. When you specify “ports_only”, the cone includes any sequential elements and tracing stops only at primary inputs and outputs.

- -fanout_off_path_stop {on | off}

  An optional switch and literal pair that specifies an additional stop condition when tracing the fan-in fault cone backward and reaching a multiple fanout gate G. When this switch is on, tracing stops at gate G unless the cone already includes all of the fanout gates of G. The default is off. You can use a separate command, set_fault_selection_options, to change the default of this stop condition.

- -fanin_off_path_stop {on | off}

  An optional switch and literal pair that specifies an additional stop condition when tracing the fanout fault cone forward and reaching a multiple fan-in gate G. When this switch is on, tracing stops at gate G unless the cone already includes all of the fan-in gates of G. The default is off. You can use a separate command, set_fault_selection_options, to change the default of this stop condition.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Stuck_at 01 | 0 | 1

  An optional switch and literal pair that specifies the stuck-at or transition faults you want to display, according to the following table:

	01 (default)	**0**	**1**
Stuck-At	stuck-at-0 stuck-at-1	stuck-at-0	stuck-at-1
Transition	slow-to-rise slow-to-fall	slow-to-rise	slow-to-fall

- *pin_pathname*

  An optional repeatable string that specifies the pathname of a top-level pin or a library model pin whose faults you want to display.

- *gate_id*

  An optional repeatable integer that specifies the gate identification number of the gate whose faults you want to display.

- combined

  An optional keyword string that specifies the display of faults due to combined tied cells for the AU.TC and AU.PC fault sub-classes.

- *object_pathname*

  An optional repeatable string that specifies a list of pins, instances, or delay paths whose faults you want to display.

- -Hierarchy *integer*

  An optional switch and integer pair that specifies the maximum hierarchy level for which you want to display a summary of the faults. However, the reported number of faults depends on other switches, such as -class.

- -Min_count *integer*

  An optional switch and integer pair that you use with the -Hierarchy option to specify the minimum number of faults that must be in a hierarchical level to display the hierarchical summary. The default is 1.

- -NOEQ

  - For stuck-at, transition, toggle, iddq, and udfm faults only, an optional switch that turns off the display of "EQ" as the fault class for any equivalent faults; the fault class displayed is then that of the representative fault. When you do not specify this switch, the tool displays an "EQ" as the fault class for any equivalent faults. This switch is meaningful only when the set_fault_mode command is set to Uncollapsed. For more information about representative and equivalent faults, see "Fault Collapsing" in the *Tessent Scan and ATPG User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- For bridge faults only, an optional switch that writes only the representative bridge fault information: the net pair and fault classes of each corresponding dominant net. The tool does not report physical data.

- -DETAILS

  An optional switch that reports the fault list including the cell names, TCA values, and impact values. The reported impact value of the fault is the average of the impact values of its test alternatives.

- -Both| -Rise | -Fall

  An optional switch that specifies which faults to display for each path already added via the read_fault_sites command. Use these switches for path delay faults only.

  -Both - An optional switch that specifies to display both the slow-to-rise and slow-to-fall faults. This is the default.

  -Rise - An optional switch that specifies to display only the slow-to-rise faults.

  -Fall - An optional switch that specifies to display only the slow-to-fall faults.

- -VErilog | -VHdl

  An optional switch that outputs the fault paths in either Verilog or VHDL syntax, rather than using the existing netlist-independent format.

- -CEll_name

  An optional switch that lists the name of the corresponding circuit element (Tessent Cell library cell, Verilog primitive, primary input or primary output) for each fault as listed in Table 6-11. Cell names for equivalent faults are based on the cells associated with the equivalent faults, not the cells associated with the representative faults. This switch is valid for the stuck-at, transition, toggle and IDDQ fault models only.

Table 6-11. Name Conventions Used by report_faults -Cell_name

Corresponding Circuit Element	Listed Cell Name
Tessent Cell library cell	Name of the Tessent Cell library cell
Supported Verilog primitive (for a list, see Table 4-1 in the *Tessent Cell Library Manual*)	Name of the Verilog primitive
Primary input or output	“primary_input” or “primary_output”

- -CLOCK_Domains {ALL | *clock_pathname*…}

  An optional switch and literal or repeatable string pair that specifies a list of clocks that the tool uses to decide the faults it reports.

  The argument choices are as follows:

  ALL — A literal that specifies all the clocks in the design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

clock_pathname — A repeatable string that specifies a particular clock. The faults reported are those determined by the add_faults command to be in the *clock_pathname* domain.

- -NO_EQUivalent_clocks

  An optional switch that prevents the -Clock_domains switch from reporting faults in equivalent clock domains.

- -EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains

  An optional switch used with -Clock_domains to instruct the tool to exclude the inter-clock faults for synchronous clocks. When you specify this switch, the tool considers only faults within clock domains when reporting faults by clock domain.

- -CAPture_procedures {ALL | *capture_procedure_name*…}

  An optional switch and literal or repeatable string pair that specifies a list of enabled named capture procedures and directs the tool to report faults that are potentially detectable by any of the specified procedures. The argument choices are as follows:

  ALL — A literal that specifies all enabled named capture procedures.

  capture_procedure_name — A repeatable string that specifies a particular enabled named capture procedure.

  See the set_capture_procedures command description for information about enabling named capture procedures.

- -DElay_data

  An optional switch that reports the fault list including the slacks of the faulty site. The reported slack is for information only and cannot be loaded back into the system.

- -TIMING_CRITICAL

  An optional switch that reports timing-critical faults.

  Note

  In order to use the -DElay_data or -TIMING_CRITICAL switches, you must have previously loaded an SDF file containing timing information. See "Timing-Aware ATPG" in the *Tessent Scan and ATPG User's Manual* for complete information.

- -SCAN_Enable

  An optional switch that reports faults that fan out to the select lines of multiplexers in the scan path. For this switch, a multiplexer is either a MUX simulation primitive or a nonprimitive type multiplexer composed of AND and OR gates. In essence, this switch reports all faults that are in the fan-in cone of local scan enable signals and are dominated by them.

- -CLOCK_Cones

  An optional switch that reports all faults in the clock cone. The clock cone is the intersection of the fan-in of the sequential element clock ports and the fanout of the clock sources. This

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

switch considers any sequential elements, such as flops, latches, RAMs, and ROMs, not just scan cells.

- -IO

  An optional switch that specifies to report faults with the following conditions:

  - Only controlled by PIs — To be acted on by the -IO switch, a PI either must not be defined as a clock or, if defined as clock (or read/write control), must be constrained off during capture.
  - Only observed by POs

- -ASYnchronous_controls

  An optional switch that reports all faults that only fan out to Set/Reset ports of state elements and RAMs.

- -Histogram Weight [-Sets *number*] -Histogram {Layer | Type}

  An optional switch and literal, or literal and optional switch and integer, group that displays a histogram of the number of equivalent classes of bridges based on either their layer or type, or their weight attributes. When you specify a weight histogram, you can optionally include the -Sets switch to specify a number of sets into which to distribute the report data. Specifying "-sets 0" forces the tool to display an individual line for each bridge of different weight.

  The display for a weight histogram consists of one weight histogram per type of bridge fault, whereas a layer or type histogram is a single histogram.

- > *filename*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *filename*.

- >> *filename*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *filename*.

- *net_pathname_pair*

  An optional string pair that specifies the bridge fault site. The first string specifies NET1, and the second string specifies NET2.

- *bridge_name*

  An optional repeatable string that specifies the bridge name.

- *net_pathname*

  An optional repeatable string that specifies a single net pathname.

- *instance_name*…

  An optional repeatable string that specifies an instance pathname.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -ATtribute

  An optional switch that displays the attributes for bridge entries. Attributes may include distance, weight, layer, and parallel_run.

- -NAME

  An optional switch that specifies all entered strings are bridge names.

- -SINGle

  An optional switch that specifies all the entered strings are either a single net or instance. The tool searches for a net first. If it cannot find a corresponding net, the tool searches for a corresponding instance.

  If the string specifies an instance that contains a non-dominant net from a bridge entry, the tool reports a fault for each bridge entry associated with the instance.

- -NET_pathname

  An optional switch that reports net pathnames for bridge entries. This is the default.

- -PIN_pathname

  An optional switch that reports pin pathnames for bridge entries instead of net pathnames.

- *instance_pathname* …

  An optional repeatable string that specifies a list of instance names you want to display. You can specify a full or partial instance path. If you specify a partial path, the tool matches instances that include the partial pathname.

- -Udfm_Type *name*

  An optional switch and string pair that specifies the name of a UDFM fault type. Use this option to display the faults associated with a specified UDFM fault type. The name string is case insensitive and may include any number of asterisk (*) and question mark (?) wildcard characters.

- -CEll *name*

  An optional, repeatable switch and string pair that specifies the name of a library cell. Use this option to display all faults associated with a specified library cell. The name string is case insensitive and may include any number of asterisk (*) and question mark (?) wildcard characters.

- -MOdule *name*

  An optional switch and string pair that specifies the name of a module. Use this option to display all faults associated with a specified module. The name string is case insensitive and may include any number of asterisk (*) and question mark (?) wildcard characters.

- -INstance *name*

  An optional switch and string pair that specifies the name of an instance. Use this option to display all faults associated with a specified instance. The name string may include any number of asterisk (*) and question mark (?) wildcard characters.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -bridge { -id *bridge_id* | -net1 *net_pathname* [-net2 *net_pathname*] }

  An optional switch with a corresponding set of switch and value pairs that specifies to report on UDFM bridge faults.

  The -id *bridge_id* switch and value pair reports on the bridge corresponding to the specified bridge identifier as provided in the loaded UDFM file(s).

  The -net1 *net_pathname* -net2 *net_pathname* switch and value pairs report on bridges between net1 and net2. If you do not supply -net2, it reports on all nets that are bridging with net1.

  Note

  The -net1 and -net2 switches require real net names in order to function. Specifying the GROUND or POWER keywords does not return results.

- -SOurce

  An optional switch that outputs the UDFM filename and line number where the fault is defined.

Power-Aware Options (applicable only after you have loaded CPF/UPF power data)

- -ON_domains

  An optional switch that reports faults on all power-on domains.

- -OFF_domains

  An optional switch that reports faults on all power-off domains.

- -Power_domains {*domain_name* …}

  An optional switch and repeatable string that reports faults on the specified power domains.

- -Isolation_cells

  An optional switch that reports faults on all isolation cells.

- -Level_shifters

  An optional switch that reports faults on all level-shifters.

- -REtention_cells

  An optional switch that reports faults on all retention cells.

Examples

Example 1

The following example modifies the current fault population in the tool with the contents of the external fault file, *my_detected_faultlist*, then displays the faults:

```
set_system_mode analysis
add_faults -all
read_faults my_detected_faultlist -merge
report_faults
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example displays information for the stuck-at-0 fault on pin /mult_8/Y, including the name of the cell associated with the fault:

report_faults -stuck_at 0 /mult_81/Y -cell_name

```
//  type   code   pin_pathname  (cell_name)
//  ----   ----   ------------------------------------
//    0    DS     /mult_8/Y     (xnor2)
```

Example 3

The following example reports on a fault associated with a primary input:

report_faults -stuck_at 1 /ip_scan_en -cell_name

```
//  type   code   pin_pathname   (cell_name)
//  ----   ----   ------------------------------------
//    0     AU    /ip_scan_en    primary_input
```

Example 4

The following example shows a fault list with the static test slack and the actual minimum test slack. Only DS faults contain the actual minimum test slack data.

report_faults -delay_data

```
// type    code       pin_pathname
//                                          static_slack   actual_min_slack
// ----    ----      ------------------    -------------  -----------------
//  0      DS        /CORE_i/IRQA_B/A      10.1010        12.3000
//  1      DS        /CORE_i/IRQA_B/B      20.1480        25.4820
//  0      AU.BB     /CORE_i/IRQA_B/C      12.1890        -
//  1      AU.BB     /CORE_i/IRQA_B/C      18.8770        -
```

Example 5

The following example displays information for the stuck-at-0 fault on pin /mult_8/Y plus all isolation cells:

report_faults -stuck_at 0 /mult_81/Y -isolation_cells

Example 6

The following example reports unlisted faults:

report_faults -unlisted

```
class detections collapsed_fault_count uncollapsed_fault_count
----- ---------- --------------------- -----------------------
UC       0               1252                2079
UC.EAB   0                452                 543
DS       1              69873               87232
DS       2              12873               21432
DS       3               9752               11974
DS       4+              4487                6293
AU.BB    0               8708               10046
AU       0               2374                3782
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 7

The following example displays the fault data for DS class faults associated with library cells that start with “mux” or “an”:

report_faults -class ds -cell mux* -cell an*

```
// Command output

FaultInformation {
   version : 1;
   FaultType (UDFM) {
      FaultList {
         Format : Identifier, Class;
         UdfmType ("intra-cell-bridges_1") {
            Cell ("MUX2") {
               Instance ("/top/i1/i4") {
                  "F2", DS;
               }
            }
            Cell ("AND3") {
               Instance ("/top/i1/i456") {
                  "F2", DS;
                  "F4", DS;
                  "F5", DS;
                  "F7", DS;
               }
            }
         }
      }
   }
}
```

Example 8

The following example shows the default format of the fault report after you specify the fault type as UDFM using the set_fault_type command:

report_faults

```
Class UdfmType    Fault  instance
----- ----------- ----- ----------
UC    my-added-sa f01   /top/i1/i4
DS    my-added_sa f02   /top/i1/i4
DS    intra-cell  f01   /top/i3
AU.BB intra-cell  f02   /top/i3
```

Adding the -source switch to the command extends the report with references to the definition sources:

report_faults -source

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Class UdfmType    Fault instance   File                        Line
----- ----------- ----- ---------- -------------------------- ----
UC    my-added-sa f01   /top/i1/i4 /home/JohnDoe/myadded.udfm   14
DS    my-added_sa f02   /top/i1/i4 /home/JohnDoe/myadded.udfm  145
DS    intra-cell  f01   /top/i3    /home/JohnDoe/intra.udfm    217
AU.BB intra-cell  f02   /top/i3    /home/JohnDoe/intra.udfm     67
```

Example 9

The following examples shows how to report specific AU.TC and AU.PC faults, and AU faults due to combined tie cells:

report_statistics -detailed_analysis # you must first run this command

report_faults -class au.tc /tap_i/tap_ctrl_i/reg_enable #specified using a pin pathname

```
type    code    pin_pathname
----   ------   ------------------------------------
  0    AU.TC    /pad_i1/tdo_buf/tri_pin/A
  0    EQ       /pad_i1/tdo_buf/ix68/Y
  1    EQ       /pad_i1/tdo_buf/ix68/A
  1    AU.TC    /pad_i1/tdo_buf/tri_pin/A
  1    EQ       /pad_i1/tdo_buf/ix68/Y
  0    EQ       /pad_i1/tdo_buf/ix68/A
  0    AU.TC    /tdo
  0    EQ       /pad_i1/tdo_buf/tri_pin/Y
  1    AU.TC    /tdo
  1    EQ       /pad_i1/tdo_buf/tri_pin/Y
  0    AU.TC    /pad_i1/tdo_buf/tri_pin/E
```

report_faults -class au.tc 1196 #specified using a gate ID.

```
type    code    pin_pathname
----   ------   ------------------------------------
  0    AU.TC    /pad_i1/tdo_buf/tri_pin/A
  0    EQ       /pad_i1/tdo_buf/ix68/Y
  1    EQ       /pad_i1/tdo_buf/ix68/A
  1    AU.TC    /pad_i1/tdo_buf/tri_pin/A
  1    EQ       /pad_i1/tdo_buf/ix68/Y
  0    EQ       /pad_i1/tdo_buf/ix68/A
  0    AU.TC    /tdo
  0    EQ       /pad_i1/tdo_buf/tri_pin/Y
  1    AU.TC    /tdo
  1    EQ       /pad_i1/tdo_buf/tri_pin/Y
  0    AU.TC    /pad_i1/tdo_buf/tri_pin/E
```

report_faults -class au.pc test_en #specified using a pin pathname

```
type    code    pin_pathname
----   ------   ------------------------------------
  1    AU.PC    /test_en
  1    AU.PC    /core_i/test_design_i/senmux/ix9/S0
  0    AU.PC    /core_i/test_design_i/senmux/ix9/A0
  1    AU.PC    /core_i/test_design_i/senmux/ix9/A0
```

report_faults -class au.pc 35 #specified using a gate ID

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
type    code    pin_pathname
----   ------   ------------------------------------
  1    AU.PC    /test_en
  1    AU.PC    /core_i/test_design_i/senmux/ix9/S0
  0    AU.PC    /core_i/test_design_i/senmux/ix9/A0
  1    AU.PC    /core_i/test_design_i/senmux/ix9/A0
```

report_faults -class AU.TC combined #combined tied cells

```
type    code    pin_pathname
----   ------   ------------------------------------
  0    AU.TC    /edt_i/edt_decompressor_i/U238/Y
  0    AU.TC    /edt_i/edt_compactor_i/decoder1/U13/B0
  0    AU.TC    /edt_i/edt_compactor_i/decoder3/U17/B0
  0    AU.TC    /edt_i/edt_compactor_i/U76/Y
  0    EQ       /edt_i/edt_compactor_i/U76/A0
  0    EQ       /edt_i/edt_compactor_i/U76/A1
  0    EQ       /edt_i/edt_compactor_i/decoder3/U14/Y
  0    EQ       /edt_i/edt_compactor_i/compactor3/U9/A0
  0    AU.TC    /edt_i/edt_compactor_i/U77/Y
  0    EQ       /edt_i/edt_compactor_i/U77/A0
  0    EQ       /edt_i/edt_compactor_i/U77/A1
  0    EQ       /edt_i/edt_compactor_i/decoder4/U19/Y
  0    EQ       /edt_i/edt_compactor_i/compactor4/U17/A1
  0    AU.TC    /edt_i/edt_compactor_i/decoder4/U15/B0
  0    AU.TC    /edt_i/edt_compactor_i/U68/Y
  0    EQ       /edt_i/edt_compactor_i/U68/A0
  0    EQ       /edt_i/edt_compactor_i/U68/A1
```

Example 10

The following example shows the format of the fault report when you specify the fault type as UDFM and you have enabled multiple detections:

report_faults

```
//  Class  Detections  UdfmType    Fault  Instance
//  -----  ----------  ----------  -----  ---------------
//  DS     2           intra-cell  F1     ip_inst/pio_i/i_42/inst
//  DS     1           intra-cell  F2     ip_inst/pio_i/i_42/inst
//  DS     2           intra-cell  F3     ip_inst/pio_i/i_42/inst
//  DS     3+          intra-cell  F4     ip_inst/pio_i/i_42/inst
//  DS     1           intra-cell  F5     ip_inst/pio_i/i_42/inst
//  DS     1           intra-cell  F6     ip_inst/pio_i/i_42/inst
```

Example 11

The following example shows the format of the fault report when you specify to display faults from a maximum of two levels of hierarchy:

report_faults -hierarchy 2

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /clk_ctrl_clk_testshell                48
//  /clk_ctrl_clk_testshell/gated_ccb_inst4            6
//  /clk_ctrl_clk_testshell/gated_ccb_inst2            6
//  /clk_ctrl_clk_testshell/i_ipoHold_10275            4
//  /clk_ctrl_clk_testshell/gated_ccb_inst1           10
//  /clk_ctrl_clk_testshell/antiskew_clock_inv         4
//  /clk_ctrl_clk_testshell/antiskew_1                 6
//  /clk_ctrl_clk_testshell/gated_ccb_inst3           12
//  /ip_inst                                     240366
//  /ip_inst/dsp_periph_em_ccb_i                    960
//  /ip_inst/siox_i                                8336
//  /ip_inst/dio_i                                25218
```

Example 12

The following examples show how to report UDFM bridge-related faults.

You can specify the UDFM bridge in one of three ways:

- By the bridge ID as Provided in the corresponding UDFM file (in this example, “B1”):

report_faults -bridge -id B1

```
//  Class  UdfmType              Fault        NetA       NetB
//  -----  --------------------  -----------  ---------  ----------------------------------
//  UU     interconnect_bridges  B1_A0_dom_B  L81_N0     .../bypass_logic_i/edt_scan_in[38]
//  UU     interconnect_bridges  B1_A1_dom_B  L81_N0     .../bypass_logic_i/edt_scan_in[38]
//  AU     interconnect_bridges  B1_B0_dom_A  L81_N0     .../bypass_logic_i/edt_scan_in[38]
//  AU     interconnect_bridges  B1_B1_dom_A  L81_N0     .../bypass_logic_i/edt_scan_in[38]
```

- By supplying one net the bridge connects to:

report_faults -bridge -net1 i1/coreA/coreB/u_c2/Q[2]

```
//  Class  UdfmType              Fault         NetA              NetB
//  -----  --------------------  ------------  ---------------   ------------------------
//  DS     interconnect_bridges  B24_A0_dom_B  i1/coreA/N359     i1 coreA/coreB/u_c2/Q[2]
//  DS     interconnect_bridges  B24_A1_dom_B  i1/coreA/N359     i1/coreA/coreB/u_c2/Q[2]
//  AU     interconnect_bridges  B24_B0_dom_A  i1/coreA/N359     i1/coreA/coreB/u_c2/Q[2]
//  AU     interconnect_bridges  B24_B1_dom_A  i1/coreA/N359     i1/coreA/coreB/u_c2/Q[2]
//  DS     interconnect_bridges  B10_A0_dom_B  i1/coreA/N1483    i1/coreA/coreB/u_c2/Q[2]
//  DS     interconnect_bridges  B10_A1_dom_B  i1/coreA/N1483    i1/coreA/coreB/u_c2/Q[2]
//  AU     interconnect_bridges  B10_B0_dom_A  i1/coreA/N1483    i1/coreA/coreB/u_c2/Q[2]
//  AU     interconnect_bridges  B10_B1_dom_A  i1/coreA/N1483    i1/coreA/coreB/u_c2/Q[2]
```

- By supplying both nets the bridge connects:

report_faults -bridge -net1 i1/coreA/coreB/u_c2/Q[2] -net2 i1/coreA/N1483

```
//  Class  UdfmType              Fault         NetA            NetB
//  -----  --------------------  ------------  --------------  ------------------------
//  DS     interconnect_bridges  B10_A0_dom_B  i1/coreA/N1483  i1/coreA/coreB/u_c2/Q[2]
//  DS     interconnect_bridges  B10_A1_dom_B  i1/coreA/N1483  i1/coreA/coreB/u_c2/Q[2]
//  AU     interconnect_bridges  B10_B0_dom_A  i1/coreA/N1483  i1/coreA/coreB/u_c2/Q[2]
//  AU     interconnect_bridges  B10_B1_dom_A  i1/coreA/N1483  i1/coreA/coreB/u_c2/Q[2]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

add_faults

analyze_fault

delete_faults

read_fault_sites

read_faults

read_cpf

read_upf

report_power_data

report_testbench_simulation_options

set_fault_mode

set_fault_sampling

set_fault_selection_options

set_fault_subclass_analysis

set_fault_type

write_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_feedback_paths

Context: all contexts

Mode: all modes

Prerequisites: You can use this command only after the tool performs the learning process, which happens immediately after flattening a design to the simulation model. Flattening occurs when you first attempt to exit Setup mode or when you issue the create_flat_model command.

Displays a report of currently identified feedback paths.

Usage

report_feedback_paths [-All | *loop_id#…*] [{> | >>} *file_pathname*]

Description

The report_feedback_paths command displays feedback paths the tool identified during the last circuit learning process. By default, the command displays a textual report of all currently identified feedback paths and their identification numbers. Issuing the command with the identification numbers of specific paths of interest limits the display to just those paths.

Note

Feedback paths include, by default, any duplicated gates.

Arguments

- -All

  An optional switch that specifies to report all currently identified feedback paths. This is the default.

- *loop_id#*

  An optional, repeatable, non-negative integer that specifies the identification number of a particular feedback path to report. The tool assigns the numbers consecutively, starting with 0.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of file_pathname.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example enters analysis mode (which flattens the simulation model and performs the learning process), reports the identification numbers of all learned feedback paths, and then reports one of the feedback paths (loop #1):

set_system_mode analysis
report_feedback_paths

```
Loop#=0, feedback_buffer=26, #gates_in_network=5
   INV   /I_956__I_582/ (51)
   PBUS  /I_956__I_582/N1/ (96)
   ZVAL  /I_956__I_582/N1/ (101)
   INV   /I_956__I_582/ (106)
   TIEX  /I_956__I_582/ (26)
Loop#=1, feedback_buffer=27, #gates_in_network=5
   INV   /I_962__I_582/ (52)
   PBUS  /I_962__I_582/N1/ (95)
   ZVAL  /I_962__I_582/N1/ (100)
   INV   /I_962__I_582/ (105)
   TIEX  /I_962__I_582/ (27)
```

report_feedback_paths 1

Related Topics

report_loops

set_loop_handling

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_flattener_rules

Context: all contexts

Mode: setup, analysis

Displays either a summary of all the flattening rule violations or the data for a specific violation.

Usage

report_flattener_rules [*rule_id* [{*occurence_id* | -Verbose}]] [{> | >>} *file_pathname*]

Description

The report_flattener_rules command displays the following information for a specific violation:

- Rule identification number
- Current number of rule failures
- Violation handling

You can use the set_flattener_rule_handling command to change the handling of the net, pin, and gate rules.

Arguments

- *rule_id*

  A literal that specifies the flattening rule violation for which you want to display information. The flattening rule violations and their identification literals are divided into the following three groups: net, pin, and gate rules violation IDs.

  - Net flattening violations are described in sections “FN1” through “FN9”.
  - Pin flattening violations are described in sections “FP1” through “FP13”.
  - Gate flattening violations are described in sections “FG3” through “FG8”.

- *occurence_id*

  A literal that specifies the identification of the exact flattening rule violation (the occurrence) for which you want to display information. For example, you can analyze the second occurrence of the FG4 rule by specifying the rule_id and the occurence_id, FG4 2. The tool assigns the occurrences of the rules violations as it encounters them; you cannot change either the rule identification number or the ordering of the specific violations.

- -Verbose

  A switch that displays the following for each flattening rule:

  - Rule identification number
  - Number of failures of each rule
  - Current handling status of that rule

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

  - Brief description of that rule

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example shows the summary information of the FG3 rule:

report_flattener_rules fg3

```
/  FG3: fails=2 handling=warning/noverbose
```

Related Topics

set_flattener_rule_handling

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_gates

Context: all contexts

Mode: setup, analysis

Displays the netlist information and simulation results for the specified gates and the simulation results for the specified user-defined ATPG functions. This command requires a flattened netlist before use.

Usage

report_gates {***function_name*** | ***pin_or_net_pathname*** | ***gate_id#*** | ***instance_name***}...
{-Value_trace [*position*] [-TRANsition] [-Endpoints] [-STOP_ambiguous | -GO_One_ambiguous | -GO_All_ambiguous]}

report_gates {-Type *gate_type*}... [-SUMmary]}

report_gates {{-Sequential_test_depth | -0Control_depth | -1Control_depth | -Observe_depth} *depth_number*}...

report_gates {-Path {***pin_or_net_pathname*** | ***gate_id#***} {***pin_or_net_pathname*** | ***gate_id#***}}

report_gates {-GLitch_masked_cells} {-x_capture_cells *number*}

report_gates {[-Endpoints] [-COnstraints] [-Forward | -Backward] {***pin_or_net_pathname*** | ***gate_id#***} [-min_static_slack] ...}

All of the above forms can redirect their output with: [{> | >>} *file_pathname*]

Description

The report_gates command displays the netlist information for either the design-level or the primitive-level gates you specify. You designate the gate by its gate identification number (ID#), a pathname of a pin connected to a gate, an instance name (design level only), or a gate type. You can specify a design cell by the pathname of a pin on the design cell. If you use a gate ID# or gate type, the command always reports the primitive-level gate. A flattened netlist is required before running this command.

The pin_or_net_pathname and instance_name arguments support regular expressions, which may include any number of '*' or '?' wildcard characters embedded in the pathname string. The '*' character matches any sequence of characters (including none) in a name, and the '?' character matches any single character. If a wildcard name is specified, the command searches for matching instance names from the top library cell level down to the primitive gates.

The design-level gate report has the following format:

```
instance_name     cell_type
   input_pin_name    I   (data)   pin_or_net_pathname...
...
   output_pin_name   O   (data)   pin_or_net_pathname...
...
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The data field displays the pin information as selected by the "set_gate_report" command.

The primitive-level report has the following format:

```
instance_name   (gate_ID#)     gate_type
   input_pin_name    I   (data)   gate_ID#-pin_or_net_pathname...
...
   output_pin_name   O   (data)   gate_ID#-pin_or_net_pathname...
...
```

When you provide the name or identification number (ID#) of a user-defined ATPG function, the command reports simulation data of the function. The data reported is that specified by the Parallel_pattern option of the set_gate_report command.

The function report has the following format:

```
function_name   (function_ID#)     function_type
   input_pin_name    I   (data)   gate_ID#-pin_or_net_pathname...
...
   output_pin_name   O   (data)   gate_ID#-pin_or_net_pathname...
...
```

The list associated with the input and output pin names indicate the pins to which they connect. For the primitive level, this also includes the gate index number of the connecting gate and only includes the pin pathname if one exists at that point. There is a limitation on reporting gates at the design level. If some circuitry inside the design cell is completely isolated from other circuitry, the command only reports the circuitry associated with the pin pathname.

You can also report the fanin or fanout cone of a specified gate. The endpoints of a cone are defined as the primary inputs, primary outputs, tied gates, rams, roms, flip-flops, and latches. All gates reported are at the primitive level.

You can change the output of the report_gates command by using the set_gate_report command.

Note

The tool does not explicitly simulate the EDT logic during scan loading and capture cycles. Therefore, report_gates cannot report simulation values for the EDT logic during these cycles. This is true regardless of the option set with the set_gate_report command.

Backslashes (\), Periods (.) and Spaces in Hierarchical Names

When reading hierarchical names into its internal database, the tool hides backslashes (\), converts periods (.) to slashes (/), and ignores spaces. For example, the tool reads the cell name, "FFH.\E/MY_SCAN_FF .SCAN_FF5" in a WGL pattern, as "FFH/E/MY_SCAN_FF/SCAN_FF5", and stores the latter in its internal database. To report on this cell, you must reference it by the name that is stored internally.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The difference between the internal hierarchical name and the name in the WGL file of the preceding example does not alter the correctness of the tool. That is, the output pattern file remains consistent with the original netlist.

The following example reports on a Verilog gate named “FFH.\E/MY_SCAN_FF .SCAN_FF5”.

report_gates FFH/E/MY_SCAN_FF/SCAN_FF5

```
//  /FFH/E/MY_SCAN_FF/SCAN_FF5 FD1SF
//      D       I  /FFH/E/MY_SCAN_FF/GATE00/Z
//      A       I  /C-4/Z
//      SI      I  /FFG/SO
//      CP      I  /INPBUF1/Z
//      B       I  /C-5/Z
//      Q       O  /GATE20/C
//      QN      O
//      SO      O  /C-7/A
```

Reporting Power Data of Gates

When you issue the read_cpf or read_upf command, power data is loaded into the tool. The report_gates command enables power data reporting by default and shows the power on (PON) or power off (POFF) status as well as the power domain of the gate. For example:

read_upf *upf_file*

report_gates /lp_case_si_rst_sms/ati_rst_sync/sync_r/U3/Udff

```
//  /lp_case_si_rst_sms/ati_rst_sync/sync_r  sync3msfqxss1ul
//      SDI    I(PON)  /vl_sms_lp_case_si_sms_proc_sms_1_stp/
U_lp_case_si_sms_proc_bist/uu5/Z
//      D      I(PON)  /lp_case_si_rst_sms/ati_rst_sync/sync_buf/Z
//      SEN    I(PON)  /se
//      CLK    I(PON)  /lp_case_si_rst_sms/ati_rst_sync/uu1/Z
//      Q      O(PON)  /lp_case_si_rst_sms/ati_rst_sync/uu2/A  /
lp_case_si_rst_sms/and_r/B
//      in power domain PD_P2
```

Reporting on the First Input of a Gate

The report_gates command provides a shortcut to display data on the gate connected to the first input of the previously-reported gate. This lets you quickly and easily trace backward through circuitry. To use report_gates in this manner, first report on a specific gate and then enter:

b

The following example shows how to use report_gates and B commands to trace backward through the first input of the previously reported gate.

report_gates 26

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /u1/inst__565_ff_d_1__13 (26)  BUF
//      "I0"   I  269-
//      "OUT"  O  268-  75-
```

b

```
//  /u1/inst__565_ff_d_1__13 (269)  LA
//      "S"    I  14-
//      "R"    I  145-
//      SCLK   I  4-/clk
//      D      I  265-/u1/_g32/X
//      ACLK   I  2-/scan_mclk
//      SDI    I  20-/u1/inst__565_ff_d_0__dff/Q2
//      "OUT"  O  26-  27-
```

b

```
//  /u1/inst__565_ff_d_1__13 (14)  TIE0
//      "OUT"  O  269-  268-
```

If you include an integer argument with “b”, the trace is backward through the specified input. For example, “b 3” traces backward through the third input of the previously specified gate. When using integer arguments with the B and F commands in the tool, all arguments must be given at the primitive level.

For pins that are not at the library cell boundary (pins internal to the model), the pin name is enclosed in double quotation marks (“”). The following example displays this issue.

set_gate_level primitive

report_gates /I_20/I_226/q

```
//  /I_20/I_226  dffsr
//      clk    I  (HX)  /I_20/I_225/out
//      d      I  (X)   /I_20/I_222/out
//      pre    I  (H1)  /PRE
//      clr    I  (H1)  /CLR
//      q      O  (X)   /I_16/i0  /I_23/I_221/i0  /I_6/i0
//      qb     O  (X)
```

set_gate_level primitive

```
//  Creating schematic for 5 instances (1 was compacted).
```

report_gates /I_20/I_226/q

```
//  /I_20/I_226 (12)  BUF
//      "I0"   I  (0)  39-
//      q      O  (X)  16-/I_16/i0  31-/I_23/I_221/i0
                      17- /I_6/i0
```

b

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /I_20/I_226 (39)  DFF
//      "S"    I  (LX)  26-
//      "R"    I  (LX)  23-
//      clk    I  (HX)  20-/I_20/I_225/out
//      d      I  (X)   36-/I_20/I_222/out
//      "OUT"  O  (0)   12-  13-
//      MASTER  cell_id=1  chain=c1  group=g1 invert_data=FFFF
```

Reporting on the First Fanout of a Gate

Similar to tracing backward through circuitry, you can also use a shortcut to trace forward through the first fanout of the previously reported gate. To use report_gates in this manner, first report on a specific gate and then enter:

f

The following example shows how to use Report Gate and F commands to trace forward through the first fanout of the previously reported gate.

report_gates 269

```
//  /u1/inst__565_ff_d_1__13 (269)  LA
//      "S"    I  14-
//      "R"    I  145-
//      SCLK   I  4-/clk
//      D      I  265-/u1/_g32/X
//      ACLK   I  2-/scan_mclk
//      SDI    I  20-/u1/inst__565_ff_d_0__dff/Q2
//      "OUT"  O  26-  27-
```

f

```
//  /u1/inst__565_ff_d_1__13 (26)  BUF
//      "I0"   I  269-
//      "OUT"  O  268-  75-
```

f

```
//  /u1/inst__565_ff_d_1__13 (268)  LA
//      "S"    I  14-
//      "R"    I  145-
//      BCLK   I  1-/scan_sclk
//      "D0"   I  26-
//      "OUT"  O  24-  25-
```

If you include an integer argument with “f”, the trace is forward through the specified fanout of the previously reported gate. For example, “f 2” traces forward through the second fanout of the previously specified gate. When using integer arguments with the B and F commands in the tool, all arguments must be given at the primitive level.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

When reporting gates, be aware that a reported X may not have been captured. For example, when the tool uses scan chain masking, it records the actual measured value for each cell only in the unmasked, selected scan chain in a compactor group; the rest of the scan chains in the group are masked, so the tool records all Xs for their scan cells. When not using scan chain masking, the tool records an X for a scan cell if it is made unmeasurable as a result of the actual occurrence of an X in the corresponding cell of a different scan chain in the same compactor group. For more information, see "Understanding Scan Chain Masking in the Compactor" in the *Tessent TestKompress User's Manual*.

When using the tool to report on RAM or ROM gates, the report_gates command displays the RAM and ROM data that describes their behavior. You can use the set_gate_report command to customize the content of the reported data. For example, assume you add_faults on a RAM pin, then generate multiple load patterns. If you follow that with a set_gate_report command, specifying the Parallel_pattern argument, the report_gates output shows for the specified pattern, the address and data written to the RAM. A hyphen (-) separates the information for each cycle. The following example shows how this information is displayed for an 8-bit output named "OUT" and a 4-cycle pattern.

```
"OUT" O (S[0]=00000011-S[0]=00001001-S-SX) 5353- 5354- 5355-
                                           5356- 5357- 5358-
                                           5359- 5360-
```

In the first cycle, the pattern writes 00000011 to address 0, in the second, it writes 00001001 to address 0. The pattern does not write anything in the third cycle. The fourth cycle is a "don't care" cycle.

The RAM and ROM simulation primitives are the same as the library primitives with the outputs being OUT gates in the RAM/ROM fanout list. The report_gates command gives RAM behavior summary information at the end of the displayed data. The report displays messages similar to the following:

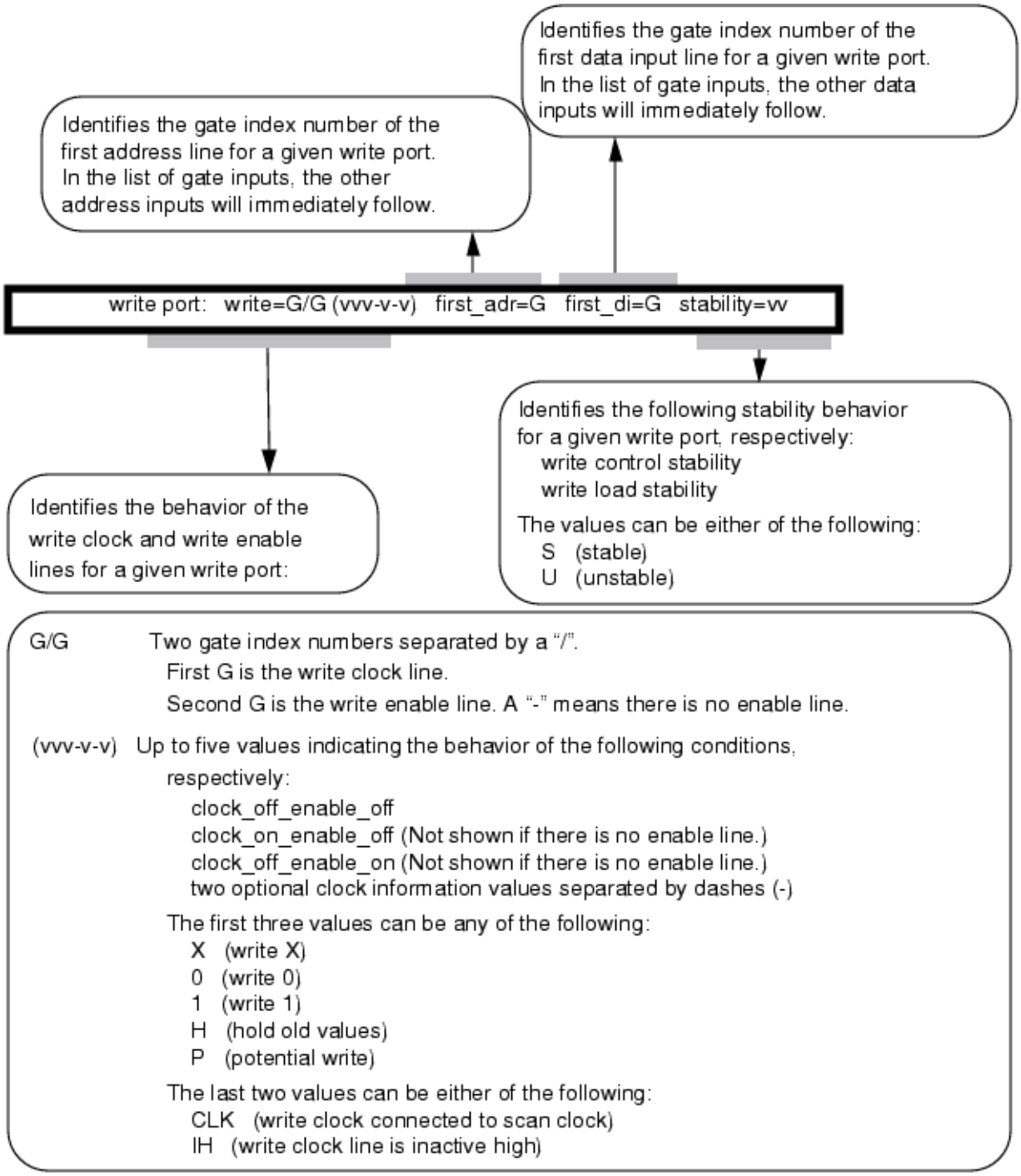
```
write port: write=G/G (vvv-v-v) first_adr=G   first_di=Gstability=vv
read port: read=G/G (vvv-v-v)   first_adr=G   first_do=Gstability=vv
Test behavior: stability=vvvv   tiex-X=v      read_only=v
               multiple_load=v/v  seq_transparent=v/v
Contention Behavior: write_write=v/v   read_read=vread_write=v/v
```

A RAM is classified as a read-only RAM if all three of the following conditions are met:

- its contents cannot be disturbed by any procedure except test setup;
- its set, reset and write ports must hold at an inactive state in the capture window;
- it has an initialization file defined in the Tessent Cell library.

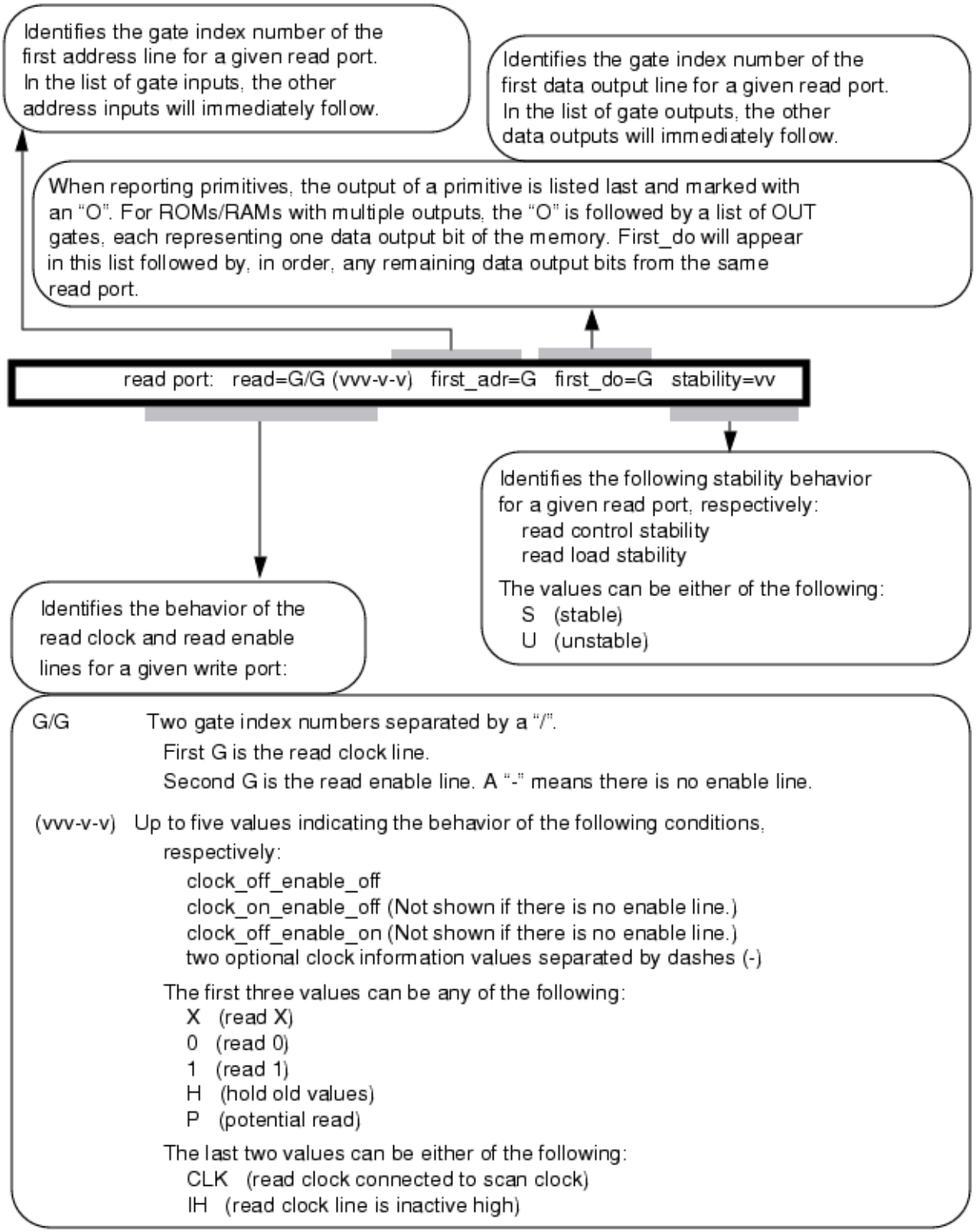
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following describes the fields for the write port message line:



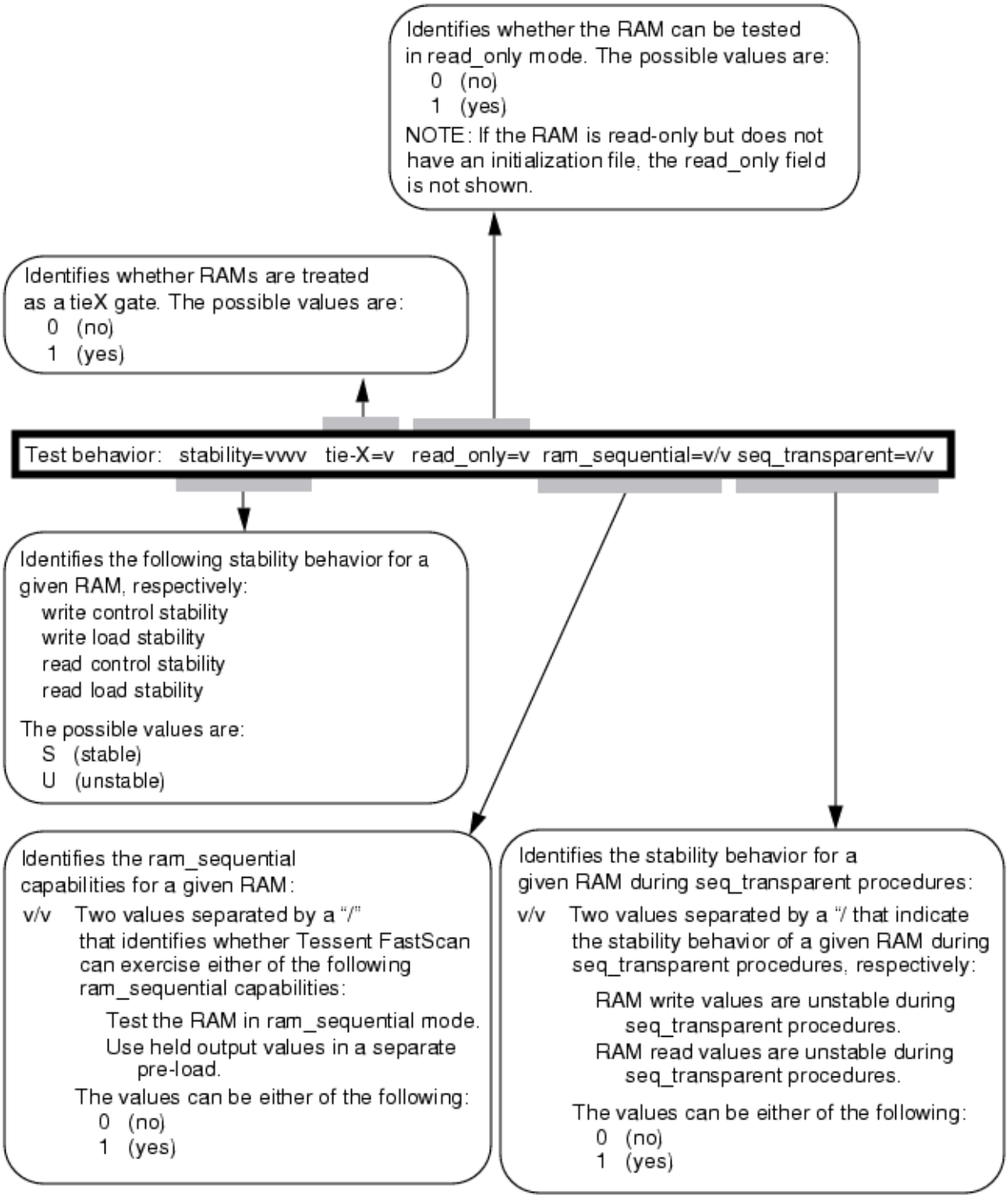
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following describes the fields for the read port message line:



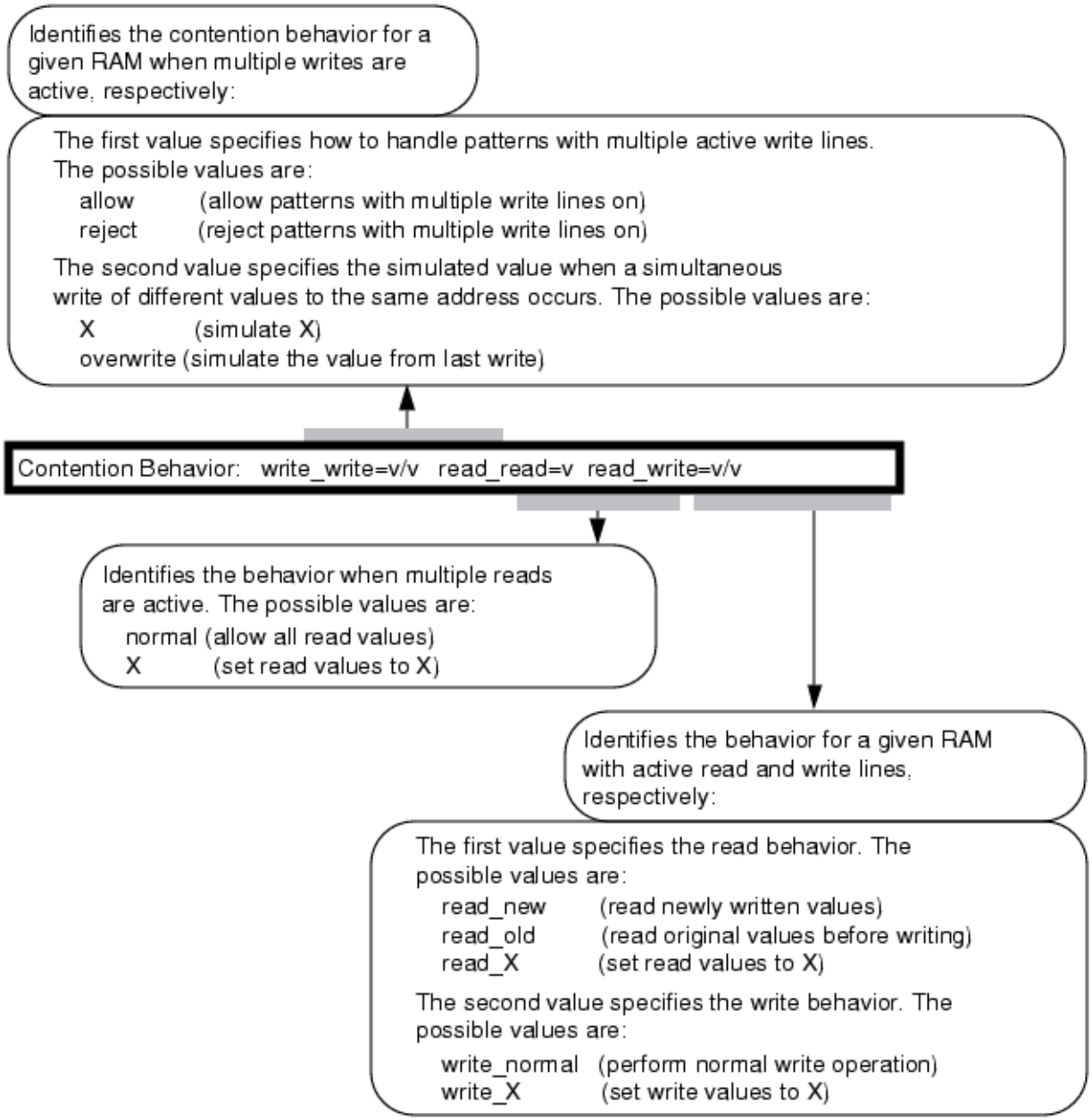
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following describes the fields for the Test behavior message line:



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following describes the fields for the Contention Behavior message line:



Arguments

The following four arguments list the four methods for naming the objects on which you want the tool to display information. You can use any number of the four argument choices, in any order.

- ***function_name***

  A repeatable string that specifies a user-defined ATPG function.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ***pin_or_net_pathname***

  A repeatable string that specifies the pathnames of pins or nets in the design netlist. You may use wildcard characters to match multiple pin or net pathnames.

  For a hierarchical pathname, the display includes information describing how that pathname maps to the driving design level pin(s) and gate(s) for which data is displayed.

- ***gate_id#***

  A repeatable integer that specifies the gate identification numbers of the objects. The value of the *gate_id#* argument is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.

  The tool also accepts gate_pin IDs specified in the following format:

  ```
  'gate_id:port_index'
  ```

- ***instance_name***

  A repeatable string that specifies the hierarchical pathname of an instance of a library cell within the design. If a valid library instance pathname is given when in primitive level, all pins on that library cell are reported. When in primitive level, *instance_name* may also be the pathname of a primitive instance.

- -Value_trace [*position*] [-TRANsition] [-Endpoints] [-STOP_ambiguous | -GO_One_ambiguous | -GO_All_ambiguous]

  An optional switch that specifies to trace backward from the output pin of the gate to the source of a value. The -Value _trace option only supports the primitive report level; therefore, all arguments must be specified at the primitive level.

  position

  An optional value that specifies which bit to trace on for multi-bit data. For example, if a clock signal is 0X0 for a shift cycle, you can specify the tracing is on the second bit (X). The default is to use position 0 (the first bit).

  -TRANsition

  An optional switch that specifies to trace backward to the source of the transition of a pair of values. When using the *position* option, *position* specifies the first bit of the transition.

  -Endpoints

  An optional switch that specifies to report only endpoints that control the specified gate. The result of using this option also depends on the tracing mode. Only when using the -GO_All_ambiguous tracing mode are all endpoints reported.

  The following three optional switches let you choose the tracing mode of operation.

  -STOP_ambiguous

  An optional switch that specifies to stop if either input could cause the value on the output. This is the default setting if -go_one_ambiguous or -go_all_ambiguous is not specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

-GO_One_ambiguous

An optional switch that specifies to trace one input if multiple input could cause the output value. For example, if the output of an OR is X, input A is X, input B is X, then the source of X is represented by both A and B. The tool picks one controlling input, say B, and continues tracing input B.

If the source of X is represented by B. The tool continues tracing input B.

-GO_All_ambiguous

An optional switch that specifies to trace all inputs if multiple input could cause the output value.

- -Type {*gate_type*}

  An optional repeatable switch where *gate_type* is a name pair that specifies the gate types for which you want to display the gate information. Table 6-12 lists the supported types.

- -SUMmary

  An optional switch that reports only the number of gates found when using the -Type switch.

- -Sequential_test_depth *depth_number*

  An optional switch and integer pair that specifies for the tool to report gates with a sequential test depth greater than the specified *depth_number*. The sequential test depth is the greater of the control 0 and control 1 depths plus the observe depth for the primitive output. This is the minimum depth required to test stuck faults at the primitive output. This is also the number used to determine the maximum sequential test depth of the circuit.

- -0Control_depth *depth_number*

  An optional switch and integer pair that specifies for the tool to report gates with a control 0 depth greater than the specified *depth_number*. The control 0 depth is the minimum sequential depth required to set the primitive output to a 0.

- -1Control_depth *depth_number*

  An optional switch and integer pair that specifies for the tool to report gates with a control 1 depth greater than the specified *depth_number*. The control 1 depth is the minimum sequential depth required to set the primitive output to a 1.

- -Observe_depth *depth_number*

  An optional switch and integer pair that specifies for the tool to report gates with an observe depth greater than the specified *depth_number*. The observe depth is the minimum sequential depth required to observe the output of the primitive at either a primary output or scan cell.

- -Path {***pin_or_net_pathname*** | ***gate_id#***} {***pin_or_net_pathname*** | ***gate_id#***}

  An optional switch that reports on the path between the first ***pin_or_net_pathname*** or ***gate_id#*** and the second ***pin_or_net_pathname*** or ***gate_id#***. Specify all paths at the primitive level. Paths do not extend through sequential elements. The tool generates a report on each primitive in the path(s), in order of increasing ***gate_id#***.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -GLitch_masked_cells

  An optional switch that specifies for the tool to report the sequential elements that have been masked due to glitches on the set or reset port. It also reports the number of patterns for which the cells are masked. This is an example of the report:

```
//  30 state elements are masked due to set/reset port glitches,
which may impact the test coverage.
//  #-pat-cycles-masked  state-element
//  -------------------  ------------------------
//                    1  '/cm0507/cm2304/' (7956)
//                    1  '/cm0507/cm2305/' (7959)
```

- -x_capture_cells *number*

  An optional switch and integer pair to report the state elements that capture Xs. The command set_simulation_options enables X capture recording for the commands create_patterns and simulate_patterns. The optional integer limits the report to that *number* of state elements capturing the most Xs.

- -Forward {***pin_or_net_pathname*** | ***gate_id#***}...

  An optional switch that reports the fanout cone of the specified gate.

- -Backward {***pin_or_net_pathname*** | ***gate_id#***}...

  An optional switch that reports the fanin cone of the specified gate.

- -Endpoints [-Forward | -Backward] {***pin_or_net_pathname*** | ***gate_id#***}...

  An optional switch that reports only the endpoint of the cone.

  Note

  Immediately after -Endpoints, you must specify either -Forward or -Backward followed by the specified gate. When using -Endpoints or -COnstraints simultaneously, -Forward or -Backward followed by the specified gate need only be entered once.

- -COnstraints [-Forward | -Backward] {***pin_or_net_pathname*** | ***gate_id#***}...

  An optional switch that takes into account the effects of pin and cell constraints.

  For a gate whose output is constrained to a fixed value, the tools report only other gates whose output is also constrained. For a gate whose output is not constrained to a fixed value, the tools report only other gates whose outputs are not constrained. During backwards tracing, a mux input, which is always blocked by a constrained value on the select line of the mux, is never backtraced.

  Note

  Immediately after -Constraints, you must specify either -Forward or -Backward followed by the specified gate. When using -Endpoints or -COnstraints simultaneously, -Forward or -Backward followed by the specified gate need only be entered once.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -min_static_slack

  An optional switch that displays the static timing critical paths in Tessent Visualizer.

  Note

  Immediately before -min_static_slack, you must specify either -Forward or -Backward followed by the specified gate.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair used, at the end of the argument list, for appending to the contents of *file_pathname*.

Reference Tables

- These are the reference tables for gate types and clock port categories:
  - Table 6-12 lists all of the types of gates that you can specify with the -Type switch.
  - Table 6-13 lists the various categories of clock ports on state elements that can appear in report_gates output.

Table 6-12. Reportable Gate Types

gate_type	Description
BUF	buffer
INV	inverter
AND	and
NAND	inverted and
OR	or
NOR	inverted or
XOR	exclusive-or
NXOR	inverted exclusive-or
DFF	D flip-flop, same as _dff library primitive
LA	latch, same as _dlat library primitive
PI	primary input
PO	primary output
TIE0	tied low
TIE1	tied high

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 6-12. Reportable Gate Types (cont.)

gate_type	Description
TIEX	tied unknown
TIEZ	tied high impedance
HIST	histogram of each primitive type used
TSD	tri-state driver, first input is active high enable line
BUS	tri-state bus
ZVAL	Z converter gate, converts Z to X
WIRE	undetermined wired gate
MUX	2-way multiplexer, first line is select line
SW	switch gate, first input is active high enable line
PBUS	pulled bus gate, where the second input is the pulled value
OUT	memory model gate, created for each read data bit
RAM	random access memory
ROM	read only memory
XDET	X detector, gives 1 when input is X
ZDET	Z detector, gives 1 when input is Z
TLA	transparent latch
STLA	seq_transparent latch
STFF	seq_transparent flip-flip
FB_BUF	internal gate type to break combinational loops in the design
MASTER	the primary memory element of a scan cell
SLAVE	an independently-clocked scan cell memory element; resides in the scan chain path.
SHADOW	non-scan flip-flop that has the D input connected to the Q output of a scan flip-flop
COPY	a memory element that lies in the scan chain path and can contain the same (or inverted) data as the associated master or slave element in the scan cell

Table 6-13. Clock Port Categories

Category	Description
IL	inactive low

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 6-13. Clock Port Categories (cont.)

Category	Description
IH	inactive high
AHS	active high slave
ALS	active low slave

For more information about using the clock port options, refer to "The ATPG Analysis Option" on page 3227.

Examples

Example 1

The following example displays the simulated values of the gate and its inputs at the primitive level:

set_system_mode analysis
set_gate_report error_pattern
set_gate_level primitive
report_gates i_1006/o

The gate report for the design level may look like the following:

```
/P2.13P  ND2
    A           I  /LD.1
    B           I  /M1.1
    Z           O  /P2.2P/S
```

The gate report for the primitive level may look like the following:

```
/P2.13P (23)  NAND
    A           I  9-/LD.1
    B           I  6-/M1.1
    Z           O  33-/P2.2P/S
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The gate report for the primitive level of a RAM gate may look like the following:

```
//  /P1.RAM/U1 (67)  RAM
//      "I0"   I  27-
//      "I1"   I  20-
//      RCK0   I  36-
//      "I3"   I  26-
//      "I4"   I  42-
//      "I5"   I  43-
//      "I6"   I  44-
//      "I7"   I  45-
//      "I8"   I  46-
//      WCK0   I  28-
//      "I10"  I  19-
//      A14    I  29-
//      A13    I  30-
//      A12    I  31-
//      A11    I  32-
//      A10    I  34-
//      D14    I  66-/P1.RAM/D1[0]
//      D13    I  65-/P1.RAM/D1[1]
//      D12    I  64-/P1.RAM/D1[2]
//      D11    I  63-/P1.RAM/D1[3]
//      D10    I  62-/P1.RAM/D1[4]
//      "OUT"  O  68-  69-  70-
//                71-  72-
//      address size =        5
//      data size =           5
//      minimum address =     0
//      maximum address =     31
//      number write ports =  1
//      number read ports =   1
//      edge_triggered =      no
//      initialization file = ram.init_file
//      write port: write=28/- (H)  first_adr=29  first_di=66
                                  stability=SS
//      read port:  read=36/- (0)  first_adr=42  first_do=68
                                  stability=SS
//      Test behavior: stability=SSSS  tie-X=0  read_only=1
                                  multiple_load=1/0  seq_transparent=0/0
//      Contention behavior: write_write=allow/X  read_read=normal
                                  read_write=read_new/write_normal
```

Example 2

The following example defines a function named “my_function” and displays all its simulation results from the last simulation pass:

set_system_mode analysis
add_atpg_functions my_function or /g1 ~/g2
set_gate_report parallel_pattern all
report_gates my_function

```
//  my_function (44)  USER_OR
//      "I0"    (0/0/0/0/0/0/0/0.../0) 12-/g1
//      ~"I1"   (1/1/1/1/1/1/1/1.../1) 13-/g2
//      "OUT"   (0/0/0/0/0/0/0/0.../0)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

The following example shows the use of wildcards:

report_gates /xscan_0_0_cch_scan_32x/ix15*

```
// /xscan_0_0_cch_scan_32x/ix151 NOR2XL
// A  I  /myop1<0> [1]
// B  I  /myop2[1]
// Y  O  /xscan_0_0_cch_scan_32x/sum_add_0_ix27/A0
// /xscan_0_0_cch_scan_32x/ix153 NAND2X1
// A  I  /myop1<0> [0]
// B  I  /myop2[0]
// Y  O  /xscan_0_0_cch_scan_32x/sum_add_0_ix1/A \
         /xscan_0_0_cch_scan_32x/ sum_add_0_ix23/B \
         /xscan_0_0_cch_scan_32x/sum_add_0_ix27/A1
// /xscan_0_0_cch_scan_32x/ix155 NAND2X1
// A  I  /myop1<0> [1]
// B  I  /myop2[1]
// Y  O  /xscan_0_0_cch_scan_32x/ix166/B0 \
         /xscan_0_0_cch_scan_32x/ sum_add_0_ix27/B0
// /xscan_0_0_cch_scan_32x/ix157 NAND2X1
// A  I  /myop1<0> [3]
// B  I  /myop2[3]
// Y  O  /xscan_0_0_cch_scan_32x/ix174/B0 \
         /xscan_0_0_cch_scan_32x/ sum_add_0_ix55/B0
```

report_gates /xscan_0_0_cch_scan_32x/ix159

```
// /xscan_0_0_cch_scan_32x/ix159 NAND2X1
// A  I  /myop1<0> [5]
// B  I  /myop2[5]
// Y  O  /xscan_0_0_cch_scan_32x/ix188/B0 \
         /xscan_0_0_cch_scan_32x/ sum_add_0_ix83/B0
```

set_gate_level primitive

report_gates /xscan_0_0_cch_scan_32x/ix157

```
// /xscan_0_0_cch_scan_32x/ix157 (43) NAND
// A  I  7-/myop1<0> [3]
// B  I  17-/myop2[3]
// Y  O  60-/xscan_0_0_cch_scan_32x/ix174/B0 \
         75-/xscan_0_0_cch_scan_32x/ sum_add_0_ix55/B0
// /xscan_0_0_cch_scan_32x/ix157 (43) NAND
// A  I  7-/myop1<0> [3]
// B  I  17-/myop2[3]
// Y  O  60-/xscan_0_0_cch_scan_32x/ix174/B0 \
         75-/xscan_0_0_cch_scan_32x/ sum_add_0_ix55/B0
// /xscan_0_0_cch_scan_32x/ix157 (43) NAND
// A  I  7-/myop1<0> [3]
// B  I  17-/myop2[3]
// Y  O  60-/xscan_0_0_cch_scan_32x/ix174/B0 \
         75-/xscan_0_0_cch_scan_32x/ sum_add_0_ix55/B0
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

The following example shows how the output report changes if the input pathname is a hierarchical pin or net. In this case, an additional line is output at the top of the report, indicating the mapping that was found.

set_gate_level design

report_gates /sub3/in2

```
// Hierarchical pin /sub3/in2 maps to /sub1/gate4/Y
// /sub1/gate4 nand02
//    A1 I /in1
//    A0 I /sub1/gate2/Y
//    Y  O /sub3/gate1/A1 /sub3/micro1/gate1/A1 /sub2/gate3/A1
//       /sub3/gate3/A1 /sub3/micro1/gate2/A1 /sub2/gate1/A1
```

report_gates /w2

```
// Hierarchical net /w2 maps to /sub1/gate4/Y
// /sub1/gate4 nand02
//    A1 I /in1
//    A0 I /sub1/gate2/Y
//    Y  O /sub3/gate1/A1 /sub3/micro1/gate1/A1 /sub2/gate3/A1
//         /sub3/gate3/A1 /sub3/micro1/gate2/A1 /sub2/gate1/A1
```

Example 5

The following example traces backward to the source of a DRC error:

```
//  Error: clock input of /ff2 (35) set to X. (T5-1)
//  Error: Scan chain c0 blocked at gate /ff2 (35) after tracing 1 cells.
(T3-1)
//  Error: Rules checking unsuccessful, cannot exit SETUP mode.
```

set_gate_level prim

set_gate_report drc shift

report_gates 35

```
//  /ff2 (35)  DFF
//      "S"    I  (000)  10-
//      "R"    I  (000)  9-
//      CLK    I  (0X0)  27-/gate8/Y
//      "D0"   I  (XXX)  32-
//      "OUT"  O  (XXX)  14-  15-
```

report_gates 27 -value_trace 1

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /gate8 (27)  AND
//      A0     I  (010)  6-/clk1
//      A1     I  (XXX)  21-/ff4/Q
//      Y      O  (0X0)  35-/ff2/CLK
//
//  /ff4 (21)  INV
//      "I0"   I  (XXX)  11-
//      Q      O  (XXX)  27-/gate8/A1
//
//  /ff4 (11)  INV
//      "I0"   I  (XXX)  33-
//      "OUT"  O  (XXX)  21-  22-
//
//  /ff4 (33)  DFF
//      "S"    I  (000)  10-
//      "R"    I  (000)  9-
//      CLK    I  (010)  7-/clk2
//      D      I  (XXX)  3-/in3
//      "OUT"  O  (XXX)  11-
//
// Total 4 gates traced which controls gate /gate8 (27) to X (bit 1).
```

Example 6

The following example traces backward to the first ambiguous gate where multiple source control the values:

set_gate_level prim

set_gate_report drc shift

report_gates 29

```
//  /sgate1 (29)  AND
//      A0     I  (XXX)  25-/sgate2/Y
//      A1     I  (111)  10-
//      Y      O  (XXX)  31-/ff1/SI
```

report_gates 29 -value_trace 1

```
//  /sgate1 (29)  AND
//      A0     I  (XXX)  25-/sgate2/Y
//      A1     I  (111)  10-
//      Y      O  (XXX)  31-/ff1/SI
//
//  /sgate2 (25)  OR
//      A0     I  (XXX)  21-/ff0/Q
//      A1     I  (XXX)  16-/ff2/QB
//      Y      O  (XXX)  29-/sgate1/A0
// Total 2 gates traced which controls gate /sgate1 (29) to X (bit 1).
```

Example 7

The following example traces backward to all ambiguous gates:

report_gates 29 -value_trace 1 -go_all

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /sgate1 (29)  AND
//      A0     I  (XXX)  25-/sgate2/Y
//      A1     I  (111)  10-
//      Y      O  (XXX)  31-/ff1/SI
//
//  /sgate2 (25)  OR
//      A0     I  (XXX)  21-/ff0/Q
//      A1     I  (XXX)  16-/ff2/QB
//      Y      O  (XXX)  29-/sgate1/A0
//
//  /ff0 (21)  INV
//      "I0"   I  (XXX)  12-
//      Q      O  (XXX)  25-/sgate2/A0
//
//  /ff0 (12)  INV
//      "I0"   I  (XXX)  35-
//      "OUT"  O  (XXX)  21-  22-
//
//  /ff0 (35)  DFF
//      "S"    I  (000)  11-
//      "R"    I  (000)  9-
//      CLK    I  (010)  6-/clk1
//      D      I  (XXX)  3-/in3
//      "OUT"  O  (XXX)  12-
//
//  /ff2 (16)  INV
//      "I0"   I  (XXX)  37-
//      QB     O  (XXX)  25-/sgate2/A1
//
//  /ff2 (37)  DFF
//      "S"    I  (000)  11-
//      "R"    I  (000)  9-
//      CLK    I  (010)  6-/clk1
//      "D0"   I  (XXX)  34-
//      "OUT"  O  (XXX)  15-  16-
// Total 7 gates traced which controls gate /sgate1 (29) to X (bit 1).
```

Example 8

The following example traces backward from the output of gate 122 to the origin of the transition:

```
//  Fault type is transition
```

set_gate_level prim
set_gate_report pattern_index 3
report_gates 122 -value_trace -transition

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /top_i/sf_0002_2 (122)  MUX
//      SE     I  (111-111)  6-/sen
//      D      I  (111-000)  3-/pi[0]
//      SI     I  (011-111)  36-/top_i/sf_0001_2/Q
//      "OUT"  O  (011-111)  419-
//  /top_i/sf_0001_2 (36)  BUF
//      "I0"   I  (011-111)  418-
//      Q      O  (011-111)  115-/top_i/an_2/A2  122-/top_i/sf_0002_2/SI
//  /top_i/sf_0001_2 (418)  DFF
//      "S"    I  (000-000)  12-
//      "R"    I  (000-000)  136-
//      CLK    I  (010-000)  5-/clk
//      "D0"   I  (111-111)  404-
//      "OUT"  O  (011-111-[1])  36-  37-
//      MASTER  cell_id=5  chain=chain1  group=grp1  invert_data=FFFF
//
//  Total 3 gates traced which control gate '/top_i/sf_0002_2/' (122) to
L_H transition (bit 0-1).
```

Example 9

The following example displays the status of the faults on the gate and its inputs:

set_gate_report fault_status
report_gates /ram1/address<4>

```
//  /ram1  picdram
//      clk    I  (AU.SEQ:AU.SEQ)  /gater2/an_4/Y
//      address<6>  I  (AU.SEQ:AU.SEQ)  /address1[6]
//      address<5>  I  (AU.SEQ:AU.SEQ)  /address1[5]
//      address<4>  I  (AU.SEQ:AU.SEQ)  /address1[4]
//      address<3>  I  (AU.SEQ:AU.SEQ)  /address1[3]
//      address<2>  I  (AU.SEQ:AU.SEQ)  /address1[2]
```

report_gates /an_10/A1

```
//  /an_10  and03
//      A0     I  (AU.BB:AU.BB)  /sf_1_6/Q
//      A1     I  (AU.BB:AU.BB)  /sf_3_6/Q
//      A2     I  (AU.BB:AU.BB)  /sf_5_6/Q
//      Y      O  (AU.BB:AU.BB)  /po6
```

report_gates /gater11/an_1/Y

```
//  /gater11/an_1  inv02
//      A      I  (AU.TC:AU.TC)  /an_13/Y
//      Y      O  (AU.TC:AU.TC)  /gater11/lat1/CLK
```

In the example, the tool reports the following untestable (AU) fault sub-classes:

- BLACK_BOXES (BB) — Untestable faults due to black boxes.
- SEQUENTIAL_DEPTH (SEQ) — Untestable faults with the current sequential_depth and may be testable with a higher depth.
- TIED_CELLS (TC) — Untestable faults due to tied or blocked tied non-scan cells.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For a complete list of these AU fault sub-classes, see Table 8-19.

Example 10

The following example shows reporting for K19 simulation:

set_gate_report drc k19

report_gates 160

```
//  /edt_i/edt_decompressor_i/ix159 (160) NXOR
//      A0     I 147-/edt_i/edt_decompressor_i/ix241/Y
//      A1     I 93-/edt_i/edt_decompressor_i/reg_lfsm_vec_lockup_4_/Q
//      Y      O 173-/edt_i/ix90/A
//
//  Proc: t ld_u sh 1 sh 2 sh 3 sh 4 sh 5 sh 6 sh 7 sh 8 sh 9 sh10 sh11 cap
//  ----- - ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---
//  Time:    234  123  123  123  123  123  123  123  123  123  123  123 o o
//        * 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 fXf
//  ----- - ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---
//  Sim:  X XX00 0000 0001 1110 0001 1110 0001 1110 0001 1110 0000 0001 111
//  Emu:  - ---- -0-- -0-- -1-- -0-- -1-- -0-- -1-- -0-- -1-- -0-- -0-- ---
//  No mismatch.
//  Monitor: EDT decompressor output driving 'chain1' (chain index 1).
//
//  Inputs:
//  A0    X XX11 1110 0000 0001 1110 0001 1111 1110 0000 0000 0001 1111 111
//  A1    X XX00 0001 1110 0000 0000 0000 0001 1111 1110 0001 1110 0001 111
//
```

Example 11

The following example shows how to report only the total number of master gates:

ANALYSIS> report_gates -type master -summary
// Total number of master gates = 376034

Example 12

This example demonstrates how to report the simulated values from all natural, tied gates, and learned constant value non-scan cells.

Use the “set_gate_report -constrain_value on” command. Then report_gates is used.

The report shows that the INV gate in /sdff2/lat is unreachable from any observation point, as indicated by UU value.

set_gate_report -constrain_value on
report_gates sdff2/lat

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /sdff2/lat (18)  BUF
//      "I0"   I  (X/01/-)  34-
//      Q      O  (X/01/-)  36-/so
//  /sdff2/lat (19)  INV
//      "I0"   I  (X/01/-)  34-
//      QN     O  (X/01/-/UU)
//  /sdff2/lat (34)  TLA
//      "S"    I  (0/1/-)  9-
//      "R"    I  (0/1/-)  8-
//      CLK    I  (-/-/B)  26-/sdff2/u0/OUT
//      D      I  (-/-/-)  29-/sdff2/u1/OUT
//      "OUT"  O  (X/01/-)  18-  19-
//  Reported all gates in instance 'sdff2/lat'
```

Example 13

This procedure file defines four events in the shift procedure (force_pi is set to occur at 5 ns rather than at the beginning of the cycle).

```
set time scale 1.000000 ns ;
set strobe_window time 1 ;
 timeplate gen_tp1 =
    force_pi 5 ;
    measure_po 10 ;
    pulse CLOCK 20 10;
    period 40 ;
 end;
 procedure shift =
    scan_group grp1 ;
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force_sci ;
        measure_sco ;
        pulse CLOCK ;
    end;
end;
procedure load_unload =
    scan_group grp1 ;
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force scan_en 1 ;
        force CLOCK 0 ;
    end ;
    apply shift 3;
 end;
 procedure test_setup =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force IN1 0 ;
    end;
 end;
```

The report_gates command reports four values, one for each event in the cycle.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_gate_level design
set_gate_report drc shift
report_gates .reg1

```
//  /reg1  dfsc
//      CK     I  (0010)  /CLOCK
//      D0     I  (XXXX)  /gate1/OUT
//      D1     I  (SSSS)  /reg2/QBAR
//      SC     I  (1111)  /scan_en
//      Q      O  (XXXX)  /gate2/B
//      QBAR   O  (SSSS)  /scan_out1
```

Example 14

This example demonstrates how to report gates capturing Xs. The report lists all gates unless you specify an optional number to limit the output to that number with the highest incidence of X captures. Use the "set_simulation_options" command to enable X recording during good machine simulation. Simulate your patterns, then run the "report_gates" command.

set_simulation_options -report_x_capturing_cells on

simulate_patterns

report_gates -x_capture_cells 4

```
//      ----------------------------  ------------
//      Gate (Id)                     Number of Xs
//      ----------------------------  ------------
//      '/chain1_inst/s1_sff1/' (90)            26
//      '/chain0_inst/s1_sff1/' (86)            26
//      '/chain0_inst/s1_sff3/' (88)            19
//      '/chain2_inst/s1_sff2/' (95)            17
//     Only the first 4 cells capturing X with the most masking have been
reported (total 7 cells).
```

report_gates -x_capture_cells

```
//      ----------------------------  ------------
//      Gate (Id)                     Number of Xs
//      ----------------------------  ------------
//      '/chain1_inst/s1_sff1/' (90)            26
//      '/chain0_inst/s1_sff1/' (86)            26
//      '/chain0_inst/s1_sff3/' (88)            19
//      '/chain2_inst/s1_sff2/' (95)            17
//      '/chain1_inst/s1_sff3/' (92)            16
//      '/chain2_inst/s1_sff3/' (96)            15
//      '/chain2_inst/s1_sff1/' (94)             9
//      A total of 7 cells capturing X have been reported.
```

Related Topics

set_gate_level

set_gate_report

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_gate_to_rtl_mapping

Context: all contexts

Mode: setup

Report rules for post-synthesis/post-layout to RTL instance name-mapping for DEF files.

Usage

report_gate_to_rtl_mapping

Description

Reports the rules added with the add_gate_to_rtl_mapping command for mapping post-synthesis/post-layout instances listed in the DEF file to RTL design instance names.

Arguments

None

Examples

The following example shows a report from mapping rules that were previously applied with the add_gate_to_rtl_mapping command:

```
report_gate_to_rtl_mapping
Index  Original instance name       Mapped instance name
-----  ---------------------------  ------------------------
1      memA                         /MEMORY.mem1
2      memB                         /MEMORY.mem2
3      memC                         /MEMORY.mem3
4      memD                         /MEMORY.mem4
5      memE                         /MEMORY.mem5
6      memF                         /MEMORY.mem6
```

The “Original instance name” is the DEF file instance that is mapped to the RTL “Mapped instance name” instance. The listed “Index” value is used with the delete_gate_to_rtl_mapping command to identify the name mapping rule to remove.

Related Topics

add_gate_to_rtl_mapping

delete_gate_to_rtl_mapping

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_graybox_statistics

Context: patterns -scan

Mode: analysis

Reports the statistics gathered by graybox analysis.

Usage

report_graybox_statistics {[-top *integer*] | [-percentage_exceeds *integer*] | [-exceeds_max_sequential_levels]}

Description

Reports the statistics gathered by graybox analysis. This command works only after running graybox analysis using the -collect_reporting_data switch of the analyze_graybox command.

Arguments

- -top *integer*

  An optional switch and integer pair that lists the graybox analysis backward tracing origins with the highest number of leaf instances. The *integer* argument specifies how many tracing origins from the top are listed in the report output. A tracing origin can be a PO, a wrapper cell input pin, or a preserve instance.

- -percentage_exceeds *integer*

  An optional switch that lists all tracing origins that exceed the specified percentage of leaf instances (with respect to all leaf instances in design) covered during backtracing.

- -exceeds_max_sequential_levels

  An optional switch that lists the tracing origins that exceed the maximum sequential levels limit specified by “analyze_graybox -max_sequential_levels”. The default value for maximum sequential levels is 2 unless specified by the -max_sequential_levels switch.

Examples

The following example shows the command output for the top four tracing origins with the highest number of leaf instances:

report_graybox_statistics -top 4

```
---------------------------------------------------------------------------
                                Number of      Percentage   Sequential
Start Point  Pin Type           Leaf Instances of Design    Levels Limit
                                                            Reached
-----------  ------------------ -------------  -----------  ---------
sff2/D,SI    WrapperCellDataPin 3              50.00        yes
sff1/D       WrapperCellDataPin 2              33.33        -
po           PrimaryOutput      1              16.67        -
so           PrimaryOutput      1              16.67        -
---------------------------------------------------------------------------
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The explanation of the columns are as follows:

- Start Point is the pathname of the tracing origin.
- Pin Type is one of the following: PrimaryOutput, WrapperCellDataPin, WrapperCellSetPin, WrapperCellResetPin, WrapperCellClockPin, PreserveInstanceInputPin.
- Number of Leaf Instances is the number of leaf instances marked for the tracing origin.
- Percentage of Design is the percentage of covered leaf instances with respect to the leaf instances in the entire design.
- Sequential Levels Limit Reached is “yes” when the specified or default maximum sequential levels limit is exceeded, and is “-” otherwise.

Related Topics

analyze_graybox

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_icl_extraction_options

Context: all contexts

Mode: all modes

Provides a human readable report of the ICL extraction options.

Usage

report_icl_extraction_options

Description

Provides a human readable report of the ICL extraction options.

Arguments

None

Examples

The following command displays the ICL extraction settings in a human readable form:

report_icl_extraction_options

```
//  Option                   | Type | Value
//  -------------------------+------+------
//  extract_learned_muxes    | BOOL | on
//  extract_library_muxes    | BOOL | on
//  extract_scan_interfaces  | BOOL | on
```

Related Topics

get_icl_extraction_options

set_icl_extraction_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_icl_modules

Context: dft, patterns

Mode: all modes

Reports loaded and extracted ICL modules in either ICL syntax or human readable form. The output can be redirected to a file.

Usage

report_icl_modules [-all | -modules *modules* [-hierarchical]]

Description

If you invoke this command without any arguments, the names of all loaded and extracted ICL modules are reported in human readable form.

Note

If you invoke this command during setup mode, before ICL extraction has taken place, only the loaded ICL modules are reported. If you invoke this command in the analysis mode, the tool may have performed ICL extraction, and the report_icl_modules command reports the loaded and extracted ICL modules.

Arguments

- -all

  An optional switch that reports all loaded and extracted ICL modules in ICL syntax.

- -modules *modules*

  An optional switch that reports the specified modules in ICL syntax. The switch can take an *object_spec* (TCL string list or object list).

- -hierarchical

  An optional switch that reports the listed modules, as well as a recursive listing of all modules that are instantiated by the listed modules and their child modules.

Examples

Example 1

In the following example, when you issue this command without any arguments, it reports the names of all the ICL modules in human readable form:

report_icl_modules

```
//          pll1
//          pll2
//          pll3
//          pll4
//          block
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the current design is set when you issue this command, the current design is marked as shown in the following example:

set_current_design top
report_icl_modules

```
//          pll1
//          pll2
//          pll3
//          pll4
//          block
//          top     (current iProcsForModule ICL module)
//          Middle
//          bottom
```

Example 2

The following example reports the ICL syntax for the module chip:

report_icl_modules -modules chip

```
Module chip {
   // ICL module read from source on or near line 1 of file './data/icl/
chip3.icl'
   TCKPort tck;
   ScanInPort tdi;
   ScanOutPort tdo {
      Source MyTap.tdo;
   }
   TMSPort tms;
   TRSTPort trst;
   Instance MyTap Of tap {
      InputPort tck = tck;
      InputPort tdi = tdi;
      InputPort tms = tms;
      InputPort trst = trst;
      InputPort fromTdr1 = MySib2.so;
   }
}
```

Example 3

The following example reports the ICL syntax for the module chip as well as a recursive listing of all modules instantiated by chip and its child modules.

report_icl_modules -modules chip -hierarchical

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module chip {
   // ICL module read from source on or near line 1 of file './data/icl/
chip.icl'
   TCKPort tck;
   ScanInPort tdi;
   ScanOutPort tdo {
      Source MyTap.tdo;
   }
   TMSPort tms;
   TRSTPort trst;
   Instance MyTap Of tap {
      InputPort tck = tck;
      InputPort tdi = tdi;
      InputPort tms = tms;
      InputPort trst = trst;
      InputPort fromTdr1 = MySib2.so;
   }
   Instance MySib1 Of sib {
      InputPort si = tdi;
      InputPort se = MyTap.se;
      InputPort ce = MyTap.ce;
      InputPort ue = MyTap.ue;
      InputPort en = MyTap.tdrEn1;
      InputPort tck = tck;
      InputPort fso = MyTdr1.so;
   }
}

// instanced as chip.MyTap
Module tap {
   // ICL module read from source on or near line 1 of file './data/icl/
tap.icl'
   TCKPort tck;
   ScanInPort tdi;
   ScanOutPort tdo {
      Source IRMux;
   }
   TMSPort tms;
   TRSTPort trst;
   ToSelectPort tdrEn1 {
      Source sel1;
   }
   ScanInPort fromTdr1;
   ToSelectPort tapEn1 {
      Source IR[2];
   }
   ToCaptureEnPort ce;
   ToShiftEnPort se;
   ToUpdateEnPort ue;
   ScanInterface TAP {
      Port tdi;
      Port tdo;
      Port tms;
   }
   ScanInterface Internal {
      Port fromTdr1;
      Port tdo;
      Port tdrEn1;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
         Port ce;
         Port se;
         Port ue;
      }
      ScanRegister IR[2:0] {
         ScanInSource tdi;
         CaptureSource 3'b001;
         ResetValue 3'b000;
      }
      ScanRegister bypass {
         ScanInSource tdi;
         CaptureSource 1'b0;
      }
      ScanMux IRMux SelectedBy fsm.irSel {
         1'b0 : DRMux;
         1'b1 : IR[0];
      }
      ScanMux DRMux SelectedBy IR[2:0] {
         3'bx00 : bypass;
         3'bx01 : fromTdr1;
      }
      LogicSignal sel1 {
         IR[2:0] == 3'bx01;
      }
      Instance fsm Of tap1_fsm {
         InputPort tck = tck;
         InputPort tms = tms;
         InputPort trst = trst;
      }
   }

   // instanced as tap.fsm
   Module tap1_fsm {
      // ICL module read from source on or near line 59 of file './data/icl/
   tap.icl'
      TCKPort tck;
      TMSPort tms;
      TRSTPort trst;
      ToIRSelectPort irSel;
      ToResetPort tlr;
   }
```

Related Topics

get_icl_instances

get_icl_modules

read_icl

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_icl_network

Context: patterns -silicon_insight -ijtag

Mode: setup

Reports various information about the IJTAG network.

Usage

report_icl_network {-current_scan_path | -node_name *node_name* | [-node_type all | sib | scanmux | scanreg | port] | -top_client_interfaces | -current_top_chains | -current_scan_chain}

Description

Returns a report about the specified IJTAG network component.

Use this command when manually debugging IJTAG networks as described in "Diagnosing and Debugging IJTAG Networks" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

Arguments

- -current_scan_path

  An optional switch that reports the currently configured scan path.

- -node_name *node_name*

  An optional switch and value pair that reports the specified ICL network node.

- -node_type all | sib | scanmux | scanreg | port

  An optional switch and literal pair that reports information about the specified node type.

- -top_client_interfaces

  An optional switch that reports the top IJTAG client interfaces.

- -current_top_chains

  An optional switch that reports the scan chains in the selected top IJTAG client interface. The selected top interface was either previously specified with the set_icl_network -top_client_interface command or is the default top interface if there is only one interface.

- -current_scan_chain

  An optional switch that reports information about the currently selected scan chain. The current scan chain is a scan chain previously specified with the set_icl_network -current_scan_chain_number command.

Examples

report_icl_network -current_scan_path

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Current Scan Path(length=4 bits, 8 nodes):
   Port: tdo
   ScanMux: tap_I1.IRMux@0
   ScanMux: tap_I1.DRMux@1
   Sib: sib_I4@0
   Sib: sib_I3@0
   Sib: sib_I2@0
   Sib: sib_I1@0
   Port: tdi
```

report_icl_network -node_name sib_I4

```
Sib: sib_I4(shortest path (SO -> <this element> -> SI) = 4)
  Input-0 : Sib('sib_I3')
  Input-1 : ScanReg('block4_I1.Selector[1:0]')
  Output-0 : ScanMux('tap_I1.DRMux')
```

Related Topics

analyze_icl_suspect

assert_icl_sib

create_icl_flush_pattern

create_icl_setup_pattern

deassert_icl_sib

get_icl_network

get_icl_scan_mux

isolate_icl_node

set_icl_network

set_icl_scan_mux

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_iclock

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: In the patterns -ijtag context, a pattern set has been opened by the open_pattern_set command.

Reports the ICL ClockPort specified by the iClock command as well as their extracted sources and cumulative freqMultiplier and freqDivider values.

Usage

report_iclock [-overridden]

Description

Reports the ICL ClockPort specified by the iClock command as well as their extracted sources and cumulative freqMultiplier and freqDivider values.

Arguments

- -overridden

  Specifies to return the ICL port and pin names on which an iClockOverride command was issued as opposed to the iClock command when the option is not specified.

Return Values

None

Examples

The following example shows the result of the report_iclock command specified inside a pattern set in which two iClock commands have been issued. The first iClock command is on the reference pin of a PLL. The root-level clock port called CLK is its source and the connection is direct. The second one is on the CLK input of a memory BIST instance.

The report reflects the fact that the source tracing went through the PLL and picked up a FreqMultiplier value of 10 and a FreqDivider value of 2 either from the ICL definition or from an iClockOverride command that could have been specified on the output clock port of the PLL as shown in the example provided for the iClock command.

report_iclock

```
// iClock    Source FreqMultiplier  FreqDivider     Period
// =======   =====  ==============  ===========     ======
// PLL.ref   CLK                 1            1      100ns
// mbist.clk CLK                10            2       20ns
```

Related Topics

add_clocks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_clocks

get_iclock_list

get_iclock_option

iClock

iClockOverride

iRunLoop

report_clocks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_id_stamp

Context: dft -edt, patterns -scan

Mode: analysis

Displays the unique identifier that the tool assigns each internal pattern set.

Usage

report_id_stamp [{> | >>} *file_pathname*]

Description

Displays the unique identifier that the tool assigns each internal pattern set.

The report_id_stamp command displays the current internal pattern set's unique identification stamp which consists of five fields, each separated by a colon (:). Due to the length of three of the fields, the tool encodes those fields and displays the encoded information. The tool encodes the three fields using 4 bytes of hexadecimal numbers. This encoding's primary use is to ensure that each pattern set has a unique identification stamp. The following list shows the information each field provides:

1. Tool version number
2. Encoded environment settings
3. Encoded DRC rules data
4. Number of patterns in the internal pattern set
5. Encoded pattern data

The tool generates the identification stamp each time you issue either the report_id_stamp command or the write_patterns -Environment command. You can use the identification stamp to tag identical patterns saved in different formats.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following example displays the unique identification stamp for the current pattern set:

report_id_stamp

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
2012.3:5c95:3e10:16:1bf2
```

Related Topics

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_iddq_exceptions

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays current exceptions to Iddq restrictions.

Usage

report_iddq_exceptions [{> | >>} *file_pathname*]

Description

Displays current exceptions to Iddq restrictions.

The report_iddq_exceptions command displays Iddq exceptions added with the add_iddq_exceptions command. For more information about IDDQ, refer to "IDDQ Test Set Creation" in the *Scan and ATPG User's Manual*.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example displays Iddq exceptions on sites for which the tool would report Iddq check violations if the exceptions did not exist.

```
report_iddq_exceptions
```

Related Topics

add_iddq_exceptions

analyze_restrictions

delete_iddq_exceptions

report_gates

report_primary_outputs

set_iddq_checks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_ijtag_instances

Context: dft patterns

Mode: setup analysis insertion

Reports options of instances for which there is a matched ICL module.

Usage

report_ijtag_instances [*name_spec*] [-below_instances *below_instances*]
[-filter *filter*] [-limit *limit*]
[-silent]

Description

Reports options of instances for which there is a matched ICL module.

You can restrict the report to instances below a given collection of instances and filter the collection based on an attribute filtering equation involving other attributes registered against or inherited by the instance object type.

The -silent option suppresses the warning that is normally generated if there are no Ijtag instances to report.

Arguments

- *name_spec*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of instances. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no name_patterns are specified, the tool searches all instances.

  If you specify the -below_instances options, the name patterns are hierarchical name patterns relative to the specified instance objects; otherwise the name patterns are relative to the current design.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for instances below the specified instance_objects. The specified name_patterns are searched relative to each instance found in the instance_objects list.

- -filter a*ttribute_equation*

  An optional switch and string that specify to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the instance object type. See section "Attribute Filtering Equation Syntax" on page 3832 for more details on filtering attribute equation format.

- -silent

  An optional switch that suppresses the warning message normally generated when the command does not find any instances to report. The warning message is intended to indicate

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

a probable mistake when specifying patterns. You can use the -silent switch to suppress the warning. This -silent option has no effect on any other error reporting.

Examples

The following example uses the report_ijtag_instances command to report the options of the first three instances of the Ijtag module. The command create_dft_specification provides access to those features if they are not already connected to the network.

report_ijtag_instances -l 3

```
//
// ijtag instance: pb_di_do
// ----------------------------------------------------------------
// use_in_dft_specification : auto
// scan_interface_count     : 1
// data_in_port_count       : 4
// data_out_port_count      : 0
//
// ijtag instance: pb_di_do/dido_bus
// ----------------------------------------------------------------
// use_in_dft_specification : auto
// scan_interface_count     : 1
// data_in_port_count       : 16
// data_out_port_count      : 16
//
// ijtag instance: pb_si
// ----------------------------------------------------------------
// use_in_dft_specification : auto
// scan_interface_count     : 1
// data_in_port_count       : 0
// data_out_port_count      : 0
//
// Reached limit of 3, skipping remaining 9 instances.
```

Related Topics

get_ijtag_instances

get_ijtag_instance_option

set_ijtag_instance_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_ijtag_logical_connections

Context: patterns -ijtag

Mode: setup, analysis

Reports the logical paths that exist within the current design between the specified source and destination pins/ports, as well as all connections from or to the specified pins/ports.

Usage

report_ijtag_logical_connections [-from *source_pin_or_port_name*]
[-to *destination_pin_or_port_name*]

Description

Reports the logical paths that exist within the current design between the specified source and destination pins/ports, as well as all connections from or to the specified pins/ports. If no pin or port connections are specified, all logical connections are listed.

Arguments

- -from *source_pin_or_port_name*

  An optional switch and value pair that specifies the source pin/port for the logical connections, which should be a top level module port or a hierarchical instance pin in the current top design module.

- -to *destination_pin_or_port_name*

  An optional switch and value pair that specifies the destination pin/port for the logical connections, which should be a top level module port or a hierarchical instance pin in the current top design module.

Examples

Example 1

The following example lists the logical connections between the pins /i1/in and /i32/u11/x10, as well as all logical connections from /i1/in and all connections to /i32/u11/x10.

report_ijtag_logical_connections -from /i1/in -to /i32/u11/x10

```
//  IJTAG Logical Connections:
//  From Source  To Destination   Inversion
//  ===========  ==============   =========
//  /i1/in       /i32/u11/x10     no
```

Example 2

The following example lists all the logical connections in the current design.

add_ijtag_logical_connection -from tap1_I1/ue -to block3_I1/ue -inverted

add_ijtag_logical_connection -from tap1_I1/ue -to logical_out -inverted

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_ijtag_logical_connection -from tdi -to tdo -inverted

add_ijtag_logical_connection -from tdi -to tms

add_ijtag_logical_connection -from tdo -to tms -inverted

report_ijtag_logical_connections

```
//  IJTAG Logical Connections:
//  From Source  To Destination  Inversion
//  ===========  ==============  =========
//  /tap1_I1/ue  /block3_I1/ue   yes
//  /tap1_I1/ue  logical_out     yes
//  tdi          tdo             yes
//  tdi          tms             no
//  tdo          tms             yes
```

Related Topics

add_ijtag_logical_connection

delete_ijtag_logical_connection

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_ijtag_retargeting_options

Context: all contexts

Mode: all modes

Reports the type and value of each option of the previous set_ijtag_retargeting_options command or their default values if the set_ijtag_retargeting_options command was not issued.

Usage

report_ijtag_retargeting_options

Description

Reports the type and value of each option of the previous set_ijtag_retargeting_options command or their default values if the set_ijtag_retargeting_options command was not issued.

Note

The value of tck_ratio might depend on the maximum tck clock frequency that can be applied to the circuit. See "IJTAG Network Performance Optimization" in the *Tessent IJTAG User's Manual* showing how to maximize the frequency of the IJTAG network test clock.

Arguments

None

Examples

The following example sets the context "patterns -ijtag" and reports the current retargeting options:

```
SETUP> set_context patterns -ijtag -no_rtl
SETUP> report_ijtag_retargeting_options
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Option                                        | Type |            Value
//  ----------------------------------------------+------+-----------------
//  test_setup_network_end_state                  | ENUM |            reset
//  test_setup_ireset                             | ENUM | when_icl_present
//  iapply_target_annotations                     | ENUM |            dense
//  compare_constant_capture_values               | ENUM |               on
//  compare_constant_data_out_ports               | ENUM |              off
//  ignore_non_reset_end_state_implicit_compares  | ENUM |             auto
//  constrain_update_en_only                      | ENUM |             auto
//  interface_type                                | ENUM |             auto
//  max_operations_per_iapply                     | INT  |               64
//  merge_irunloop_only                           | BOOL |              off
//  disable_pio_operations                        | BOOL |              off
//  ignore_iwrite_when_value_already_present      | BOOL |              off
//  allow_skip_rti                                | BOOL |               on
//  generated_timeplate_name                      | STR  |          gen_tp%
//  tck_ratio                                     | INT  |             auto
//  inject_cycles                                 | INT  |                0
//  annotation_parameter_values                   |      |
//   - user                                       | BOOL |               on
//   - iapply_targets                             | ENUM |            dense
//   - scan_path_registers                        | BOOL |               on
//   - scan_load_unload_value                     | BOOL |               on
//   - tap_states                                 | BOOL |               on
//   - network_end_state                          | BOOL |               on
//   - scanin                                     | BOOL |              off
//   - pragma_annotation                          | BOOL |               on
......
......
```

Related Topics

get_ijtag_retargeting_options

set_ijtag_retargeting_options

reset_ijtag_retargeting_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_input_constraints

Context: all contexts

Mode: setup, analysis

Displays the present constraint status of primary input pins.

Usage

Context: dft -scan and dft -test_points

report_input_constraints
[-ALl | *primary_input_pin_name*… | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs | -EQuivalent] [*constraint_switch*]

Context: patterns -scan

report_input_constraints
[-ALl | *primary_input_pin_name*… | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs | -EQuivalent] [*constraint_switch*] [-HOld] [-NO_Z] [-SLow_pad [-CELL *model_name*]]

Context: patterns -ijtag

report_input_constraints
[-ALl | *primary_input_pin_name*… | -BIDI_Only | -BIDI_Exclude| -EQuivalent] [*constraint_switch*]

Description

Displays the present constraint status of primary input pins.

The report_input_constraints command displays the constraint status of primary input pins to which you have applied constraints using the add_input_constraints command.

Arguments

- -ALl

  An optional switch that specifies to display the current constraint status of every primary input pin to which you previously applied an input constraint using the add_input_constraints command. Pins from which you removed the constraint with the delete_input_constraints command prior to reporting the status are included in the report.

- *primary_input_pin_name*

  An optional, repeatable string that specifies the name of a primary input pin for which to display the constraint status.

- -BIDI_Only

  An optional switch that specifies to display the constraint status of bidirectional pins only.

- -BIDI_Exclude

  An optional switch that specifies to display the constraint status only of primary input pins that are not bidirectional.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -SCAN_INputs

  An optional switch that specifies to display the constraint status of scan input pins only.

- -EQuivalent

  An optional switch that specifies to display the constraint status of primary input pins that have a pin equivalence definition applied to them. These are pins for which you used the -Equivalent switch with the add_input_constraints command. The -C_all constraint switch is the only constraint switch you can include in the same command line with -Equivalent.

- *constraint_switch*

  An optional switch that specifies to report only on primary input pins that have a particular type of constraint applied. The choices, from which you can specify only one, are as follows:

  -C0 — Constant 0

  -C1 — Constant 1

  -CX — Constant X (unknown)

  -CZ — Constant Z (high-impedance)

  -CT0 — Constant TIE0

  -CT1 — Constant TIE1

  -CTZ — Constant TIEZ

  Note

  For additional information about the constraints associated with the preceding switches, refer to the same switches under the add_input_constraints command.

  -C_All — All C-type constraints

- -HOld

  An optional switch that specifies to report only input constraints that currently exhibit hold behavior. For information about what hold behavior is, refer to the -Hold argument under the add_input_constraints command.

- -NO_Z

  An optional switch that specifies to report only input constraints that currently exhibit no-Z behavior. For information about what no-Z behavior is, refer to the -No_z argument under the add_input_constraints command.

- -SLow_pad

  An optional switch that specifies to report only input constraints that currently exhibit slow-pad behavior. For information about what slow-pad behavior is, refer to the -Slow_pad argument under the add_input_constraints command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -CELL *model_name*

  An optional switch and string pair that, together with the -Slow_pad switch, specifies to prefix the instance name of each instance of the Tessent Cell library model, *model_name*, to the *primary_input_pin_name* and look up each resulting name as the pin on which to report.

Examples

The following example reports input constraints for a primary input.

report_input_constraints indata2

```
//  Primary Input  C-Type  Hold      No-Z      Slow-Pad
//  -------------  ------  ----  ------------  --------
//  indata2          C0     No   Undetermined     No
```

Related Topics

add_input_constraints

get_input_constraints

delete_input_constraints

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_insert_test_logic_options

Context: dft -scan, dft -test_points

Mode: setup, analysis, insertion

Reports the prefix/infix values that are used when logic is created by the insert_test_logic command.

Usage

report_insert_test_logic_options

Description

The report_insert_test_logic_options command reports the prefix/infix values that are used when logic is created by the insert_test_logic command.

Arguments

None

Examples

This example reports the current prefix/infix values.

set_context dft -scan
report_insert_test_logic_options

```
SETUP> set_context dft -scan
SETUP> report_insert_test_logic_options
//  'insert_test_logic' Command Option              Current Value
//  ----------------------------------------    --------------------
// -inserted_object_naming                      standard
// -inserted_object_prefix                      ts_
// -logic_type clock_control_fix                clkfix_
// -logic_type clock_gater_fix                  cgfix_
// -logic_type dedicated_input_holding_wrapper  dihw_
// -logic_type dedicated_input_wrapper          diw_
// -logic_type dedicated_output_holding_wrapper dohw_
// -logic_type dedicated_output_wrapper         dow_
// -logic_type generic                          logic_
// -logic_type lockup                           lockup_
// -logic_type pipeline                         pipeline_
// -logic_type ram_fix                          ramfix_
// -logic_type se_pipeline                      se_pipeline_
// -logic_type shared_input_holding_wrapper     sihw_
// -logic_type shared_output_holding_wrapper    sohw_
// -logic_type testpoint_control                cp_
// -logic_type testpoint_observation            op_
// -logic_type testpoint_observation_scan       osp_
// -logic_type tri_state_fix                    trifix_
// -logic_type x_bounding                       xb_
// -persistent_cell_prefix                      tessent_persistent_cell_
// -persistent_clock_cell_prefix                tessent_persistent_cell_
// -reuse_modules_when_possible                 off
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

get_insert_test_logic_option

set_insert_test_logic_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_instance_polygons

Context: patterns -scan_diagnosis

Mode: analysis

Returns the x, y coordinate location of the specified instance in the LDB.

Usage

report_instance_polygons ***instance_path*** [-ldb_units]

Arguments

- instance_path

  Required string that specifies the path to a polygon instance within the LDB.

- -ldb_units

  An optional switch that returns the x, y coordinates in LDB units rather than the default micron units.

Description

This command returns the x and y coordinate location corresponding to a specified instance path in the LDB. The tool returns chip-level coordinates.

By default, the command returns coordinates expressed in micron units. You can override the default by specifying the -ldb_units switch.

Related Topics

get_instance_polygons

get_xy_instance

report_net_polygons

report_xy_instance

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_iprocs

Context: all contexts

Mode: all modes

Displays a table listing of all iProcs and their related namespace iProcNameSpace for each ICL module.

Usage

report_iprocs

Description

Displays a table listing of all iProcs and their related namespace iProcNameSpace for each ICL module. The iProcNameSpace column is not displayed if all iProcs are defined in the global namespace.

Arguments

None

Examples

The following is an example of the report generated by this command:

report_iprocs

```
//  Module  iProcNameSpace  iProcs
//  ==============================================================
//  chip                    chipProc1 chipProc2 chipProc3 chipProc4
//              Bar         chipProc1 chipProc3
//              Foo         chipProc2 chipProc4
//  block                   blockProc1 blockProc2 blockProc3
//                          blockProc4 blockProc5
//              Bar         blockProc1 blockProc3 blockProc5
//              Foo         blockProc2 blockProc4
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 7
Command Dictionary (Report_j-R)

The Command Dictionary describes the application commands for Tessent Shell. For quick reference, the commands appear alphabetically with each beginning on a separate page.

The command usage line syntax conventions are common to all products documented in this manual. See "Conventions for Command Line Syntax" on page 97.

Command Descriptions 1926
report_layout_core_information 1933
report_layout_files 1935
report_layout_hierarchy 1937
report_layout_rules 1939
report_lbist_configuration 1944
report_lbist_pins 1946
report_lfsr_connections 1948
report_lfsrs 1949
report_licenses 1950
report_lists 1951
report_load_unload_timing_options 1952
report_loadboard_loopback_pairs 1954
report_loops 1955
report_lpct_configuration 1957
report_lpct_pins 1958
report_measure_cycles 1960
report_memory_cluster_configuration 1965
report_memory_dft_summary 1971
report_memory_identification 1979
report_memory_instances 1982
report_memory_repair_groups 1985
report_mismatch_sources 1987
report_misrs 1991
report_misr_connections 1993
report_module_matching 1995
report_module_matching_options 1999
report_multicycle_paths 2001
report_multiprocessing_options 2006
report_net_polygons 2008
report_nofaults 2009
report_nonscan_cells 2011
report_nonscan_models 2014

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_notest_points 2015
report_output_masks. 2017
report_pade_logical_connections 2018
report_pattern_filtering. 2020
report_pattern_sets 2021
report_pattern_trigger 2023
report_patterns 2024
report_power_data 2035
report_power_metrics 2037
report_power_control 2044
report_primary_inputs 2048
report_primary_outputs 2050
report_procedures 2052
report_processors 2056
report_read_controls 2059
report_read_verilog_options. 2060
report_register_value 2061
report_resources 2064
report_rtl_to_gates_mapping 2065
report_run_synthesis_options 2067
report_scan_cells 2070
report_scan_chains 2082
report_scan_elements 2088
report_scan_enable 2091
report_scan_groups. 2093
report_scan_models 2094
report_scan_modes 2095
report_scan_path. 2096
report_scan_polygons 2109
report_scan_segments. 2113
report_scan_volume 2116
report_seq_transparent_procedures 2120
report_sequential_fault_depth 2122
report_shift_registers 2124
report_silicon_insight_result 2127
report_simdut_data 2129
report_simdut_faults 2131
report_simulation_contexts. 2132
report_simulation_forces 2134
report_simulation_library_sources 2136
report_ssn_configuration 2137
report_static_dft_signal_settings 2139
report_statistics 2141
report_synchronous_clock_groups 2161
report_tcd_ldb_validation. 2163

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_tcl_shell_options 2165
report_test_end_icall. 2166
report_test_logic 2167
report_test_points 2168
report_test_point_statistics 2172
report_test_setup_icall 2174
report_test_stimulus 2175
report_testbench_simulation_options 2184
report_tester_options 2185
report_tester_statistics 2186
report_tied_signals 2188
report_timeplates 2190
report_udfm_statistics. 2192
report_version_data 2198
report_wrapper_cells 2199
report_write_controls 2203
report_write_patterns_options 2204
report_xbounding 2206
report_xy_instance 2209
report_xy_net 2210
reset_attribute_value. 2211
reset_au_faults 2213
reset_bypass_chains 2215
reset_capture_timing_options. 2216
reset_compactor_connections. 2217
reset_default_ssn_datapath_configuration 2218
reset_design 2219
reset_di_faults. 2222
reset_ijtag_retargeting_options. 2225
reset_load_unload_timing_options. 2228
reset_open_pattern_set 2229
reset_physical_block_settings 2230
reset_state 2231
reset_static_dft_signal_values 2232
restore_design 2235
run_synthesis 2236
run_testbench_simulations 2245

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Command Descriptions

This section describes the Tessent Shell commands alphabetically, Report_j through R. The beginning of each command description shows the contexts and modes that support the command and provides the following information:

Context: Lists each context and sub-context that supports the command. “All contexts” means that the command is supported in all available contexts. “Unspecified” means that you do not have to set a context to use the command.

Mode: Lists each mode that supports the command. “All modes” means that the command is supported in all available modes. In some cases, a particular mode supports the command only within certain contexts, and this is noted in parentheses after the mode type.

Table 7-1. Commands Report_j Through R

Command	Description
report_layout_core_information	In hierarchical layout-aware diagnosis, lists the instance pathnames of a core stored in an instance-aware core-level LDB.
report_layout_files	Returns the flat models and pattern sets associated with RCD constants stored in the LDB.
report_layout_hierarchy	Reports the layout hierarchy and resource estimation as analyzed by the analyze_layout_hierarchy command.
report_layout_rules	Provides detailed reporting of layout rule violations.
report_lbist_configuration	Reports the global LogicBIST controller configuration parameters.
report_lbist_pins	Reports the LogicBIST controller pins using the set_lbist_pins command.
report_lfsr_connections	Displays a list of all the connections between Linear Feedback Shift Registers (LFSRs) and primary pins.
report_lfsrs	Displays a list of definitions for all the current Linear Feedback Shift Registers (LFSRs).
report_licenses	Reports the license features currently checked out by the tool.
report_lists	Displays the list of pins whose values the tool report during simulation.
report_load_unload_timing_options	Reports the load/unload timing options defined for a specified icl_module, module, or the current physical block.
report_loadboard_loopback_pairs	Reports the loopback connections and the ac_delay, if any, that were created using the add_loadboard_loopback_pairs command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 7-1. Commands Report_j Through R (cont.)

Command	Description
report_loops	Displays information about circuit loops.
report_lpct_configuration	Reports the current low pin count test controller (LPCT) parameters to the standard display output.
report_lpct_pins	Reports the low pin count test (LPCT) controller pins and connection information to the standard display output.
report_measure_cycles	Displays data for the specified components/cycles/patterns from the test patterns loaded for the current session. If nothing is specified, data for all the test patterns is reported.
report_memory_cluster_configuration	Reports the configuration of cluster memories tested in subsequent controller steps for a specified memory cluster and memory access level.
report_memory_dft_summary	Generates a DFT summary report for the MemoryBist controllers in the current design or current DftSpecification, depending on the context.
report_memory_identification	Reports the validation results of the memory module matching performed from the name patterns specified with the set_memory_identification_options command.
report_memory_instances	Reports the options of all memories within editable instances for which there is a matched TCD Memory description.
report_memory_repair_groups	Reports information about the repair groups from a DftSpecification.
report_mismatch_sources	Displays the sources of simulation mismatches.
report_misrs	Reports the MISRs you have added to your design.
report_misr_connections	Reports MISR connections.
report_module_matching	Reports design modules and their matching modules of the specified format, if the set_current_design command has already been run.
report_module_matching_options	Reports the current settings defined by the set_module_matching_options command.
report_multicycle_paths	Displays multicycle path definitions you previously read into the tool using the read_sdc command.
report_multiprocessing_options	Displays the values of all variables used when running multiprocessing commands for ATPG or fault simulation.
report_net_polygons	Context: patterns -scan_diagnosis
report_nofaults	Displays the nofault settings for the specified pin pathnames or pin names of instances.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 7-1. Commands Report_j Through R (cont.)

Command	Description
report_nonscan_cells	Displays the non-scan cells whose model type you specify.
report_nonscan_models	Displays the sequential non-scan model list.
report_notest_points	Displays all the circuit points for which you do not want the tool to insert controllability and observability.
report_output_masks	Displays a list of the currently masked primary output pins.
report_pade_logical_connections	Reports the logical paths that exist within the current design between the specified source and destination pins/ports, as well as all connections from or to the specified pins/ports. Only logical connections created by the add_pade_logical_connection command are reported by this command.
report_pattern_filtering	Displays the current pattern filtering configuration.
report_pattern_sets	Creates a report of a specified pattern set or of all pattern sets.
report_pattern_trigger	Reports pattern triggers set with the add_pattern_trigger command.
report_patterns	Displays information about patterns.
report_power_data	Reports the power data currently loaded in the system from either a read_cpf or read_upf command.
report_power_metrics	Displays shift and capture power metrics for specified test patterns.
report_power_control	The report_power_control command generates a report on power control settings for the entire design or for instances, clock domains, and partitions with power control enabled.
report_primary_inputs	Displays a list of the primary inputs of a circuit.
report_primary_outputs	Displays the specified primary outputs.
report_procedures	Displays the specified procedure.
report_processors	Displays information about the processors used for the current multiprocessing environment.
report_read_controls	Displays all of the currently defined read control lines.
report_read_verilog_options	Reports options specified with the set_read_verilog_options command.
report_register_value	Reports register information for each register name.
report_resources	Displays the machine resources used by the active tool session.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 7-1. Commands Report_j Through R (cont.)

Command	Description
report_rtl_to_gates_mapping	Reports the rules for mapping names from RTL to post-synthesis/post-layout.
report_run_synthesis_options	Reports the values for all options set with the set_run_synthesis_options command.
report_scan_cells	Displays a report on the scan cells that reside in the specified scan chains.
report_scan_chains	Displays a report on all the current scan chains.
report_scan_elements	Reports the scan elements that are created by the tool for scan insertion purposes.
report_scan_enable	Reports on scan_enable signals and associated scan chains.
report_scan_groups	Displays a report on all the current scan chain groups.
report_scan_models	Displays the sequential scan models currently in the scan model list.
report_scan_modes	Reports all available scan modes after they are inserted by Tessent Scan.
report_scan_path	Traces scan paths and reports the scan cells and combinational logic cells present in the given scan chain.
report_scan_polygons	Traces scan chains and reports the cell and output net polygons for inter-scan cell logic located along the traces.
report_scan_segments	Reports the scan segment declarations that are defined either via add_scan_segments command or tcd_scan files.
report_scan_volume	Displays the volume of scan data used by the pattern set.
report_seq_transparent_procedures	Displays a list of seq_transparent test procedures along with the associated data that you specify.
report_sequential_fault_depth	Displays estimates of maximum test coverage possible with different sequential depth settings.
report_shift_registers	Reports the identified shift registers in the design after switching to analysis mode.
report_silicon_insight_result	Reports the summary diagnosis results for failing flops and pseudo failing flops for ATPG.
report_simdut_data	Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight
report_simdut_faults	Reports the SimDUT faults.
report_simulation_contexts	Lists the available simulation contexts and indicates the current simulation context.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 7-1. Commands Report_j Through R (cont.)

Command	Description
report_simulation_forces	Lists the active forces on the specified gate_pin objects.
report_simulation_library_sources	Reports the values specified with the set_simulation_library_sources command.
report_ssn_configuration	Reports the current SSN configuration settings.
report_static_dft_signal_settings	Reports the setting specified or implied with the set_static_dft_signal_values command.
report_statistics	Displays a detailed statistics report to the screen.
report_synchronous_clock_groups	Displays a list of all user-defined synchronous clock groups.
report_tcd_ldb_validation	Verifies that the core instance pathnames and hierarchy in the TCD matches the core instance pathnames and hierarchy in the LDB.
report_tcl_shell_options	Lists the currently specified Tcl shell options.
report_test_end_icall	Reports a single view of all existing iCalls specified with the set_test_end_icall command.
report_test_logic	Displays the test logic that the tool added during the scan insertion process.
report_test_points	Displays test points that are inserted with the insert_test_logic command.
report_test_point_statistics	Use this command to report statistics regarding the location of test points inside the design.
report_test_setup_icall	Reports a single view of all existing iCalls specified with the set_test_setup_icall command.
report_test_stimulus	Displays the stimulus necessary to satisfy the specified set, write, or read conditions.
report_testbench_simulation_options	Reports the default values for many options used in the run_testbench_simulations command.
report_tester_options	Enables you to confirm which tester-specific options are set to control how the tool writes tester patterns when you use SSN.
report_tester_statistics	Reports tester statistics, such as tester cycles and test time.
report_tied_signals	Displays a list of the tied floating signals and pins.
report_timeplates	Displays the specified timeplate.
report_udfm_statistics	Reports detailed fault coverage based on the UDFM defect models.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 7-1. Commands Report_j Through R (cont.)

Command	Description
report_version_data	Displays the current software version information.
report_wrapper_cells	Reports information about the identified wrapper cells for each I/O port subjected to wrapper cell identification.
report_write_controls	Displays the currently defined write control lines and their off-states.
report_write_patterns_options	Displays the current set of ports or current set of vector callbacks to use when writing patterns.
report_xbounding	Reports the X-sources and optionally the scan cells used in bounding. You must run this command to report X-bounding.
report_xy_instance	Context: patterns -scan_diagnosis
report_xy_net	Context: patterns -scan_diagnosis
reset_attribute_value	Resets an attribute to its default value for the design objects specified in *obj_spec*.
reset_au_faults	Resets the faults in certain untestable categories by changing their classifications.
reset_bypass_chains	Removes user-specified EDT bypass chain connections.
reset_capture_timing_options	Resets all, or a specified SSN scan operation capture procedure timing option to its default value and clears the associated is_specified flag.
reset_compactor_connections	Removes user-specified EDT compactor connections.
reset_default_ssn_datapath_configuration	Deletes all override settings of an ICL network's default SSN datapath configuration.
reset_design	Clears most design setup data, including data from previous runs, but retains the design and library in memory.
reset_di_faults	Reclassifies DI faults to the UC category.
reset_ijtag_retargeting_options	Resets all, or a specified IJTAG retargeting option to its default value and clears the associated is_specified flag.
reset_load_unload_timing_options	Resets all, or a specified SSN scan operation load/unload procedure timing option to its default value and clears the associated is_specified flag.
reset_open_pattern_set	Clears the content of the currently open pattern set.
reset_physical_block_settings	Clears the settings specified with the set_current_physical_block command.
reset_state	Resets the circuit status.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 7-1. Commands Report_j Through R (cont.)

Command	Description
reset_static_dft_signal_values	Resets the static DFT signal values specified with the set_static_dft_signal_values command.
restore_design	Discards all edits performed in insertion mode and restores the design to its state prior to entering insertion mode.
run_synthesis	Runs a synthesis manager to perform synthesis of the test logic RTL.
run_testbench_simulations	Runs a simulation manager to run a set of simulation testbenches.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_layout_core_information

Context: patterns -scan_diagnosis

Mode: analysis

In hierarchical layout-aware diagnosis, lists the instance pathnames of a core stored in an instance-aware core-level LDB.

Usage

report_layout_core_information [-chip_design_name *name*]

Description

To use this command you must have a valid LDB that you previously opened using the open_layout command.

By default, this command lists the instance pathnames for all the designs stored in the LDB.

This command reports the pathnames, placement and orientation for all core instances for all top-level SoC designs stored in the currently opened LDB. It also reports the top-level DEF design name for the SoC design set in the current tool run.

Refer to "Diagnosis for Hierarchical Design" in the *Tessent Diagnosis User's Manual* for more information.

Arguments

- -chip_design_name *name*

  An optional switch and string pair that returns only the instance pathnames for the specified design.

Examples

The following example shows the reporting format for the report_layout_core_information command.

```
report_layout_core_information
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Top design name: <top DEF design name-1>
//  Instances of <core DEF design name>
//    '<Instance Path Name-1>'  Placed at: <Placement-x>,  <Placement-y>
//    Oriented: <Orientation>
//    '<Instance Path Name-2>'  Placed at: <Placement-x>,  <Placement-y>
//    Oriented: <Orientation>
//        ...
//
//  Top design name: <top DEF design name-2>
//  Instances of <core DEF design name>
//    '<Instance Path Name-1>'  Placed at: <Placement-x>,  <Placement-y>
//    Oriented: <Orientation>
//    '<Instance Path Name-2>'  Placed at: <Placement-x>,  <Placement-y>
//    Oriented: <Orientation>
//        ...
//
//  Current design set to: <top DEF design name for current tool run>
```

Related Topics

add_layout_core_information

delete_layout_core_information

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_layout_files

Context: patterns -scan_diagnosis

Mode: setup, analysis

Returns the flat models and pattern sets associated with RCD constants stored in the LDB.

Usage

report_layout_files [-include_file_paths]

Arguments

- -include_file_paths

  An optional switch that displays the full pathnames of the files.

Description

When generating RCD constants for RCD analysis, you may run the create_feature_statistics command multiple times on different combinations of flat models and pattern sets. Similarly, you may have verified the LDB against multiple flat models. The report_layout_files command reports which flat models and pattern sets are associated with the RCD constants stored in the LDB. It also reports the flat models that have been verified against the LDB.

The report displays the following information:

- VERIFY_DB: Within the LDB, the tool generates a verification database (*.verifydb*) after layout verification against a flat model.
- RCD_DB: Within the LDB, the tool generates RCD database (*.rcddb*) files every time you run create_feature_statistics to create RCD constants.
- flat model: The flat model name associated with VERIFY_DB.
- flat MD5: The MD5 signature for the flat model.
- design MD5: The structural MD5 of the design represented in the flat model.
- pattern: The pattern filename associated with RCD_DB. There may be multiple pattern files for each RCD_DB.
- pattern MD5: The MD5 signature for the pattern.
- pattern set MD5: The MD5 signature for the collective set of patterns in RCD_DB.

Examples

In the following example, the report_layout_files report shows that there are two verification databases within the current LDB. The first one is marked Active because the loaded flat model (current design) and pattern (current pattern) match its RCD_DB MD5 signatures.

read_flat_model design0.flat.gz

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_pattern pattern.wgl.gz

open_layout layoutDB

report_layout_files

```
RCD constants match for current design & pattern:
current design : design0.flat.gz MD5: c5ce18a2cfa9a98e3654a2d52b361c43
current pattern: pattern.wgl.gz MD5: 347b200939c92c0d82c2fbd30a1074a9
-----------------------------------------------------------------------
 Active   Type             Path/MD5
-----------------------------------------------------------------------
          VERIFY_DB        c5ce18a2cfa9a98e3654a2d52b361c43.verifydb
 ******   RCD_DB           c5ce18a2cfa9a98e3654a2d52b361c43_da37.rcddb
          flat model       design0.flat.gz
          flat MD5         c5ce18a2cfa9a98e3654a2d52b361c43
          design MD5       7401fc0833c7c2c3169597dbf4d66566
          pattern          pattern.wgl.gz
          pattern          MD5347b200939c92c0d82c2fbd30a1074a9
          pattern set MD5  da37e5d36ee09142ac4f3d4c4510dee2

          VERIFY_DB        54445c6b42d8237d88a4faa67a862abc.verifydb
          RCD_DB           54445c6b42d8237d88a4faa67a862abc_f522.rcddb
          flat model       design1.flat.gz
          flat MD5         54445c6b42d8237d88a4faa67a862abc
          design MD5       7401fc0833c7c2c3169597dbf4d66566
          pattern          pattern.wgl.gz
          pattern MD5      c15c47fa389a898d73337f2bb067152b
          pattern set MD5  f522b1857b12ed8af570486d041ddb34
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_layout_hierarchy

Context: patterns -scan_diagnosis

Mode: setup

Reports the layout hierarchy and resource estimation as analyzed by the analyze_layout_hierarchy command.

Usage

report_layout_hierarchy ***layout_hierarchy_db_dir*** [-core *core_name*]

Description

This command lists the hierarchy of the layout that is stored in the layout hierarchy database that you previously generated using the analyze_layout_hierarchy command. It also provides resource estimations for LDB generation for core and top-level LDBs.

Arguments

- ***layout_hierarchy_db_dir***

  A required string that specifies the name of the layout hierarchy database that contains the results of the layout hierarchy analysis.

- -core *core_name*

  An optional switch and string pair that specifies the name of the core from which the hierarchy is reported and for which resources are estimated. By default, the tool reports the entire hierarchy from the top module down, and it estimates resources based on the top module.

Examples

The following command successfully creates a layout hierarchy database named *my_layout.hierdb* because there is only one top module, sts. The resulting report describes the DEF hierarchy tree beneath the sts top module and the resource estimates for generating the LDB.

```
analyze_layout_hierarchy my_layout
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// ---------------------------------------------------------------------------------
// Top module (1 of 1) : sts
// DieArea: minx=0.000000, miny=0.000000, maxx=1340.000000, maxy=6200.000000 in microns
// ---------------------------------------------------------------------------------
// ---------------------------------------------------
// MODULE NAME TREE  | #INSTANTIATIONS IN PARENT MODULE
// ------------------+--------------------------------
// sts               |                              --
//   gctl            |                               1
//     RingOsc_all   |                               1
//     RingOsc_lym   |                               8
//     digpll        |                               1
//       digvco      |                               1
//   lym0            |                               1
//     RingOsc_lym   |                              16
//

// ---------------------------------------------------------------------------------
// MODULE NAME | TOTAL           | #COMPONENTS | #NETS     | DIEArea     | FILENAME
//             | #INSTANTIATIONS | IN MODULE   | IN MODULE | in um       |
//             | IN DESIGN       |             |           |             |
// ------------+-----------------+-------------+-----------+-------------+--------------
//  sts        |      1          |   1710457   |  20523    | 1340 x 6200 | src/def1.gz
//  gctl       |      1          |   987266    |  13527    | 901 x 592   | src/def2.gz
// RingOsc_all |      1          |   36601     |  3566     | 131 x 132   | src/def3.gz
// RingOsc_lym |      24         |   99        |  83       | 42 x 2      | src/def4.gz
// digpll      |      1          |   893       |  454      | 33 x 20     | src/def5.gz
// digvco      |      1          |   779       |  376      | 14 x 13     | src/def6.gz
// lym0        |      1          |   751794    |  451639   | 518 x 876   | src/def7.gz
//

//  LDB resource estimation for design ls169 with 75 nets.
//  ---------------------------------------------------------------
//  LDB resource estimation             |  Verbose  |    Compact
//  ------------------------------------+-----------+--------------
//  Estimated LDB creation time (hrs)   |   0.20    |      0.10
//  Estimated LDB file size (GB)        |   3.00    |      1.50
//  Estimated peak RAM (GB)             |   2.50    |      2.50
//
//  Note: Estimation calculation assumes 'create_layout -threads 16' to create layout
database (recommended).
//  ---------------------------------------------------------------------------------
//  The hierarchical GDS file of the specified DEF file structures has been
//  created. Top design: sts
//  Execute: calibredrv
//  home/design1/def_hierarchy_info.gds -l
//  home/design1/def_hierarchy_info.gds.layerprops
//  ---------------------------------------------------------------------------------
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_layout_rules

Context: patterns -scan_diagnosis

Mode: analysis

Provides detailed reporting of layout rule violations.

Usage

report_layout_rules [-Fails_summary | -Summary | *layout_rule_id*... | *layout_rule_id-occurrence#*... | -All_fails | -MISmatch_report | -NO_LAYout_names] {[-INClude ***design_module_name***...] | [-EXClude ***design_module_name***...]} [{> | >>} *file_pathname*]

Description

Provides detailed reporting of layout rule violations. Before using this command, you must run layout verification as part of the create_layout command or use the open_layout command with an existing Layout Database.

Note

Use this command in conjunction with the create_layout command to debug layout rule violations before you create the diagnosis Layout Database.

See "Layout-Aware Diagnosis and Reporting" in the *Tessent Diagnosis User's Manual* for complete information.

Arguments

- -Fails_summary

  A switch that specifies to display the following for each user-controllable rule that resulted in a violation (fail) during layout verification:

  - Rule identification (ID)
  - Number of failures of the rule
  - Current handling status of the rule
  - Brief description of the rule

  This is the default.

- -Summary

  A switch that specifies to display a summary report of layout rule violations. When you report on all rules (report_layout_rules -Summary), the following is reported for each user-controllable rule, whether or not it resulted in a violation (fail) during layout verification:

  - Rule identification (ID)
  - Number of failures of the rule

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

  - Current handling status of the rule
  - Brief description of the rule

- *layout_rule_id*

  A repeatable string that specifies the identification literal (ID) of a particular layout rule for which you want to display the violations.

- *layout_rule_id-occurrence#*

  A repeatable string that specifies the identification literal (ID) of a particular layout rule and the violation occurrence for which you want to display the occurrence message. This argument must include the specific design rule ID (*rule_id*), the specific occurrence number of the violation, and the hyphen between them. Numbers to occurrences of layout rule violations are assigned as they are encountered; you cannot change the number assigned to a specific occurrence.

- -All_Fails

  A switch that specifies to display all occurrence messages for all occurrences of layout rule violations. The displayed information can be lengthy, as it is the same information you would get if you consecutively entered a "report_layout_rules <*layout_rule_id*>" command for each rule that had a violation. Use this switch to output a report of all violation occurrences (most likely to a log file) for later analysis.

- -MISmatch_report

  An optional switch that re-generates the layout verification rule violation report. Normally, you use this switch in conjunction with the report_layout_rules command's -INClude and -EXClude switches to filter the layout rule violations—see "Guidelines for Including or Excluding Design Modules From Mismatch Reporting" in the *Tessent Diagnosis User's Manual*.

- -NO_LAYout_names

  An optional switch that suppresses the reporting of instance/net/pin path names in layout format for selected layout rule violations. See "Instance, Net, and Pin Path Names in Layout Rule Violation Reports" in the *Tessent Diagnosis User's Manual* for complete information.

- -INClude ***design_module_name***

  A switch and repeatable string that defines an excluded area of the design's hierarchy when performing layout-to-design verification. The excluded area comprises all design elements that are not below instances of the included modules.

  The ***design_module_name*** is a space-delimited list of design modules. For example:

  ```
  report_layout_rules -include moduleA moduleB moduleD
  ```

  The -INClude and -EXClude switches are mutually exclusive.

  For detailed usage of this switch, see "Guidelines for Including or Excluding Design Modules From Mismatch Reporting" in the *Tessent Diagnosis User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -EXClude ***design_module_name***

  A switch and repeatable string that defines an excluded area of the design's hierarchy when performing layout-to-design verification. The excluded area comprises all design elements contained below instances of the excluded modules.

  The ***design_module_name*** is a space-delimited list of design modules. For example:

  ```
  report_layout_rules -include moduleA moduleB moduleD
  ```

  The -INClude and -EXClude switches are mutually exclusive.

  For detailed usage of this switch, see "Guidelines for Including or Excluding Design Modules From Mismatch Reporting" in the *Tessent Diagnosis User's Manual*.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

report_layout_rules

```
// Warning: Rule DesignCellPinMatch violated 1 times out of 226 checks.
// Warning: Rule DesignInstanceMatch violated 48 times out of 5344
// checks.
// Warning: Rule DesignNetPinMatch violated 139 times out of 18630 checks.
// Warning: Rule LayoutNetPinMatch violated 66 times out of 18557 checks.
// Warning: Rule PinExistenceMacro violated 3 times out of 44202 checks.
```

Without any switches a summary of all layout rule violations is printed. In addition to the number of violations, the total number of times a particular rule was checked is also printed. As an example, for the DesignInstanceMatch rule above, this rule was checked 5344 times (specifically there must be 5344 library cell instances in this design) and it was violated 48 out of these 5344 checks.

Example 2

report_layout_rules PinExistenceMacro

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
PinExistenceMacro: #fails=3 #checks=44202 handling=warning (pin not exist
in macro).
// Warning: PinExistenceMacro-1: The specified NET cpu_i/uOPD/nx104 is
// connected to MACRO oai222 which uses a PIN C0 which is not defined in
// the LEF files.
// Warning: PinExistenceMacro-2: The specified NET cpu_i/INTR_PRI1_7_ is
// connected to MACRO oai222 which uses a PIN C0 which is not defined in
// the LEF files.
// Warning: PinExistenceMacro-3: The specified NET cpu_i/INTR_PRI1_0_ is
// connected to MACRO oai222 which uses a PIN C0 which is not defined in
// the LEF files.
```

By specifying a rule name all the violations of that rule are printed. First a summary for the user specified rule is printed and then each rule violation is printed in detail.

Example 3

report_layout_rules PinExistenceMacro-2

```
PinExistenceMacro: #fails=3 #checks=44202 handling=warning (pin not exist
in macro).
// Warning: PinExistenceMacro-2: The specified NET cpu_i/INTR_PRI1_7_ is
// connected to MACRO oai222 which uses a PIN C0 which is not defined in
// the LEF files.
```

This prints one particular violation of a given rule.

Example 4

In this case the instance name in DFT format differs from that in layout format since the last part of the name is flattened. Since the layout is being created from the LEF/DEF the instance path name in layout format is expected to be as follows:

```
/core0/cntl_1/mac\/adder\/AND0
```

Note that in LEF/DEF naming convention individual hierarchy delimiters are escaped.

Report the corresponding Instance Rules rule violation using the following command:

report_layout_rules DesignInstanceMatch-1

```
// DesignInstanceMatch-1: Design instance /core0/cntl_1/mac/adder/AND0
// (Expected Layout Name: /core0/cntl_1/mac\/adder\/AND0) not found in
// layout.
```

You can suppress reporting of the layout name for this violation using the report_layout_rules command as follows:

report_layout_rules DesignInstanceMatch-1 -no_layout_names

```
// DesignInstanceMatch-1: Design instance /core0/cntl_1/mac/adder/AND0 not
// found in layout.
```

Related Topics

create_layout

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

open_layout

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_lbist_configuration

Context: dft -logic_bist

Mode: setup, analysis

Reports the global LogicBIST controller configuration parameters.

Usage

report_lbist_configuration [> | >> *file_pathname*] [-hardware_default_compatibility]

Description

Reports the global LogicBIST controller configuration parameters. To report the parameters for individual EDT/LogicBist blocks, use the report_edt_configurations command. This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

- -hardware_default_compatibility

  An optional switch that specifies to report hardware default configuration values. This switch is only available for fault simulation. When used in setup mode, it reports the hardware default values. When used in analysis mode, it reports these values plus fault simulation settings and compatibility on a per setting and composite basis. The tool issues a warning if you specify this switch outside of the TCD flow.

Examples

Example 1

The following example shows how to configure the LBIST controller parameters and then report on what has been specified.

```
set_lbist_controller_options -max_shift 400 -max_capture 3 -max_pattern 100000 \
        -capture {clk1P 30 clk2P 60 rstP 10}

report_lbist_configuration
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// IP version:                        1
// Max Shift cycles:                  400
// Max Capture cycles:                3
// Max pattern count:                 100000
// Capture procedures:               clk1P 30%//
//                                   clk2P 60%
//                                   rstP 10%
```

Example 2

The following snippet shows the report format that the tool returns when you specify the -hardware_default_compatibility switch while in analysis mode. The report indicates that some settings are compatible with hardware default mode and others are not.

```
//  --------------                   --------  ---------------- ----------------  ---------
//  Core Instance                    Setting   Hardware Default Fault Simulation Compatible
//  --------------                   --------  ---------------- ----------------  ---------
//  LBIST                            shift length            56               40     no
//                                   capture width            4               15     no
//                                   pattern count          500             1024     yes
//                                   masked chains            0                0     yes
//                                   NCP 1 count      154 (60%)        103 (40%)     no
//                                   NCP 2 count       26 (10%)        102 (40%)     no
//                                   NCP 3 count       76 (30%)         51 (20%)     no
//  ...
//  ...
//  --------------                   --------  ---------------- ----------------  ---------
//  Current fault simulation settings are not compatible with hardware default mode
```

Related Topics

report_edt_configurations

set_lbist_power_controller_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_lbist_pins

Context: dft -logic_bist

Mode: setup, analysis

Reports the LogicBIST controller pins using the set_lbist_pins command.

Usage

report_lbist_pins [> | >> *file_pathname*]

Description

When you have renamed pins using the set_lbist_pins -on_controller_module command, the report includes a column entitled "Name on LBIST module." By default, this column is hidden.

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

This example reports all the design pins to which the LogicBIST controller's control signals, including the TAP interface signals are connected.

report_lbist_pins

```
//
//  PinType                  Connection
//  -------                  ----------
//  Clock                    refclk
//  Scan enable              scan_en
//  TCK                      tck
//  Xbounding enable         lbist_en
//  Setup Shift Scan in      tdi
//  Setup Shift Scan out     tdo jtag/scanCfgReg_so
//  Shift DR                 - jtag/shift_dr
//  Capture DR               - jtag/capture_dr
//  Update DR                - jtag/update_dr
//  Test logic reset         - jtag/tlr
//  TAP instruction decode   - jtag/scanCfgReg_en
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

report_edt_pins

set_lbist_pins

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_lfsr_connections

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays a list of all the connections between Linear Feedback Shift Registers (LFSRs) and primary pins.

Usage

report_lfsr_connections

Description

The report_lfsr_connections command displays all of the connections between the LFSRs and the primary pins. These connections were specified with the add_lfsr_connections commands.

Arguments

None

Examples

The following example displays the connections between the LFSRs and primary pins:

```
add_lfsrs lfsr1 prpg 5 15 -serial -in
add_lfsrs lfsr2 prpg 5 13 -serial -in
add_lfsrs misr1 misr 5 11 -parallel -in
add_lfsr_taps lfsr1 1 3
add_lfsr_taps lfsr2 2 3
add_lfsr_connections scan_in.1 lfsr1 3
add_lfsr_connections scan_out.0 misr1 2
report_lfsr_connections
```

Related Topics

add_lfsr_connections

delete_lfsr_connections

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_lfsrs

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays a list of definitions for all the current Linear Feedback Shift Registers (LFSRs).

Usage

report_lfsrs

Description

The report_lfsrs command displays all of the LFSRs with their current values and tap positions that you specified using the add_lfsrs and add_lfsr_taps commands.

Arguments

None

Examples

Example 1

The following example displays the definitions of all the current LFSRs:

```
add_lfsrs lfsr1 prpg 5 15 -serial -in
add_lfsrs lfsr2 prpg 5 13 -serial -in
add_lfsrs misr1 misr 5 11 -parallel -in
report_lfsrs
```

Example 2

The following example defines a 24-bit PRPG with self-feedback cells at positions 3, 7, 12, 15, and 23 and then lists the LFSRs:

```
add_lfsrs my_prpg prpg 24 FFFFFFFF -CA
add_lfsr_taps my_prpg 3 7 12 15 23
report_lfsrs
```

Output:

```
list of LFSRs
my_prpg  PRPG  length=24  shift=serial  ca_taps=3,12,15,23
my_prpg_pre_unload_value = 111111111111111111111111
my_prpg_post_unload_value = 111111111111111111111111
```

Related Topics

add_lfsrs

add_lfsr_taps

delete_lfsrs

set_lfsrs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_licenses

Context: unspecified, all contexts

Mode: all modes

Reports the license features currently checked out by the tool.

Usage

report_licenses

Description

Reports the license features currently checked out by the tool.

Arguments

None

Examples

The following example displays the licenses currently being used by the automatic diagnosis session.

report_licenses

The following report displays:

```
Licenses:
4 yieldascandiag
1 yieldaylearn
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_lists

Context: dft -edt, patterns -scan

Mode: analysis

Displays the list of pins whose values the tool report during simulation.

Usage

report_lists [{> | >>} *file_pathname*]

Description

The report_lists command displays pins the tool is currently set up to report on during simulation. Use the add_lists command to add pins to the list and the delete_lists command to remove pins from the list.

When you switch to Setup mode, the tool discards all pins from the list.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following example displays the pins whose values the tool report during simulation:

add_lists /i_1006/o /i_1007/o
report_lists

```
2 pins are currently monitored.
i_1006/o
i_1007/o
```

Related Topics

add_lists

delete_lists

set_list_file

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_load_unload_timing_options

Context: all contexts

Mode: all modes

Reports the load/unload timing options defined for a specified icl_module, module, or the current physical block.

Usage

report_load_unload_timing_options
[-icl_modules *icl_module_names* | -modules *module_names* | -current_physical_block | -all]

Description

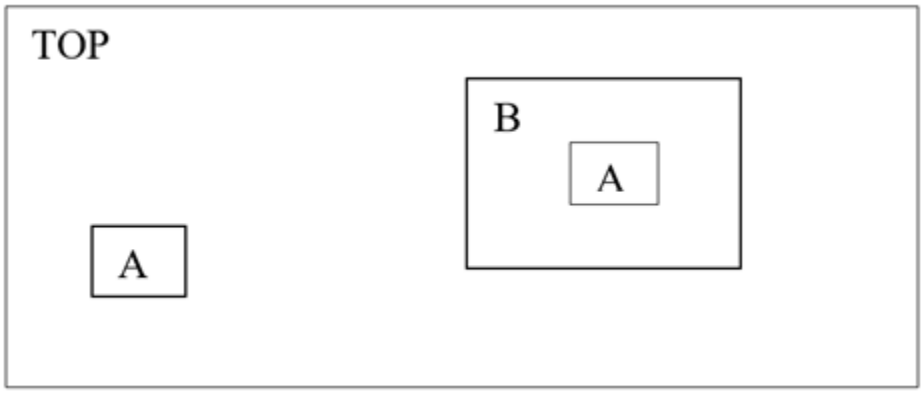
If you specified a module with the set_current_physical_block command and the load/unload options for the module, the tool reports those values. If you did not specify a module, the tool traverses the design hierarchy until it finds a parent that was specified as a physical block. Otherwise, the tool uses the module inherited from the current design, and reports the default values if no options were set.

If the module has multiple instantiations and all its instances inherit from the same parent, then the tool reports the values of that parent. If the instances of module inherit from different parents (see Figure 7-1), then the tool issues the following error:

```
//  Error: The specified Module 'foo'  was not set as a physical block by
//  'set_current_physical_block ' command. It has multiple instances that
//  do not belong to the same physical block.
```

For example, Figure 7-1 shows that module A was never specified as a physical block, and module B was specified as a physical block. Because module A has two instances, one of them inherits from block B and the other inherits from current design TOP. This scenario will result in an error.

Figure 7-1. Design Inheritance and Module Matching



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- -icl_modules *icl_module_names*

  A switch and string pair that specifies the name of a valid ICL module. You can specify more than one ICL module name.

- -modules *module_names*

  A switch and string pair that specifies the name of a module. You can specify more than one module name.

- -current_physical_block

  A switch that specifies the current physical block.

- -all

  A switch that reports the load/unload timing options for all specified physical blocks, including any previously specified using the set_current_physical_block command.

Examples

The following examples show various uses of the command:

```
report_load_unload_timing_option -icl_module xtea
report_load_unload_timing_option -module kekelo
report_load_unload_timing_option -current_physical_block
```

Related Topics

get_load_unload_timing_option

reset_load_unload_timing_options

set_load_unload_timing_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_loadboard_loopback_pairs

Context: all contexts

Modes: setup, analysis

Prerequisites: Design must be elaborated prior to using this command.

Reports the loopback connections and the ac_delay, if any, that were created using the add_loadboard_loopback_pairs command.

Usage

report_loadboard_loopback_pairs

Description

This command reports any loopback connection pairs created using the add_loadboard_loopback_pairs command. It also reports the ac_delay for the loopback pairs.

Arguments

None

Examples

This example shows the report of loopback pairs and ac_delay values for the elaborated design:

add_loadboard_loopback_pairs -inputs {din[4] ue} -outputs {dout[5] dout[0]}

report_loadboard_loopback_pairs

```
-----------------------------
Inputs  Outputs  AC Time to Z
------  -------  ------------
din[4]  dout[5]  N/A
ue      dout[0]  N/A
-----------------------------
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_loops

Context: all contexts

Mode: setup, analysis

Displays information about circuit loops.

Usage

report_loops [-All | *loop_id#…*] [{> | >>} *file_pathname*]

Description

The report_loops command displays information about currently identified loops in the circuit. For each loop, the report indicates whether the loop is a coupling loop, a single multiple fanout loop, or broken by a constant. The report also includes the pin pathname and gate type of each gate in each loop.

You can write the loops report information to a file by using the command's redirection operators or the write_loops command.

Arguments

- -All

  An optional switch that specifies to report all the loops in the circuit. This is the default.

- *loop_id#*

  An optional, repeatable, positive integer that specifies the identification number of a particular loop to report. The tool assigns loop identification numbers consecutively, starting with 1.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

Example 1

The following example displays a list of all the loops in the circuit:

```
set_system_mode analysis
report_loops
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Loop = 1: (coupling loop)
    my_design/my_minibus (SBUS)
    my_design/PAD (BUF)
    my_design/my_minibus (Z2X)
Loop = 2: (coupling loop)
...
Loop = 8: (single multiple fanout)
    my_design/al/pl/padx (BUF)
    my_design/al/pl/pad (BUF)
    my_design/pad (WIRE)
```

Example 2

The following example displays loop 8 only:

report_loops 8

```
Loop = 8: (single multiple fanout)
    my_design/al/pl/padx (BUF)
    my_design/al/pl/pad (BUF)
    my_design/pad (WIRE)
```

Example 3

The following example writes the display information for loop 8 to a new file named *my_loop_file*:

report_loops 8 > my_loop_file

```
... writing to file my_loop_file
```

Related Topics

report_feedback_paths

write_loops

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_lpct_configuration

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Reports the current low pin count test controller (LPCT) parameters to the standard display output.

Usage

report_lpct_configuration [> | >> *filename*]

Description

After saving the patterns, you can use the report_register_value command to report the different fields of the test configuration registers such as shift_length, capture_depth, scan_pattern_count, and chain_pattern_count.

For more details on the values reported by this command, see the set_lpct_controller command.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair used at the end of an argument list to create or replace the contents of ***file_pathname*** with command output.

- >> *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list to append the command output to the contents of ***file_pathname***.

Examples

The following example shows a report generated from Setup mode during EDT logic creation:

report_lpct_configuration

```
// Controller type:                Type 3
// Max Shift cycles:               1000
// Max Capture cycles:             8
// Max Scan patterns:              1048575
// Max Chain patterns:             1023
// Test mode detect sequence:      1010000101001
// Test mode detect period:        100 cycles
// Scan shift control:             internally generated clock
```

Related Topics

report_lpct_pins

set_lpct_controller

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_lpct_pins

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Reports the low pin count test (LPCT) controller pins and connection information to the standard display output.

Usage

report_lpct_pins [{> | >>} *filename*]

Description

Reports the low pin count test (LPCT) controller pins and connection information to the standard display output.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair used at the end of an argument list to create or replace the contents of ***file_pathname*** with command output.

- >> *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list to append the command output to the contents of ***file_pathname***.

Examples

The following example shows a report generated from Setup mode during EDT logic creation:

report_lpct_pins

```
//
//  Input pin            Pin name        Inversion  Internal connection
//  ---------            --------        ---------  -------------------
//  Clock                port_clock          -               -
//  Reset (Active High)        -             -      pwr_up/reset
//  Data in              lpct_data_in        -               -
//  Test mode            lpct_test_mode      -               -
//
//  Output pin        Connection
//  ----------        ----------
//  Capture enable
//  Clock mux select
//  Shift clock
//  Scan enable       scan_en
//  Reset out
//  Test end
//  Data out
//  Test active
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

report_lpct_configuration

set_lpct_controller

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_measure_cycles

Prerequisites: A WGL or STIL pattern set must be loaded.

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: analysis

Displays data for the specified components/cycles/patterns from the test patterns loaded for the current session. If nothing is specified, data for all the test patterns is reported.

Usage

report_measure_cycles [-All_cycles] [-Suite *test_suite_ID*] [{-Pattern_range | -Cycle_range} *begin end*] | [-List *cycles* …] [*observe_point*…] [> | >> *filename*]

Description

Displays data for the specified components/cycles/patterns from the test patterns loaded for the current session. If nothing is specified, data for all the test patterns is reported.

Depending on the arguments used with this command, you can report expect bits, masked bits, cycle padding, cycles, patterns, and chain cells and lengths.

You can use this command to troubleshoot data verification errors that occur before diagnosis. Only WGL and STIL test patterns are supported by this command.

Use this command to debug failure file mismatches reported by the tool. You can use this command to report the failing cycle, failing pattern, expect values, and boundary cycles related to a specific mismatch.

Arguments

- -All_cycles

  An optional switch that generates a full report for all cycles.

  The tool assigns each cycle to a specific category, and groups the neighboring cycles belonging to the same category into a cycle block, which include one or more cycles.

  For each cycle block, the tool reports the following in columnar format:

  - **First** — First cycle.
  - **Last** — Last cycle (unless the cycle is the same as the first cycle).
  - **#Cycle** — Count of cycles.
  - **Pid** — Pattern ID.
  - **Type** — Pattern type.
  - **Sid** — Suite ID. Reported if there is more than one test suite.
  - **RelPid** — Relative Pattern ID. Reported if there is more than one test suite.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

  - **Operation** — Function of the cycle block, for example “Test setup”, “Setup shift”, “Force PI”, “Measure PO” and “Shift and measure SCO”.

- -Suite *test_suite_ID*

  An optional switch and string pair that specifies the targeted test suite when you use more than one test suite. When you specify a valid test suite ID, the pattern IDs (begin end) specified are treated as relative IDs for this targeted test suite. Otherwise, the pattern IDs (begin end) are considered as absolute numbers.

  By default, if you omit this switch, then the pattern IDs (begin end) are considered as absolute numbers. If only the -suite switch is used and the -pattern is omitted, then the tool reports all patterns of the specified test suite.

- -Pattern_range *begin end*

  Optional switch and value pair that specifies a range of patterns.

- -Cycle_range *begin end*

  Optional switch and value pair that specifies a range of cycles. This option cannot be used with -All_cycles because a full cycle-oriented report is generated; the tool ignores the -all_cycles option, if supplied.

- -List *cycles*

  Optional, repeatable switch and value pair used to specify one or more specific cycles.

- *observe_point*

  Optional value that specifies the name of one or more components to focus the report on. Observe points are usually pin names, but may also represent any of the following:

  - chain
  - scanout_pin
  - edt_channel
  - edt_scanout_pin
  - POpin

  For a compressed pattern design, if an internal scan chain is used as an observe_point, no value is reported. The observe point for a compressed pattern should be either edt channel, edt channel output pin, or primary output.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*. Use this argument to write the report to a file.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*. Use this argument to append the report to the specified file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example displays a specified range of patterns:

report_measure_cycles -pattern 10 20

```
total cycles = 41103
pattern set # 1 total cycles: 41103 last chain: 9 last scan: 382
( pat/pat.wgl )
  Pattern   Measure_PO  Measure_SCO (first,last,step)
 ---------  ----------  ----------------------------
     10 (S)   2277    { (2279,2380,+1) }
     11 (S)   2381    { (2383,2484,+1) }
     12 (S)   2485    { (2487,2588,+1) }
     13 (S)   2589    { (2591,2692,+1) }
     14 (S)   2693    { (2695,2796,+1) }
     15 (S)   2797    { (2799,2900,+1) }
     16 (S)   2901    { (2903,3004,+1) }
     17 (S)   3005    { (3007,3108,+1) }
     18 (S)   3109    { (3111,3212,+1) }
     19 (S)   3213    { (3215,3316,+1) }
     20 (S)   3317    { (3319,3420,+1) }
```

Example 2

The following example displays the cycles, patterns, and expect bits information for the chain test for chain3:

report_measure_cycles chain3

```
total cycles = 41103
cycle  pattern  chain{cell}:scanout  expect
-----  -------  -------------------  ------

105     0       chain3{0}:scan_out3 L  chaintest
106     0       chain3{1}:scan_out3 L  chaintest
107     0       chain3{2}:scan_out3 H  chaintest
108     0       chain3{3}:scan_out3 H  chaintest
109     0       chain3{4}:scan_out3 L  chaintest
110     0       chain3{5}:scan_out3 L  chaintest
111     0       chain3{6}:scan_out3 H  chaintest
112     0       chain3{7}:scan_out3 H  chaintest
113     0       chain3{8}:scan_out3 L  chaintest
114     0       chain3{9}:scan_out3 L  chaintest
115     0       chain3{10}:scan_out3 H  chaintest
116     0       chain3{11}:scan_out3 H  chaintest
117     0       chain3{12}:scan_out3 L  chaintest
118     0       chain3{13}:scan_out3 L  chaintest
119     0       chain3{14}:scan_out3 H  chaintest
120     0       chain3{15}:scan_out3 H  chaintest
```

Example 3

The following example reports all cycles based on cycle blocks.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_measure_cycles -all_cycles

```
total cycles = 41103
pattern set # 1 total cycles: 41103 last chain: 9 last scan: 382
( pat/pat.wgl )

First  Last   #Cycle  Pid  Type  Operation

0             1       0    C     Test setup
1             1       0    C     Setup shift
2      103    102     0    C     Shift (load only)
104           1       0    C     Setup shift
105    206    102     0    C     Shift and measure SCO
207           1       1    C     Setup shift
208    309    102     1    C     Shift and measure SCO
310           1       2    C     Setup shift
311    412    102     2    C     Shift and measure SCO
413           1       3    C     Setup shift
414    515    102     3    C     Shift and measure SCO
516           1       4    C     Setup shift
517    618    102     4    C     Shift and measure SCO
619           1       5    C     Setup shift
620    721    102     5    C     Shift and measure SCO
722           1       6    C     Setup shift
723    824    102     6    C     Shift and measure SCO
825           1       7    C     Setup shift
826    927    102     7    C     Shift and measure SCO
928           1       8    C     Setup shift
929    1030   102     8    C     Shift and measure SCO
1031          1       9    C     Setup shift
1032   1133   102     9    C     Shift and measure SCO
1134          1       0    S     Setup shift
1135   1236   102     0    S     Shift (load only)
1237          1       0    S     Measure PO and pulse clock
1238          1       0    S     Setup shift
1239   1340   102     0    S     Shift and measure SCO
1341          1       1    S     Measure PO and pulse clock
```

Example 4

The following example reports the measurement cycles for test suite 1.

report_measure_cycles -suite 1

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
total cycles = 82206
pattern set # 1 total cycles: 41103 last chain: 9 last scan: 382
( pat/pat.wgl )
pattern set # 2 total cycles: 41103 last chain: 9 last scan: 382
( pat/pat.wgl )
  Pattern (Suite/Pid)  Measure_PO  Measure_SCO (first,last,step)
 --------------------  ----------  ------------------------------
      0 ( 1/   0) (C)   ----    { (105,206,+1) }
      1 ( 1/   1) (C)   ----    { (208,309,+1) }
      2 ( 1/   2) (C)   ----    { (311,412,+1) }
      3 ( 1/   3) (C)   ----    { (414,515,+1) }
      4 ( 1/   4) (C)   ----    { (517,618,+1) }
      5 ( 1/   5) (C)   ----    { (620,721,+1) }
      6 ( 1/   6) (C)   ----    { (723,824,+1) }
      7 ( 1/   7) (C)   ----    { (826,927,+1) }
      8 ( 1/   8) (C)   ----    { (929,1030,+1) }
      9 ( 1/   9) (C)   ----    { (1032,1133,+1) }
      0 ( 1/   0) (S)   1237    { (1239,1340,+1) }
      1 ( 1/   1) (S)   1341    { (1343,1444,+1) }
      2 ( 1/   2) (S)   1445    { (1447,1548,+1) }
      3 ( 1/   3) (S)   1549    { (1551,1652,+1) }
      4 ( 1/   4) (S)   1653    { (1655,1756,+1) }
      5 ( 1/   5) (S)   1757    { (1759,1860,+1) }
      6 ( 1/   6) (S)   1861    { (1863,1964,+1) }
```

Example 5

The following example reports in blocks, all cycles between 0 and 200, and shows that cycles 105-206 are measure cycles for the first chain pattern.

report_measure_cycles -cycle_range 0 200

```
total cycles = 41103
pattern set # 1 total cycles: 41103 last chain: 9 last scan: 382
( pat/pat.wgl )

First  Last   #Cycle  Pid  Type  Operation

0             1       0    C     Test setup
1             1       0    C     Setup shift
2      103    102     0    C     Shift (load only)
104           1       0    C     Setup shift
105    206    102     0    C     Shift and measure SCO
```

Related Topics

diagnose_failures

read_patterns

set_diagnosis_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_memory_cluster_configuration

Context: dft, patterns

Mode: all modes

Reports the configuration of cluster memories tested in subsequent controller steps for a specified memory cluster and memory access level.

Usage

Context: dft

```
report_memory_cluster_configuration
    { [-design_name                  design_name] }
      [-design_id                    design_id]
      [-controller_id                controller_id]
      [-cluster_id                   cluster_id] }
    | { [-tcd_memory_cluster_name    cluster_module_name]
        [-memory_access_level  {from_defaults_specification | logical | auto | physical} ] }
```

Context: patterns

```
report_memory_cluster_configuration
    { [-design_name                  design_name]
      [-design_id                    design_id]
      [-pattern_id                   pattern_id] }
      [-patterns_name                patterns_name]
      [-test_step_name               test_step_name]
      [-controller_icl_instance      controller_icl_instance] }
    | [-tcd_memory_bist_controller_name  tcd_memory_bist_controller_name]
```

Description

Creates a tabular report with steps and lists of cluster memories tested in consecutive controller steps. In the dft context, the report is based on the memory cluster library description and the selected memory access level. In the patterns context, the report is based on the MemoryBIST controller TCD description associated with the given MemoryBIST controller instance.

Context : dft

The command accepts a minimum set of options to uniquely identify the reported memory cluster instance in either the DftSpecification or the /Core/MemoryCluster, in case the DftSpecification was not loaded. When a design contains a single memory cluster, it is not necessary to specify any options with the command. When multiple memory clusters exist in a design, an error is reported if it is not possible to uniquely identify a memory cluster_id with the provided command options. In this case, the DftSpecification can be consulted with the report_config_data command to identify the available memory clusters.

If the reported memory cluster is included in the DftSpecification, it is accompanied with the memory access level value defined either explicitly with the /DftSpecification/MemoryBist/

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

MemoryCluster/memory_access_level property, or inferred from the /DefaultsSpecification/DftSpecification settings.

If no DftSpecification is present, Tessent Shell reports the memory cluster configuration of the /Core/MemoryCluster defined in the tcd partition. Similarly to the DftSpecification generation, the cluster memory access level is assumed from the DefaultsSpecification, unless the "-memory_access_level" option is used.

Context : patterns

The command accepts a minimum set of options to uniquely identify a MemoryBIST controller TCD loaded in the tcd partition. Alternately, you may directly specify a controller TCD using the "-tcd_memory_bist_controller_name" option. When a design contains a single memory cluster, it is not necessary to specify any options with the command. When multiple memory clusters exist in a design, an error is reported if it is not possible to uniquely identify a memory cluster_id with the provided command options. In this case, the DftSpecification can be consulted with the report_config_data command to identify the available memory clusters.

Arguments

- -design_name *design_name*

  An optional switch and value pair that specifies which DftSpecification or PatternsSpecification wrapper the reported memory cluster is located in. This is used in case multiple DftSpecifications or PatternsSpecifications for different design names are loaded in memory. In the dft context, this switch is mutually exclusive with the "-tcd_memory_cluster_name" switch. In the patterns context, this switch is mutually exclusive with the "-tcd_memory_bist_controller_name" switch.

- -design_id *design_id*

  An optional switch and value pair that narrows down the search of the DftSpecification or PatternsSpecification when multiple DftSpecifications or PatternsSpecifications are loaded in memory for the same design. In the dft context, this switch is mutually exclusive with the "-tcd_memory_cluster_name" switch. In the patterns context, this switch is mutually exclusive with the "-tcd_memory_bist_controller_name" switch.

- -controller_id *controller_id*

  An optional switch and value pair to narrow down the search to only a single DftSpecification/MemoryBist/Controller wrapper containing the reported memory cluster. This switch is mutually exclusive with the "-tcd_memory_cluster_name" switch.

- -cluster_id *cluster_id*

  An optional switch and value pair that informs which /DftSpecification/MemoryBist/Controller/MemoryCluster to select for the memory cluster report. This switch is mutually exclusive with the "-tcd_memory_cluster_name" switch.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -tcd_memory_cluster_name *cluster_module_name*

  An optional switch and value pair to reference the reported memory cluster in /Core/MemoryCluster. This switch is mutually exclusive with the following switches referencing DftSpecification location: “-design_name”, “-design_id”, “-controller_id”, “-cluster_id”.

- -memory_access_level {from_defaults_specification | logical | auto | physical}

  An optional switch and enumeration symbol pair used together with the “-tcd_memory_cluster_name” switch to assume the given cluster memory access level when generating the report. If the switch is not provided, Tessent Shell assumes the access level currently defined in the /DefaultsSpecification/DftSpecification.

- -pattern_id *pattern_id*

  An optional switch and value pair to specify a particular PatternsSpecification to further narrow down the search for the memory cluster report. This switch is mutually exclusive with the “-tcd_memory_bist_controller_name” switch and is only available in the patterns context.

- -patterns_name *patterns_name*

  An optional switch and value pair to specify a particular /PatternsSpecification/Patterns wrapper to further narrow down the search for the memory cluster report. This switch is mutually exclusive with the “-tcd_memory_bist_controller_name” switch and is only available in the patterns context.

- -test_step_name *test_step_name*

  An optional switch and value pair to specify a particular /PatternsSpecification/Patterns/TestStep wrapper to further narrow down the search for the memory cluster report. This switch is mutually exclusive with the “-tcd_memory_bist_controller_name” switch and is only available in the patterns context.

- -controller_icl_instance *controller_icl_instance*

  An optional switch and value pair to specify a particular /PatternsSpecification/Patterns/TestStep/MemoryBist/Controller wrapper to further narrow down the search for the memory cluster report. This switch is mutually exclusive with the “-tcd_memory_bist_controller_name” switch and is only available in the patterns context.

- -tcd_memory_bist_controller_name *tcd_memory_bist_controller_name*

  An optional switch and value pair to reference the reported memory controller in the /Core/MemoryBistController configuration data. This switch only available in the patterns context, and is mutually exclusive with the following switches referencing PatternsSpecification location: “-patterns_name”, “-pattern_id”, “-test_step_name”, and “-controller_icl_instance”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1 (Context: dft)

Given the following DftSpecification:

```
DftSpecification(WIRELESS_CORE,rtl) {
  MemoryBist {
    Controller(c1) {
      MemoryCluster(c1) {
        instance_name: top/a/b/mc;
        memory_access_level: physical;
      }
    }
  }
}
```

The following command can be used to report the test scheduling for memory cluster instance top/a/b/mc:

report_memory_cluster_configuration -design_name WIRELESS_CORE -controller_id c1

Note

The cluster is tested at the physical memory access level in this example.

Example 2 (Context: dft)

Assuming no design is loaded, but the following DftSpecification exists (no_insertion mode):

```
DftSpecification(WIRELESS_CORE,rtl) {
  MemoryBist {
    Controller(c1) {
      MemoryCluster(c1) {
        memory_cluster_library_name: CLUSTER;
      }
    }
  }
}
```

The same command given in Example 1 can be used to print the test scheduling. Please note however, this time the memory access level defined in the /DefaultsSpecification/ DftSpecification applies.

Example 3 (Context: dft)

If the DftSpecification is not available, one may print the test scheduling with the following command:

**report_memory_cluster_configuration -tcd_memory_cluster_name CLUSTER \
-memory_access_level physical**

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4 (Context: patterns)

Given the following PatternsSpecification loaded in memory:

```
PatternsSpecification(WIRELESS_CORE,rtl,signoff) {
  Patterns(ICLNetwork) {
    ICLNetworkVerify(WIRELESS_CORE) {
    }
  }
  Patterns(MemoryBist_P1) {
    TestStep(run_time_prog) {
      MemoryBist {
        run_mode : run_time_prog;
        reduced_address_count : off;
        Controller(mbist_c1_controller_inst) {
          DiagnosisOptions {
            compare_go : on;
            compare_go_id : on;
          }
        }

Controller(mbist_c2_shared_bus_assembly_inst.mbist_c2_controller_inst) {
          DiagnosisOptions {
            compare_go : on;
            compare_go_id : on;
          }
        }
      }
    }
  }
}
```

The following command invocation can be used to print the report for the second controller:

```
report_memory_cluster_configuration -controller_icl_instance \
      mbist_c2_shared_bus_assembly_inst.mbist_c2_controller_inst
```

Example 5 (Context: patterns)

If the controller TCD is loaded in the tcd partition in memory, the report can be printed with the following command:

```
SETUP> report_memory_cluster_configuration \
             -tcd_memory_bist_controller_name mbist_c2_controller
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This command results in the following table report:

```
// controller id: mbist_c2_controller
// ==============================================================
// ----  -------  -------  --------  ---------  -------  --------  --------
// Step  Access   Logical  Physical  Cluster    Logical  Physical  Memory
//       level    access   access    interface  memory   memory    instance
//                code     code      id         id       id        id
// ----  -------  -------  --------  ---------  -------  --------  --------
// 0     logical  3'b001   -         I1         LM_0               I1_LM_0
// 1     logical  3'b010   -         I1         LM_1               I1_LM_1
// 2     logical  3'b011   -         I1         LM_2               I1_LM_2
// 3     logical  3'b100   -         I1         LM_3               I1_LM_3
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_memory_dft_summary

Context: patterns -ijtag, dft

Mode: all modes

Generates a DFT summary report for the MemoryBist controllers in the current design or current DftSpecification, depending on the context.

Usage

report_memory_dft_summary [-return_dictionary]

Description

The report_memory_dft_summary command generates a complete DFT summary report, from ICL file and TCD information, for all MemoryBist controllers in the current design when invoked in the patterns -ijtag context.

When the report_memory_dft_summary command is invoked in the dft context, a DFT summary report is generated for the MemoryBist controllers defined in the current DftSpecification. The process_dft_specification -validate_only command must be run prior to invoking report_memory_dft_summary in order to generate the required DFT summary information.

The information provided in the summary report can also be used to experiment with memory partitioning. The test time and power metrics provided for each memory instance, controller step, and controller can be used to guide the adjustment of partitioning constraints available through the DefaultsSpecification or memory instance options to achieve the desired DftSpecification.

Arguments

- -return_dictionary

  An optional switch that specifies the report will be returned as a Tcl dictionary you can use for custom formatting. The dictionary structure is outlined in the following figures.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 7-2. report_memory_dft_summary Tcl Dictionary Structure

```
SummaryInfo {
  version int
  current_design mod

  DesignInstance {
    ## ICL instance of module with tessent_design_level attribute.
    ## '-' denotes the current design.
    icl_inst_or_dash {
      module_name mod
      design_instance inst_or_dash ## '-'denotes the current design
      design_level chip | sub_block | physical_block

      DesignId {
       ## Identifier assigned to DFT insertion pass on design module
        design_id {
          ## BAP information from the same DFT insertion pass.
          ## Included for controller or BAP direct access
          ## interface only.
          BistAccessPort {
            icl_inst {
              design_instance inst
              module_name mod
              DirectAccess {
               ## Maximum values for the ExecutionSelection feature
               ## among all controllers attached to this BAP.
               controller_selection_size int
               operation_set_selection_size int
               algorithm_selection_size int
               memory_selection_size int
              }
            }
          }

          ## Controller information from the same DFT insertion pass
          Controller {
            ## Controller's ICL instance or identifier
            ## from DftSpecification.
            icl_inst_or_id {
              ## Refer to Figure 7-3 for the Controller
              ## dictionary structure.
            }
            ...
          }
        } ## end design_id
        ...
      }
    } ## end icl_inst_or_null
    ...
  }
} ## end SummaryInfo
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 7-3. Controller Tcl Dictionary Structure

```
Controller {
  ## Controller's ICL instance or identifier from DftSpecification
  icl_inst_or_id {
    ## MBIST clock selected at DFT insertion
    bist_clock_period_ns real
    bist_clock_connection port_or_pin
    ## Reference clock sourcing the MBIST clock
    reference_clock_period_ns real
    reference_clock_connection port_or_pin
    ## Controller's design module name and hierarchical instance
    module_name mod
    design_instance inst
    ## Attached BAP's design module name and hierarchical instance
    bap_module_name mod
    bap_design_instance inst
    ## Unique BAP output that enables the controller
    bist_port pin
    ## Maximum average power among all controller steps
    power_mw real
    ## Estimated total algorithm execution time
    run_time_ns real
    ## Estimated total algorithm execution cycles
    test_cycles int
    ## Estimated register count in MBIST logic
    flop_count int
    ## Number of soft instructions for soft programmable controller
    soft_instruction_count int

    Step {
      ## Memories tested concurrently in the same controller step.
      ## The first step begins at 0.
      int {
        StepInfo {
          ## Estimated average power in milliwatts
          power_mw real
          ## Estimated algorithm execution time in nanoseconds
          run_time_ns real
          ## Estimated algorithm execution cycles
          test_cycles int
          comparator_location shared_in_controller | per_interface
        }
        MemoryInstance {
          icl_inst {
            ## Memory's design module name and hierarchical instance
            design_instance inst
            module_name mod
            ## Estimated average power during algorithm execution
            power_mw real
            type sram | rom | dram
            ## Number of read-only, write-only, and read/write ports
            logical_ports xRyWzRW
            ## Number of algorithm passes to cover all read and
            ## write logical ports.
            test_ports int
            ## Identifier from DftSpecification
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
                interface_id string


                MemoryInterfaceInfo {
                  ## Associated memory interface's design module name
                  ## and hierarchical instance.
                  design_instance inst
                  module_name mod
                }
                ## Included for controller direct access interface only
                DirectAccess {
                  ## Bit index on memory selection bus that enables
                  ## this memory.
                  selection_bit int
                }
                ## Spare element information for repairable memory
                RepairInfo {
                  type row | io | row_column
                  spare_rows int
                  spare_io int
                  spare_columns int
                  repair_group string
                }
              } ## end icl_inst
              ...
            } ## end MemoryInstance
          } ## end int
          ...
        } ## end Step

        ## Selection code of available hardcoded algorithms.
        ## Included for controller or BAP direct access interface only.
        Algorithm {
          algo_name int
        }

        ## Selection code of available hardcoded operation sets.
        ## Included for controller or BAP direct access interface only.
        OperationSet {
          opset_name int
        }
      } ## end icl_inst_or_id
      ...
    } ## end Controller
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The report_memory_dft_summary command is issued on the lower-level module named blka described in Example 2. The following is the summary report that is produced:

```
// DFT Summary for MemoryBist Controller
// blka_i1.unita_i1_blka_rtl_tessent_mbist_c1_controller_inst
// ==========================================================================================
// ----  -----------  -------  ---------  ----------------------------------------------
// Step  Test Time    Power    Interface  Memory Instances
// ----  -----------  -------  ---------  ----------------------------------------------
// ALL   7274.07us    38.46mW
// 0     962078.0ns   23.08mW  m1_d12     blka_i1/unita_i1/SYNC_1R1W_512x12_a1
//                             m2_d10     blka_i1/unita_i1/SYNC_1RW_256x10_a1
//                             m3_d12     blka_i1/unita_i1/SYNC_1R1W_512x12_a2
// 1     2347852.0ns  38.46mW  m4_d10     blka_i1/unita_i1/SYNC_1RW_256x10_a2
//                             m5_d16     blka_i1/unita_i1/SYNC_2RW_512x16_a1
//                             m6_d10     blka_i1/unita_i1/SYNC_1RW_256x10_a3
// 2     3668808.0ns  30.77mW  m7_d16     blka_i1/unita_i1/SYNC_2RW_512x16_a2
//                             m8_d10     blka_i1/unita_i1/SYNC_1RW_256x10_a4
// 3     295334.0ns   7.69mW   m9_d12     blka_i1/unita_i1/SYNC_1R1W_512x12_a3
//
```

Example 2

A bottom-up DFT flow was applied to the design in this example. Memory BIST was first inserted into the lower-level module named blka, which contains one controller named c1. The top module “chip” instantiates module blka. Memory BIST was next inserted into the top module “chip”, which contains two memory BIST controllers named c2 and c3. The report_memory_dft_summary command was invoked with the -return_dictionary argument from the top-level module “chip”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following is the overall dictionary, without the Controller sections:

```
SummaryInfo {
  version 1
  current_design chip

  - { ## Current design
    module_name chip
    design_level chip
    rtl { ## Design ID
      BistAccessPort {
        chip_rtl_tessent_mbist_bap_inst { ## ICL instance
          design_instance chip_rtl_tessent_mbist_bap_inst
          module_name chip_rtl_tessent_mbist_bap
        }
      }
      Controller {
        chip_rtl_tessent_mbist_c2_controller_inst { ## ICL instance
          ...
        }
        chip_rtl_tessent_mbist_c3_controller_inst { ## ICL instance
          ...
        }
      }
    }
  }

  blka_i1 {
   module_name blka
    design_level sub_block
    design_instance blka_il
    rtl { ## Design ID
      BistAccessPort {
        blka_i1.blka_rtl_tessent_mbist_bap_inst { ## ICL instance
          design_instance blka_i1/blka_rtl_tessent_mbist_bap_inst
          module_name blka_rtl_tessent_mbist_bap
        }
      }
      Controller {
        ## ICL instance
        blka_i1.unita_i1_blka_rtl_tessent_mbist_c1_controller_inst {
          ...
        }
      }
    }
  }
}
```

The following is the sub-dict for the top-level memory BIST controller c2. Three memories are tested in parallel by this controller.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Controller {
  chip_rtl_tessent_mbist_c2_controller_inst { ## ICL instance
    bist_clock_period_ns 8.0
    design_instance chip_rtl_tessent_mbist_c2_controller_inst
    module_name chip_rtl_tessent_mbist_c2_controller
    power_mw 37.5
    run_time_ns 590032.0
    test_cycles 73754
    bap_module_name chip_rtl_tessent_mbist_bap
    bap_design_instance chip_rtl_tessent_mbist_bap_inst
    bist_port chip_rtl_tessent_mbist_bap_inst/bistEn[0]


    Step {
      0 {
        StepInfo {
          power_mw 37.5
          run_time_ns 590032.0
          test_cycles 73754
        }
        MemoryInstance {
          SYNC_1R1W_512x12_c1 { ## ICL instance
            design_instance SYNC_1R1W_512x12_c1
            module_name SYNC_1R1W_512x12
            power_mw 12.5
            type sram
            interface_id m1_d12

            MemoryInterfaceInfo {
              design_instance SYNC_1R1W_512x12_c1_interface_inst
              module_name chip_rtl_tessent_mbist_c2_interface_m1_d12
            }
          }

          SYNC_1RW_256x10_c1 { ## ICL instance
            design_instance SYNC_1RW_256x10_c1
            module_name SYNC_1RW_256x10
            power_mw 12.5
            type sram
            interface_id m2_d10

            MemoryInterfaceInfo {
              design_instance SYNC_1RW_256x10_c1_interface_inst
              module_name chip_rtl_tessent_mbist_c2_interface_m2_d10
            }
          }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
            SYNC_1R1W_512x12_c2 { ## ICL instance
              design_instance SYNC_1R1W_512x12_c2
              module_name SYNC_1R1W_512x12
              power_mw 12.5
              type sram
              interface_id m3_d12

              MemoryInterfaceInfo {
                design_instance SYNC_1R1W_512x12_c2_interface_inst
                module_name chip_rtl_tessent_mbist_c2_interface_m3_d12
              }
            }
          } ## MemoryInstance
        }
      } ## Step
    }
    chip_rtl_tessent_mbist_c3_controller_inst { ## ICL instance
      ...
    }
  }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_memory_identification

Context: dft

Mode: analysis, insertion

Reports the validation results of the memory module matching performed from the name patterns specified with the set_memory_identification_options command.

Usage

report_memory_identification

Description

Creates a tabular report showing the memory module matching results. It is required to have previously used the set_memory_identification_options command with a valid match expression before requesting the report created by report_memory_identification.

The report contains a Memory Identification Report table with the following information:

- **Design module** — The name of the memory design module identified either by the pattern(s) specified with the set_memory_identification_options command, or recognized by the tool by matching a provided memory TCD.
- **TCD name** — If found, the name of the memory TCD associated with the design memory module is reported.
- **Pattern match** — A “yes” or “no” report as to whether a design memory module matches any pattern specified with the set_memory_identification_options command.
- **Instance count** — The number of instances associated with a design memory module.
- **Action** — Recommended actions for the user to review. The reported actions are:
  - **missing pattern** — The listed design module has not been identified by a matching pattern expression specified with the set_memory_identification_options command, but has been recognized as memory because the memory TCD for the module exists. The pattern matching expression specified for set_memory_identification_options needs to be verified.
  - **missing TCD** — The listed design module has been identified as memory by a matching pattern expression specified with the set_memory_identification_options command, but the corresponding memory TCD has not been found. The memory TCD needs to be provided, or the pattern expression needs to be updated if the design module is not a memory module.

The report also contains a Matching Patterns Expressions table with the following information:

- **Matching pattern expression** — The regular or wildcard matching pattern specified with the set_memory_identification_options command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **Matched modules count** — The number of design modules that are matched by the corresponding regular or wildcard matching pattern.

Arguments

None

Examples

During the setup mode for this example design, the following design module matching patterns are specified:

set_memory_identification_options "{(.*SYNC_1.*)} {(.*SYNC_2.*)} {(.*SYNC_3.*)}" -regexp

When check_design_rules is run, the report is automatically generated if errors or warning exist between design module matching and memory TCD.

Later in the flow, while in either the analysis or insertion mode, the report_memory_identification command can be run to view the report as shown in the example below:

report_memory_identification

```
// Memory identification report table
// ==================================
// --------------  --------  -------------  --------------  -----------
// Design module   TCD name  Pattern match  Instance count  Action
// --------------  --------  -------------  --------------  -----------
// SYNC_1R1W_16x8  -         yes            5               missing TCD
// SYNC_2R2W_12x8  -         yes            3               missing TCD
//
//  Actions legend:
//
//   Action 'missing TCD':
//     Memory has been identified by matching pattern expression, but
//     corresponding memory TCD library has not been found. Provide the
//      missing TCD library file or correct the matching pattern expression
//     if identified design module is not a memory module.


// Matching pattern expressions table
// ==================================
// ---------------------------  ---------------------
// Matching pattern expression  Matched modules count
// ---------------------------  ---------------------
// (.*SYNC_1.*)                 1
// (.*SYNC_2.*)                 1
// (.*SYNC_3.*)                 0
//
//  Error: Memory validation by matching pattern expression (enabled with
//         set_memory_identification_options) has identified potentially
//         untested memories due to missing TCD library file.
//         Refer to the above memory identification report table.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

set_memory_identification_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_memory_instances

Context: dft

Mode: all modes

Prerequisite: The current design must be set with the set_current_design command.

Reports the options of all memories within editable instances for which there is a matched TCD Memory description.

Usage

report_memory_instances [*name_patterns*] [-below_instances *below_instances*]
[-filter *filter*] [-limit *limit*]
[-silent]

Description

Reports the options of all memories, for which there is a matched TCD Memory description, that are inside editable modules and for which the user has the opportunity to insert DFT. See "Tessent Core Description" on page 4793 for further information on TCD Memory descriptions.

Memory instances within a lower level physical block, and any instance that has the "is_editable" attribute set to "false", are excluded from the reporting. You can restrict the report to instances below a given collection of instances, and filter the collection based on an attribute filtering equation involving other attributes registered against or inherited by the instance object type.

The -silent option suppresses the warning that is normally generated if there are no memory instances to report.

The "get_memory_instances -filter tcd_memory_name" command and option can be used to list all memories in the current design. Note that this command only list memory instances and does not return any DFT specific information, as does the report_memory_instance command.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of instances. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no name_patterns are specified, the tool searches all instances.

  If you specify the -below_instances options, the name patterns are hierarchical name patterns relative to the specified instance objects; otherwise, the name patterns are relative to the current design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for instances below the specified instance_objects. The specified name_patterns are searched relative to each instance found in the instance_objects list.

- -filter *attribute_equation*

  An optional switch and string pair that specifies to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the instance object type. See "Attribute Filtering Equation Syntax" on page 3832 for more details on the attribute filtering equation format.

- -limit *limit*

  An optional switch and number pair that defines the maximum number of memory instances the tool should report. The default of limit is 10.

- -silent

  An optional switch that suppresses the warning message normally generated when the command does not find any instances to report. The warning message is intended to indicate a probable mistake when specifying patterns. You can use the -silent switch to suppress the warning. This -silent option has no effect on any other error reporting.

Examples

The following example uses the report_memory_instances command to report the options of the first three instances of the memory module.

```
SETUP> report_memory_instances -limit 3
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//
// Memory Instance: blockA_l1_i1/blockA_l2_i1/mem1
// ----------------------------------------------
// bist_data_in_pipelining              : off
// physical_cluster_override            :
// power_domain_island                  :
// test_clock_override                  :
// use_in_memory_bist_dft_specification : auto
// use_in_memory_bisr_dft_specification : auto
//
// Memory Instance: blockA_l1_i1/blockA_l2_i1/mem2
// ----------------------------------------------
// bist_data_in_pipelining              : off
// physical_cluster_override            :
// power_domain_island                  :
// test_clock_override                  :
// use_in_memory_bist_dft_specification : auto
// use_in_memory_bisr_dft_specification : auto
//
// Memory Instance: blockA_l1_i1/blockA_l2_i1/mem3
// ----------------------------------------------
// bist_data_in_pipelining              : off
// physical_cluster_override            :
// power_domain_island                  :
// test_clock_override                  :
// use_in_memory_bist_dft_specification : auto
// use_in_memory_bisr_dft_specification : auto
//
// Reached limit of 3, skipping remaining 15 instances.
```

Related Topics

get_memory_instances

get_memory_instance_option

set_memory_instance_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_memory_repair_groups

Context: dft

Mode: setup, analysis

Reports information about the repair groups from a DftSpecification.

Usage

report_memory_repair_groups [-show_memory_instances]
{[-design_name *design_name* -insertion_id *id*]}

Description

The report_memory_repair_groups command reports information about the repair groups to be implemented from a DftSpecification. Repair groups are present only if repair sharing is specified. For more information on memory repair sharing, refer to the "Repair Sharing" topic in *Tessent MemoryBist User's Manual For Use With Tessent Shell*. Repair group information for shared bus memories are not reported if the command is run prior to process_dft_specification, since the shared bus repair groups do not exist until the DftSpecification is processed.

By default, the command reports the repair groups from the DftSpecification for the current elaborated design and insertion id. If it is desired to report the repair groups from a DftSpecification of a different design than the one that is currently elaborated, or if there isn't one currently elaborated, the -design_name and -insertion_id arguments must be specified.

The report contains the controller for each group and its size, expressed in terms of total memory bits that are sharing a spare element. If the -show_memory_instances argument is specified, the report also provides an additional list of all memory instances within each repair group.

Arguments

- -show_memory_instances

  An optional switch that specifies a list of all memory instances with each repair group is also reported.

- -design_name *design_name*

  An optional switch and value pair that specifies the module design name from which the repair groups are reported. The -insertion_id argument must also be specified to uniquely identify the design module.

- -insertion_id *id*

  An optional switch and value pair that specifies the module id associated with the design module from which the repair groups are reported. The -design_name argument must also be specified to uniquely identify the design module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example shows the reporting of repair group information from the DftSpecification for the currently elaborated design. The use of the -show_memory_instances argument provides the additional list of memory instances within each repair group.

report_memory_repair_groups -show_memory_instances

```
Repair Group Memory Instances
=============================
----------  ------------  -----------
Controller  Repair Group  Size
----------  ------------  -----------
c1          RG0           3.0kilobits
c2          RG0           512.0bits
c2          RG1           576.0bits
c3          RG0           1.0kilobits
c4          RG0           512.0bits
c5          RG0           256.0bits
c5          RG1           1.0kilobits
c6          RG0           1.0kilobits
c6          RG1           256.0bits


Repair Group Memory Instances
=============================
----------  ------------  ---------------
Controller  Repair Group  Memory Instance
----------  ------------  ---------------
c1          RG0           MEM00
c1          RG0           MEM01
c2          RG0           \mem@10
c2          RG1           mem1
c3          RG0           mem2
c3          RG0           mem3
c4          RG0           mem4
c4          RG0           mem5
c5          RG0           mem6
c5          RG1           mem7
c6          RG0           mem8
c6          RG1           mem9
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_mismatch_sources

Context: dft -edt, patterns -scan

Mode: analysis

Displays the sources of simulation mismatches.

Usage

report_mismatch_sources [*src-id* | -all] [-display [flat_schematic] [hierarchical_schematic]] [(> | >>) *file_pathname*]

Description

Displays the sources of simulation mismatches.

The report_mismatch_sources command displays detailed information about the sources of simulation mismatches found by a previous analyze_simulation_mismatches command or a write_patterns -Debug command. For each mismatch source you specify, the command reports the following information in columnar format:

- ID# — Positive integer that identifies a specific mismatch source. The tool numbers mismatch sources consecutively, starting at 1.
- Instance name — Hierarchical instance name of the mismatch source.
- Pattern number — The number of the first failing pattern caused by a mismatch source. In the Questa SIM Wave window, the pattern number is the value of the "_pattern_count" variable at the time the mismatch occurred. This variable is displayed in the top line of the Wave window and increments as each pattern is simulated. This number corresponds to the pattern number in ASCII patterns.
- Time (ns) — First simulation time where the source causes a value in the external simulator that is inconsistent with what the tool expected.
- Capture cycle — The capture cycle where the first mismatch occurs (starting from 0).
- Expected value (ATPG_val) — Expected output value from the tool.
- Simulated value (VCD_val) — Simulated output value from the external simulator. Possible values are:

  0 — Logic 0

  1 — Logic 1

  Z — Tristate

  X — Unknown

  U — Unused

  . — No VCD data was saved.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Reason — The following is a list of possible mismatch reasons:
  - “incorrect input”: Mismatch is due to wrong force values at PI.
  - “incorrect PO measure”: Mismatch is due to wrong measure PO time.
  - “incorrect loading”: Mismatch is due to incorrect scan load.
  - “incorrect capture”: Mismatch is due to incorrect capture clock even at the state element.
  - “incorrect unloading”: Mismatch is due to incorrect scan unload.
  - “wrong clock pulse”: Mismatch is due to incorrect clock pulse.
  - “setup time violation”: Mismatch is due to setup time violation, specifically the combinational path is not stable in one clock cycle.
  - “incorrect logic”: Mismatch is due to difference logic between Tessent Cell library model and the Verilog model. This includes different netlists used between ATPG tool and Verilog simulator, or Tessent Cell library model difference to the corresponding Verilog model.
  - “mismatched RAM model”: Mismatch is due to difference on RAM model between Tessent Cell library and Verilog library.
  - “incorrect library cell”: Mismatch is due to difference between Tessent Cell library model and Verilog library cell.
  - “incorrect ATPG setting”: Mismatch is due to improper setting on ATPG tool. Most common mistakes are incorrect setting of set_split_capture and set_clock_off_simulation.
  - “hold time violation”: Mismatch is due to hold time violation. The sink flop of the hold time violation is reported as the mismatch source gate.
  - “clock/data port race”: mismatch is due to clock and data racing. Common clock racing situations include C6 and C4 violation.

Arguments

- *src-id*

  An optional positive integer that specifies the identification number of the mismatch source to report. The tool assigns source IDs to mismatch sources consecutively, starting with 1.

- -display [flat_schematic] [hierarchical_schematic]

  An optional switch that specifies to display the specified source(s) in the Flat or Hierarchical Schematic window of Tessent Visualizer. If the Display switch is specified without an argument, the Flat Schematic window is displayed.

- -ALl

  An optional switch that specifies to display all the mismatch sources. This is the default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

In the Flat Schematic window, pin names assigned to the value "." indicate that the VCD debug testbench did not capture the VCD value for that location. It is not possible to capture VCD values for every node in the design due to very large file sizes and run time; the testbench only captures values in the combination cone behind the failure location, which in most cases provides sufficient information to explain the mismatch.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example assumes the pathname of the Questa SIM bin directory is defined by the PATH variable and patterns are generated but not yet saved. The example first specifies the invocation command for the external simulator, then saves the generated pattern set. Next, it creates the Verilog library *my_mismatch_work*, compiles the source for the top-level design and supporting simulation parts library into the library, and analyzes the simulation mismatches. Finally, Tessent Visualizer is invoked and displays the simulation, test pattern, and design data with the mismatch sources highlighted in red.

```
set_external_ simulator vsim -c -voptargs=+acc=npr
write_patterns results pat_p.v -verilog -parallel -replace
sys vlib my_mismatch_work
sys vlog -work my_mismatch_work my_design_source/my_design.v -v \
      my_verilog_source/verilog_lib.v
```

```
** Note: (vsim-3812) Design is being optimized...

//                              ... Tool Banner ...
//
-- Compiling module DFFS_scan
-- Compiling module my t op level module
-- Scanning library file 'my verilog source/verilog lib.v'
-- Compiling module xor3
-- Compiling module mux21
...
-- Compiling module buff
-- Compiling module mgc latch
-- Scanning library file 'my_verilog_source/verilog_lib.v'
-- Compiling UDP ud_dff
-- Compiling UDP ud_dlat

Top level modules: my_top_level_module
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_simulation_mismatches -auto -internal -lib my_mismatch_work

```
// New directory work_dft_debug has been created to store files for mismatch debug.
// Previous simulation mismatch sources are deleted.
//-----------------------------------------------------------------------------------------
//Step-1 Creating Verilog testbench work_dft_debug/mentor_default.v with failure file
//dump.
//-----------------------------------------------------------------------------------------
//Step-2 Running Verilog testbench work_dft_debug/mentor_default.v to create failure file
//-----------------------------------------------------------------------------------------
//Step-3 Loading failure file work_dft_debug/mentor_default.v.fail.
//-----------------------------------------------------------------------------------------
//Step-4 Creating Verilog testbench work_dft_debug/mentor_default.v_vcdtb.v with VCD dump
//-----------------------------------------------------------------------------------------
//Step-5 Running Verilog testbench work_dft_debug/mentor_default.v_vcdtb.v to create VCD
//file.
//Running Questa Sim to create VCD file work_dft_debug/mentor_default.v_debug.vcd
//-----------------------------------------------------------------------------------------
//Step-6 Loading VCD file work_dft_debug/mentor_default.v debug.vcd.
//-----------------------------------------------------------------------------------------
//Step-7 Analyzing 25 internal scan patterns in parallel format.
//-----------------------------------------------------------------------------------------
//ID instance (gate id) pattern-ix time(ns) Capture_cycle ATPG_val VCD_val mismatch_reason
//-- ------------------ ---------- -------- ------------- -------- ------- ----------------
// 1 /ix1286/Y (364)     scan-1       1900             0        1       0  incorrect lib
cell "and4"
// 2 /ix1278/Y (265)     scan-15     17300             0        1       0  incorrect lib
cell "and4"
```

report_mismatch_sources 2 -display flat_schematic

Tessent Visualizer opens and displays the following information about mismatches in the Transcript window. The sources are also displayed in the Flat Schematic window.

```
//ID instance (gate id) pattern-ix time(ns) Capture_cycle ATPG_val VCD_val mismatch_reason
//-- ------------------ ---------- -------- ------------- -------- ------- ----------------
// 2 /ix1278/Y (265)     scan-15     17300             0        1       0  incorrect lib
//                                                                              cell "and4"
```

For more information about the Flat Schematic window, refer to "Flat Schematic" in the *Tessent Shell User's Manual*.

Related Topics

analyze_simulation_mismatches

set_external_simulator

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_misrs

Context: pattern -scan

Mode: setup, analysis

Reports the MISRs you have added to your design.

Usage

report_misrs {**-all** | ***misr_name*** …} [> | >> *file_pathname*]

Description

Reports the MISRs you have added to your design.

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- **-all**

  A required switch that reports all MISRs.

- ***misr_name***

  A required repeatable string that specifies the name of the MISR.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

This example reports all the MISRs defined across all the EDT/logicBIST blocks.

```
report_misrs -all
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
list of MISRs
A_misr_0  MISR  length=24  in_stream_taps=20,21,23
  A_misr_0_pre_unload_value = 000000000000000000000000
A_misr_1  MISR  length=32  in_stream_taps=4,5,31
  A_misr_1_pre_unload_value = 00000000000000000000000000000000
A_misr_2  MISR  length=32  in_stream_taps=4,5,31
  A_misr_2_pre_unload_value = 00000000000100010001000100000000
A_misr_3  MISR  length=32  in_stream_taps=4,5,31
  A_misr_3_pre_unload_value = 00000000000000000000000000000000
B1_misr  MISR  length=24  in_stream_taps=20,21,23
   B1_misr_pre_unload_value = 000000000000000000000000
B2_misr_0  MISR  length=24  in_stream_taps=20,21,23
B2_misr_0_pre_unload_value = 000000000000000000000000
B2_misr_1  MISR  length=24  in_stream_taps=20,21,23
B2_misr_1_pre_unload_value = 000000000000000000000000
```

Example 2

This example reports a specific MISR: A_misr_2:

report_misrs A_misr_2

```
list of MISRs
A_misr_2  MISR  length=32  in_stream_taps=4,5,31
  A_misr_2_pre_unload_value = 00000000000100010001000100000000
```

Related Topics

add_misrs

delete_misrs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_misr_connections

Context: patterns -scan

Mode: setup, analysis

Reports MISR connections.

Usage

report_misr_connections ***misr_name*** [> | >> *file_pathname*]

Description

Reports MISR connections.

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- ***misr_name***

  A required repeatable string that specifies the name of the MISR.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

This example shows the connection between the LBIST scan chain output and the MISR input for a 24-bit MISR named 'misr'. The 16 scan chains are compacted with a 4-to-1 XOR and connected to the first 4 inputs of the MISR.

```
report_misr_connections misr
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
list of MISR connections
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[0]  MISR=misr
connection_points=1
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[1]  MISR=misr
connection_points=1
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[2]  MISR=misr
connection_points=1
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[3]  MISR=misr
connection_points=1
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[4]  MISR=misr
connection_points=2
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[5]  MISR=misr
connection_points=2
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[6]  MISR=misr
connection_points=2
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[7]  MISR=misr
connection_points=2
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[8]  MISR=misr
connection_points=3
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[9]  MISR=misr
connection_points=3
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[10]  MISR=misr
connection_points=3
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[11]  MISR=misr
connection_points=3
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[12]  MISR=misr
connection_points=4
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[13]  MISR=misr
connection_points=4
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[14]  MISR=misr
connection_points=4
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[15]  MISR=misr
connection_points=4
```

Related Topics

add_misrs

set_misr_connections

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_module_matching

Context: all contexts

Mode: all modes

Reports design modules and their matching modules of the specified format, if the set_current_design command has already been run.

Usage

report_module_matching -icl | -format { icl | tcd_bscan | tcd_scan | tcd_fusebox | tcd_memory | tcd_memory_cluster }

Description

If the current design is already described by related read commands, the tool does not do module matching processing. In case of ICL an elaboration process loads missing ICL if necessary and possible.

Arguments

- -icl

  An optional switch that specifies to report matching ICL modules.

  If the attribute “ignore_during_icl_extraction” is defined for any instance, an additional column “Ignored” is added to the report specifying whether the instance was ignored during ICL extraction. However, the “Ignored” column is not included if the value of the attribute is set to “false” for all instances for which the attribute is defined.

- -format { icl | tcd_bscan | tcd_scan | tcd_fusebox | tcd_memory | tcd_memory_cluster }

  An optional switch that specifies the format of the matching modules to be reported.

Return Values

None

Examples

Example 1

The following example shows an example report generated by this command.

```
read_icl ../data/icl/block3.icl
set_design_sources -format icl -search_design_load_path On
set_module_matching_options -suffix_pattern_list "_\[0-9\]" -regexp
set_current_design chip
report_module_matching -icl
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Design     Design                   ICL         ICL
Module     Instance                 Module      File
-----------------------------------------------------------------
PLL_CORE   chip/pll1_I1/iPLL        PLL_CORE ../data/icl_prim/PLL_CORE.icl
block1_0   chip/block1_I1           block1   ../data/icl/block1.icl
block1_1   chip/block1_I2           block1   ../data/icl/block1.icl
block2     chip/block2_I1           block2   ../data/icl/block2.icl
block3     chip/block3_I1           block3   ../data/icl/block3.icl
pll1       chip/pll1_I1             pll1     ../data/icl_prim/pll1.icl
raw1_0     chip/block1_I1/raw1_I1   raw1     ../data/icl_prim/raw1.icl
raw1_1     chip/block3_I1/raw1_I4   raw1     ../data/icl_prim/raw1.icl
raw1_2     chip/block3_I1/raw1_I3   raw1     ../data/icl_prim/raw1.icl
raw1_3     chip/block3_I1/raw1_I2   raw1     ../data/icl_prim/raw1.icl
raw1_4     chip/block3_I1/raw1_I1   raw1     ../data/icl_prim/raw1.icl
raw1_5     chip/block2_I1/raw1_I2   raw1     ../data/icl_prim/raw1.icl
raw1_6     chip/block2_I1/raw1_I1   raw1     ../data/icl_prim/raw1.icl
raw1_7     chip/block1_I2/raw1_I2   raw1     ../data/icl_prim/raw1.icl
raw1_8     chip/block1_I2/raw1_I1   raw1     ../data/icl_prim/raw1.icl
raw1_9     chip/block1_I1/raw1_I2   raw1     ../data/icl_prim/raw1.icl
sib1_0     chip/sib1_IPLL           sib1     ../data/icl_prim/sib1.icl
sib1_1     chip/block1_I2/sib1_I2   sib1     ../data/icl_prim/sib1.icl
sib1_2     chip/block1_I2/sib1_I1   sib1     ../data/icl_prim/sib1.icl
sib1_3     chip/block1_I1/sib1_I2   sib1     ../data/icl_prim/sib1.icl
sib1_4     chip/block1_I1/sib1_I1   sib1     ../data/icl_prim/sib1.icl
sib1_5     chip/sib1_I4             sib1     ../data/icl_prim/sib1.icl
sib1_6     chip/sib1_I3             sib1     ../data/icl_prim/sib1.icl
sib1_7     chip/sib1_I2             sib1     ../data/icl_prim/sib1.icl
sib1_8     chip/sib1_I1             sib1     ../data/icl_prim/sib1.icl
tap1       chip/tap1_I1             tap1     ../data/icl_prim/tap1.icl
tdr1       chip/pll1_I1/tdr1_IPLL1  tdr1     ../data/icl_prim/tdr1.icl
tdr1_1     chip/block3_I1/tdr1_I4   tdr1     ../data/icl_prim/tdr1.icl
tdr1_2     chip/block3_I1/tdr1_I3   tdr1     ../data/icl_prim/tdr1.icl
tdr1_3     chip/block3_I1/tdr1_I2   tdr1     ../data/icl_prim/tdr1.icl
tdr1_4     chip/block3_I1/tdr1_I1   tdr1     ../data/icl_prim/tdr1.icl
tdr1_5     chip/block1_I2/tdr1_I2   tdr1     ../data/icl_prim/tdr1.icl
tdr1_6     chip/block1_I2/tdr1_I1   tdr1     ../data/icl_prim/tdr1.icl
tdr1_7     chip/block1_I1/tdr1_I2   tdr1     ../data/icl_prim/tdr1.icl
tdr1_8     chip/block1_I1/tdr1_I1   tdr1     ../data/icl_prim/tdr1.icl
tdr2       chip/block2_I1/tdr2_I1   tdr2     ../data/icl_prim/tdr2.icl
```

Example 2

In the following example, the “ignore_during_icl_extraction” attribute is defined for three instances which are identified in the “Ignored” column in the report generated by the report_module_matching command.

```
set_attribute_value chip/sib1_I3 -name ignore_during_icl_extraction -value false
set_attribute_value chip/sib1_I2 -name ignore_during_icl_extraction -value true
set_attribute_value chip/sib1_I1 -name ignore_during_icl_extraction -value true
report_module_matching -icl
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following is a snippet of the generated report:

```
Design      Design         Ignored    ICL        ICL
Module      Instance                  Module     File
-----------------------------------------------------------------------

sib1_5      chip/sib1_I4   No         sib1      ../data/icl_prim/sib1.icl
sib1_6      chip/sib1_I3   No         sib1      ../data/icl_prim/sib1.icl
sib1_7      chip/sib1_I2   Yes        sib1      ../data/icl_prim/sib1.icl
sib1_8      chip/sib1_I1   Yes        sib1      ../data/icl_prim/sib1.icl
```

Example 3

The following example shows an example report of matching tcd_memory_cluster modules.

report_module_matching -format tcd_memory_cluster

The following is a snippet of the generated report:

```
Design  Design                                Module     File
Module  Instance
-------------------------------------------------------------------------------
A2Mem1  A1Inst1/A2InstMem1                    A2Mem1  ../data/A2Mem1.tcd_mem_lib_cluster
A2Mem1  A1Inst1/A2InstMem2                    A2Mem1  ../data/A2Mem1.tcd_mem_lib_cluster
B4Mem1  B1Inst1/B2Inst1/B4InstMem1            B4Mem1  ../data/B4Mem1.tcd_mem_lib_cluster
B4Mem1  B1Inst1/B2Inst1/B3Inst1/B4InstMem1    B4Mem1  ../data/B4Mem1.tcd_mem_lib_cluster
B4Mem2  B1Inst1/B2Inst1/B3Inst1/B4InstMem2    B4Mem2  ../data/B4Mem2.tcd_mem_lib_cluster
B4Mem4  B1Inst1/B2Inst1/B3Inst1/B4InstMem4    B4Mem4  ../data/B4Mem4.tcd_mem_lib_cluster
```

Example 4

The following example shows the reporting if the tool does not do ICL module matching and only performs ICL elaboration, due to already loaded ICL for the current design. Introspection commands are used to get the information about the ICL elaboration.

```
read_verilog top.v
read_design top -icl_only -design_id rtl
read_icl .../tsdb_outdir/dft_inserted_designs/top_rtl.dft_inserted_design/top.icl
set_current_design top
report_module_matching -icl
```

```
// No ICL module matching data available.
```

```
foreach_in_collection icl_mod [get_icl_modules] { \
puts "icl module [get_name_list $icl_mod]" \
puts "from icl file [get_attribute_value_list $icl_mod -name file_path_name]" \
puts "design modules [get_name_list [get_modules -of_icl_modules $icl_mod -silent]]" \
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
icl module top
from icl file .../tsdb_outdir/dft_inserted_designs/top_rtl.dft_inserted_design/top.icl
design modules top
icl module top_rtl_tessent_lbist
from icl file .../tsdb_outdir/dft_inserted_designs/top_rtl.dft_inserted_design/top.icl
design modules top_rtl_tessent_lbist
icl module top_rtl_tessent_edt_lbist_c10
from icl file .../tsdb_outdir/dft_inserted_designs/top_rtl.dft_inserted_design/top.icl
design modules top_rtl_tessent_edt_lbist_c10
icl module top_rtl_tessent_occ
from icl file .../tsdb_outdir/dft_inserted_designs/top_rtl.dft_inserted_design/top.icl
design modules top_rtl_tessent_occ
icl module top_rtl_tessent_sib_2
from icl file .../tsdb_outdir/dft_inserted_designs/top_rtl.dft_inserted_design/top.icl
design modules
icl module top_rtl_tessent_sib_1
from icl file .../tsdb_outdir/dft_inserted_designs/top_rtl.dft_inserted_design/top.icl
design modules top_rtl_tessent_sib_1
...
```

Related Topics

report_module_matching_options

set_current_design

set_design_sources

set_module_matching_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_module_matching_options

Context: all contexts

Mode: all modes

Reports the current settings defined by the set_module_matching_options command.

Usage

report_module_matching_options

Description

Reports the current settings defined by the set_module_matching_options command.

Arguments

None

Return Values

None

Examples

Example 1

The following example shows an example report generated by this command.

```
SETUP> set_design_sources -format icl -search_design_load_path On
SETUP> set_module_matching_options -suffix_pattern_list "_mod_\[0-9\]" -regexp
SETUP> set_module_matching_options -prefix_pattern_list mgc -append
SETUP> set_module_matching_options -suffix_pattern_list "_\[0-9\]" -regexp -append
SETUP> report_module_matching_options
```

```
//  Option                        Value
//  ------------------------      -----
//  case insensitivity override   off
//
//  affix type  regexp  pattern
//  ----------  ------  ------------
//  prefix      no      "mgc"
//  suffix      yes     "_mod_[0-9]"
//  suffix      yes     "_[0-9]"
```

Example 2

This example shows the output of the report_module_matching_options command when you have excluded modules using the “set_module_matching_options -excluded_modules_from_module_matching” command as follows:

```
SETUP> set_module_matching_options -format icl \
         -excluded_modules_from_module_matching top_ijtag_sib

SETUP> report_module_matching_options
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Option                        Value
// ---------------------------   -----
// case insensitivity override   off
//
// affix type  regexp  pattern
// ----------  ------  --------------------------------
// prefix      yes     ".*"
// suffix      yes     "(_7nm.*|_16nm.*|(_[0-9a-f]+)*)"
//
// Format  Module to exclude
// ------  ----------------------------------------------------------------
// ICL     top_ijtag_sib
```

Related Topics

set_module_matching_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_multicycle_paths

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays multicycle path definitions you previously read into the tool using the read_sdc command.

Usage

report_multicycle_paths [-All] [-Verbose] [-VIolations [-EXclude invalid_design_objects]]
[-From *source_node…* | -FROM_Clock *clock_signal_name…*]
[-TO *sink_node…* | -TO_Clock *clock_signal_name…*]
[{-THrough *through_node…*}…]
[{> | >>} *file_pathname*]

Description

Displays multicycle path definitions you previously read into the tool using the read_sdc command.

Note that you can display SDC filenames and line numbers by first issuing the "set_timing_exceptions_handling -line_no" command. To analyze if a pin or a gate is affected by a multicycle path, you can use "set_gate_report -timing_exceptions on" and report_gates commands.

If a specified starting point, end point, source node, clock signal name, sink node, or through node does not translate to an internal gate, zero (0) multicycle paths are reported.

Arguments

Tip

When using the following arguments, you can include one or more asterisks (*) in pin or instance pathnames. The command treats an asterisk as a wildcard character, enabling you to use it to match many pathnames in the design.

- -All

  An optional switch that displays all previously-defined multicycle paths. This is the default.

- -Verbose

  An optional switch that displays all the gate information in the from, to, and through gate lists.

- -VIolations

  An optional switch that displays only the false paths that have violations, along with their respective violation messages.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -EXclude invalid_design_objects

  An optional switch and keyword pair to exclude the display of violation paths due to invalid objects.

- -From *source_node…*

  An optional switch and repeatable string pair that specifies the starting point of a multicycle path. A valid starting point is a pin pathname, an instance pathname, a clock primary input (PI), or an internal clock pin name. For an instance pathname, all the output pins of the instance are considered to be starting points. For a clock PI, the outputs of all the non-transparent latches, flip-flops, and RAMs associated with the clock PI are considered to be starting points. A pin pathname is considered to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and the internal clock pin is handled the same as a clock PI.

- -FROM_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the starting point of a multicycle path by its associated clock pin pathname. Any pin pathname you specify with this argument is interpreted to be the source of a clock signal, and the outputs of all non-transparent latches, flip-flops, and RAMs associated with this clock signal are considered to be starting points.

- -TO *sink_node…*

  An optional switch and repeatable string pair that specifies the end point of a multicycle path. A valid end point is a pin pathname, an instance pathname, a clock primary input (PI), or an internal clock pin name. For an instance pathname, all the input pins of the instance are considered to be end points. For a clock PI, the data inputs of all non-transparent latches, flip-flops, and RAMs associated with the clock PI are considered to be end points. A pin pathname is considered to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and the internal clock pin is handled the same as a clock PI.

  Note

  For a 2-port latch with ports: Port1 = (D1, CLK1), Port2 = (D2, CLK2), and primary input clocks CLK1 and CLK2, the command “report_multicycle_paths... -to CLK2” would report multicycle paths with an end point at D2.

- -TO_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the end point of a multicycle paths by its associated clock pin pathname. Any pin pathname you specify with this argument is interpreted to be the source of an internal clock signal and the data inputs of all non-transparent latches, flip-flops, and RAMs associated with this clock are considered to be end points.

- {-THrough *through_node…*}…

  An optional, repeatable switch and repeatable string pair that specifies circuit node(s) through which the multicycle path must pass. A valid through node is a pin or instance

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

pathname. Clock PIs are not supported as through nodes and no attempt is made to determine if a pin pathname is that of an internal clock pin.

When you use multiple -Through arguments, their order left-to-right in the command string is considered to be the order in which a signal would pass through the nodes on the specified path. For example, the following multicycle path specification:

```
report multicycle paths -from A -through B C -through D
E -to F
```

specifies that all paths that start from A pass through either B or C, then pass through D or E, and end at F. If you do not use the -Through argument when specifying a path, all paths from the specified starting points to the specified end points are considered to be multicycle paths.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

Example 1

The following example displays all previously-defined multicycle paths.

report_multicycle_paths -all

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Multicycle Path -from -from /cpu/d_in[7] -through /cpu/u1 -cycles 2
// Multicycle Path -to /cpu/c -cycles 2
//
// ---------------------------------------------------------------
// The multicycle path below has an error.
// ---------------------------------------------------------------
// Multicycle Path -from name1 -to name2 -cycles 2
//
// ---------------------------------------------------------------
// The multicycle paths below have warnings.
// ---------------------------------------------------------------
// Multicycle Path -from /cpu/d_in[7] -through /cpu/d_in_[6]
//    /cpu/u25 -cycles 2
// Multicycle Path -through I1 I2 I3 I4 I5 SIN CLK1 SEN -cycles 3
// Multicycle Path -from CLK1 -through /cpu/xray[6] CLK1 -cycles 2
//
// Total reported multicycle paths = 6
// Warning: The last 4 paths reported have 1 error and 3 warnings.
//          Use the command "report_multicycle_paths -debug_error" to
//          report the causes of the errors/warnings.
// Warning: Multicycle path defined through all primary inputs, including
//          all clocks, which prohibits at-speed testing. Deleted clocks
//          from this list.
// Warning: There exists a multicycle path that prohibits at-speed testing
//          of clock domain /CLK1.
//          This may result in lower test coverage.
```

Example 3

The following example displays a verbose report of multicycle paths that start from /cpu/d_in[7]. Notice the empty “( )” for the incorrectly defined path.

report_multicycle_paths -verbose -from /cpu/d_in[7]

```
// Multicycle Path -from /cpu/d_in[7] (/reg_d_1_7_/Q (81)) -through
//      /cpu/u1 (/reg_d_1_6_/Q (73)) -cycles 2
//
// ----------------------------------------
// The multicycle path below has an error.
// ----------------------------------------
// Error: Multicycle Path -from /cpu/d_in[7] ( ) -through /cpu/xray[6] ( )
//        -cycles 2
//
// Total reported false paths = 2
```

For examples of how to use the asterisk (*) character to specify multiple pathnames with one pathname string, see the add_false_paths command.

Example 4

The following shows multicycle path reporting after turning on reporting of SDC filenames and line numbers:

read_sdc sdc2
set_timing_exceptions_handling -line_no on
report_multicycle_paths

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Multicycle Path -from sff108/Q -to sff209/D -cycles 3 -setup
//        (filename = sdc2, line 1)
```

Related Topics

add_false_paths

delete_false_paths

delete_multicycle_paths

read_sdc

report_false_paths

set_timing_exceptions_handling

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_multiprocessing_options

Context: unspecified, dft -edt, patterns -scan

Mode: setup, analysis, insertion

Displays the values of all variables used when running multiprocessing commands for ATPG or fault simulation.

Usage

report_multiprocessing_options

Description

The report_multiprocessing_options command displays, in human-readable format, the values of multiprocessing variables that affect the behavior of the tool's built-in multiprocessing manager. These are values that you can set with the set_multiprocessing_options command. To obtain the value of a multiprocessing processing variable for Tcl scripting, use the get_multiprocessing_option command.

For more information, refer to "Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual* and "The Tessent Tcl Interface" in the *Tessent Shell User's Manual*.

Arguments

None

Examples

The following example shows how to display the current settings of the multiprocessing variables:

```
report_multiprocessing_options
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Multiprocessing options:
//  Option                        Type    Value      Description
//  ---------------------------   ------  ---------  ---------------------------------------
//  generic_delete                string             generic job scheduler job deletion
//                                                   command
//  generic_scheduler             string             generic job scheduler job submission
//                                                   command
//  license_timeout               string  5          positive integer specifying in minutes
//                                                   how long to queue for a license,
//                                                   'unlimited' or 'no_queue'
//  lsf_heuristics                on/off  on         true for LSF job scheduler heuristic
//                                                   learning
//  lsf_learning                  on/off  on         true for LSF job scheduler learning
//  lsf_options                   string             options for LSF job scheduler
//  multithreading                on/off  on         turn on/off multithreading flow
//  processors_per_grid_request   number  -1         # processors grouped for each grid
//                                                   request (default: -1 implies 8 if the
//                                                   tool knows how to request more than one)
//  remote_shell                  string  rsh        rsh or ssh remote_shell setting
//  result_time_limit             float   45         time limit (min) used to detect
//                                                   non-responsive secondary processes.
//  scheduler_timeout             string  10         positive integer specifying in minutes
//                                                   how long to wait for secondary
//                                                   processors, or 'unlimited'
//  sge_options                   string             options for SGE job scheduler
```

Related Topics

add_processors

delete_processors

get_multiprocessing_option

report_processors

set_multiprocessing_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_net_polygons

Context: patterns -scan_diagnosis

Mode: analysis

Returns the x, y coordinate location of the specified net in the LDB.

Usage

report_net_polygons ***net_path*** [-ldb_units]

Description

This command returns the x and y coordinate location corresponding to a specified LEF/DEF net path in the LDB. The tool returns chip-level coordinates.

By default, the command returns coordinates expressed in microns units. You can override the default by specifying the -ldb_units switch.

Arguments

- ***net_path***

  Required string that specifies the LEF/DEF net path within the LDB.

- -ldb_units

  An optional switch that returns the x, y coordinates in LDB units rather than the default micron units.

Related Topics

get_instance_polygons

get_xy_instance

report_instance_polygons

report_xy_instance

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_nofaults

Context: dft -edt, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays the nofault settings for the specified pin pathnames or pin names of instances.

Usage

report_nofaults ***pathname***... | **-All** [-Instance] [-Stuck_at {01 | 0 | 1}] [-Class {Full | User | System}] [{> | >>} *file_pathname*]

Description

The report_nofaults command displays, for pin pathnames or pin names of instances, the nofault settings that you previously specified with the add_nofaults command.

Arguments

- ***pathname***

  A repeatable string that specifies the pin pathnames or the instance pathnames for which you want to display the nofault settings. If you specify an instance pathname, you must also specify the -Instance switch.

- **-All**

  A switch that displays the nofault settings on either all pin pathnames or, if you also specify the -Instance switch, all pin names of instances.

- -Instance

  An optional switch that specifies that the *pathname* or -All argument indicates instance pathnames.

- -Stuck_at {01 | 0 | 1}

  An optional switch and literal pair that specifies the stuck-at or transition nofault settings you want to display. The choices are as follows:

  01 — A literal that specifies to display both the “stuck-at-0” and “stuck-at-1” nofault settings for stuck-at faults; or both the “slow-to-rise” and “slow-to-fall” nofault settings for transition faults. This is the default.

  0 — A literal that specifies to display only the “stuck-at-0” nofault settings (“slow-to-rise” nofault settings for transition faults).

  1 — A literal that specifies to display only the “stuck-at-1” nofault settings (“slow-to-fall” nofault settings for transition faults).

- -Class {Full | User | System}

  An optional switch and literal pair that specifies the source (or class) of the nofault settings which you want to display. The literal choices are as follows:

  Full — A literal that displays all the nofault settings in the user and system class. This is the default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

User — A literal that displays only the user-entered nofault settings.

System — A literal that displays only the netlist-described nofault settings.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example displays all pin names of the instances that have the nofault settings:

```
add_nofaults i_1006 i_1007 i_1008 -instance
report_nofaults
```

Related Topics

add_faults

add_synchronous_clock_group

delete_faults

delete_nofaults

report_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_nonscan_cells

Context: dft -edt, patterns -scan

Mode: analysis

Displays the non-scan cells whose model type you specify.

Usage

report_nonscan_cells [-All | -TIE0 | -TIE1 | -TIEX | -TLatch | -Clock_sequential | -INIT0 | -INIT1] [-VERilog] [{> | >>} *file_pathname*]

Description

The report_nonscan_cells command displays the instance name, gate number, and model type of the non-scan cells that you specify. The tool assigns a model type to the non-scan cells during the Design Rules Check (DRC) in order to model non-scan memory behavior more exactly.

During scan loading, scan clocks can affect non-scan memory elements. The worst case is to assume that all non-scan memory elements have an unknown state right after scan loading, which makes some faults difficult to test. Therefore, the tool assigns an appropriate model type to the non-scan cells, which sets them to known values. The following argument descriptions describe the conditions that the tool uses to make the various model type assignments.

Note

The tool displays a clock_stability_flag for non-scan cells modeled as TIEX cells. If "clock_stability_flag=1", the non-scan cell is stable in the clock_off frame when all defined Primary Input (PI) clocks are off. The ATPG engine can test these non-scan cells deterministically. For more information about the clock_off frame, see "Event Simulation for DFFs and Latches" in the Tessent Scan and ATPG User's Manual.

If "clock_stability_flag=0", the clock port of the non-scan cell is unstable (it is X when all defined PI clocks are off). The tool treats these unstable non-scan cells as TIEX during ATPG, and the associated faults can only be detected through simulation.

Arguments

- -All

  An optional switch that displays all non-scan cells. This is the default.

- -TIE0

  An optional switch that displays the non-scan cells which are always 0 after each loading and before the next unloading.

  Non-scan cells that the tool models as TIE0 indicate that the pin constraints hold the cell's value during non-scan operation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -TIE1

  An optional switch that displays the non-scan cells which are always 1 after each loading and before the next unloading.

  Non-scan cells that the tool models as TIE1 indicate that the pin constraints hold the cell's value during non-scan operation.

- -TIEX

  An optional switch that displays the non-scan cells which are always unknown (X) after each loading and before the next unloading.

- -TLatch

  An optional switch that displays the non-scan cells which the tool models as transparent latches.

- -Clock_sequential

  An optional switch that displays all clock_sequential cells that are valid when the sequential depth is set to non-zero.

- -INIT0

  An optional switch that displays a list of cells which DRC has identified as being initialized to 0 by scan load but has failed to prove that it is tied to 0.

  Non-scan cells that the tool models as INIT0 indicate one of the following:

  - The test procedure can set the memory element to 0 after scan loading.
  - The non-off clock is a reset.

- -INIT1

  An optional switch that displays a list of cells which DRC has identified as being initialized to 1 by scan load, but has failed to prove that it is tied to 1.

  Non-scan cells that the tool models as INIT1 indicate one of the following:

  - The test procedure can set the memory element to 1 after scan loading.
  - The non-off clock is a set.

- -VERilog

  An optional switch that outputs the cell instance and pin pathnames in Verilog syntax, rather than using the default netlist independent format.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example reports only the non-scan cells that the tool models as transparent latches.

add_scan_groups g1 proc.g1
add_scan_chains c1 g1 scin scout
add_clocks 0 clk
set_system_mode analysis
report_nonscan_cells -tlatch

Example 2

The following example reports various non-scan cells in a design.

ANALYSIS> report_nonscan_cells -tie0

```
DFF '/inst/retiming_reg/' (61316) modeled as TIE0, clock_stability_flag=0
DFF '/inst/\tdr_reg[0] /' (61315) modeled as TIE0, clock_stability_flag=0
LA '/clock/latch_reg/' (61238) modeled as TIE0, clock_stability_flag=0
LA '/inst/retiming_reg/Q' (61214) modeled as TIE0, clock_stability_flag=0
...
```

ANALYSIS> report_nonscan_cells -tie1

```
LA '/clock/latch_reg/' (61239) modeled as TIE1, clock_stability_flag=0
DFF '/inst/ijtag_data/' (61234) modeled as TIE1, clock_stability_flag=0
LA '/inst/retiming_reg/Q' (61230) modeled as TIE1, clock_stability_flag=0
...
```

ANALYSIS> report_nonscan_cells -tiex

```
DFF '/inst/reg[0] /' (61313) modeled as TIEX, clock_stability_flag=1
DFF '/inst/reg[1] /' (61312) modeled as TIEX, clock_stability_flag=1
LA '/inst/retiming_reg/Q' (61236) modeled as TIEX, clock_stability_flag=0
...
```

ANALYSIS> report_nonscan_cells -tlatch

```
LA '/U_2/c22_i/Q' (58534) modeled as TLA, clock_stability_flag=0
LA '/U_1/c14_i/Q' (58010) modeled as TLA, clock_stability_flag=0
LA '/U_0/c6_i/Q' (57481) modeled as TLA, clock_stability_flag=0
...
```

ANALYSIS> report_nonscan_cells -clock_sequential

```
DFF '/inst/SLOW_CLK_reg/' (61189) modeled as TIE1, clock_stability_flag=1
DFF '/inst/SLOW_CLK_reg/' (61177) modeled as TIE1, clock_stability_flag=1
...
```

ANALYSIS> report_nonscan_cells -verilog

```
DFF 'inst.c1_i.\reg[0] .' (61313) modeled as TIEX, clock_stability_flag=1
DFF 'inst.c1_i.\reg[1] .' (61312) modeled as TIEX, clock_stability_flag=1
DFF 'inst.c1_i.out_reg.' (61275) modeled as TIEX, clock_stability_flag=1
DFF 'inst.c1_i.pipe_reg.' (61275) modeled as TIEX, clock_stability_flag=1
...
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_nonscan_models

Context: dft -scan, dft -test_points

Mode: setup, analysis

Displays the sequential non-scan model list.

Usage

report_nonscan_models [-Class { User | System}]

Description

Displays sequential models that you added by using the add_nonscan_instances command.

Arguments

-Class {User | System}

An optional switch and literal pair that specifies the class code of the non-scan model that you specify. The valid literal names are as follows:

User — A literal that specifies that the list of non-scan models were previously added by using the add_nonscan_instances command. This is the default class.

System — A literal that specifies that the list of non-scan models were added by the tool.

Examples

The following example displays all sequential non-scan models from the non-scan model list:

```
set_system_mode analysis
add_nonscan_instances -modules d_flip_flop1 d_flip_flop2
report_nonscan_models
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_notest_points

Context: dft -scan, dft -test_points

Mode: setup, analysis

Displays all the circuit points for which you do not want the tool to insert controllability and observability.

Usage

report_notest_points [-Paths] [-tool_identified]

Description

The report_notest_points command displays the circuit points added using the add_output_masks command and which therefore, the tool cannot use for testability insertion.

You can also list the critical path definitions that added notest points by using the -Paths switch.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of file_pathname.

- -Paths **(dft -test_points context only)**

  An optional switch that displays the definitions for all currently loaded critical paths that have marked notest points. If this switch is not specified, the tool reports only the resulting notest points.

- -tool_identified

  An optional switch that displays only the notest points added by the tool.

  Note that if you issue the report_notest_points command without any switch, it displays the notest points added by the user.

Examples

The following example displays the list of all circuit points that the tool cannot use for testability insertion:

```
add_notest_points i_1006/o i_1007/o
add_notest_points i_1009/o
report_notest_points
```

Related Topics

add_output_masks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_notest_points

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_output_masks

Context: dft, dft -edt, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays a list of the currently masked primary output pins.

Usage

report_output_masks [{> | >>} *file_pathname*]

Description

The report_output_masks command displays the masked primary output pins using the add_output_masks command. When you mask a primary output pin, the tool marks that pin as an invalid observation point during the fault detection process. The tool uses all unmasked primary output pins as possible observation points to which the effects of all faults propagate for detection.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following example masks two primary outputs and then displays the results:

add_output_masks qb1 qb2
report_output_masks

```
TIEX qb1
TIEX qb2
```

Related Topics

add_output_masks

delete_output_masks

set_output_masks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_pade_logical_connections

Context: dft

Mode: setup, analysis, insertion

Reports the logical paths that exist within the current design between the specified source and destination pins/ports, as well as all connections from or to the specified pins/ports. Only logical connections created by the add_pade_logical_connection command are reported by this command.

Usage

report_pade_logical_connections [-from *source_pin_or_port_name*]
[-to *destination_pin_or_port_name*]

Description

This command reports the logical paths that exist within the current design between the specified source and destination pins/ports and were added by the add_pade_logical_connection command, as well as all connections from or to the specified pins/ports. If no pin or port connections are specified, all logical connections are listed.

Arguments

- -from *source_pin_or_port_name*

  An optional switch and value pair that specifies the source pin/port for the logical connections, which should be a top-level module port or a hierarchical instance pin in the current top design module.

- -to *destination_pin_or_port_name*

  An optional switch and value pair that specifies the destination pin/port for the logical connections, which should be a top-level module port or a hierarchical instance pin in the current top design module.

Examples

Example 1

The following example lists the logical connections between the pins /i1/in and /i32/u11/x10, as well as all logical connections from /i1/in and all connections to /i32/u11/x10.

report_pade_logical_connections -from /i1/in -to /i32/u11/x10

```
//  PADE Logical Connections:
//  From Source  To Destination   Inversion
//  ===========  ==============   =========
//  /i1/in       /i32/u11/x10     no
```

Example 2

The following example lists all the logical connections in the current design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add_pade_logical_connection -from tap1_I1/ue -to block3_I1/ue -inverted

add_pade_logical_connection -from tap1_I1/ue -to logical_out -inverted

add_pade_logical_connection -from tdi -to tdo -inverted

add_pade_logical_connection -from tdi -to tms

add_pade_logical_connection -from tdo -to tms -inverted

report_pade_logical_connections

```
//  PADE Logical Connections:
//  From Source  To Destination  Inversion
//  ===========  ==============  =========
//  /tap1_I1/ue  /block3_I1/ue   yes
//  /tap1_I1/ue  logical_out     yes
//  tdi          tdo             yes
//  tdi          tms             no
//  tdo          tms             yes
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_pattern_filtering

Context: dft -edt, patterns -scan

Mode: analysis

Displays the current pattern filtering configuration.

Usage

report_pattern_filtering [{> | >>} *file_pathname*]

Description

The report_pattern_filtering command displays the pattern filtering configuration set previously with the set_pattern_filtering command.

Note

The report_environment command reports if pattern filtering is on, but shows no status for pattern filtering when it is disabled.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following example displays the current pattern filtering configuration:

report_pattern_filtering

```
   Current pattern filtering configuration
-----------------------------------------------
   original_patterns (internal)  = 125
   current_filtered_pattern(s)   = 7
   sampled_patterns_per_type     = 2
   sample_capture_clock          = on
   sample_pattern_type(s)        = all
-----------------------------------------------
```

Related Topics

report_environment

set_pattern_filtering

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_pattern_sets

Context: patterns -ijtag

Mode: analysis

Creates a report of a specified pattern set or of all pattern sets.

Usage

report_pattern_sets [-sort_by {declaration_order | name}] [*pattern_set_list*]

Description

This command creates a human-readable report of all pattern sets when the pattern_set_list argument is omitted. The report contains the names of the pattern sets, the associated timeplates, the number of vectors, the TAP start and TAP end states, and whether the pattern set has been saved.

Arguments

- -sort_by {declaration_order | name}

  An optional switch and keyword pair that sorts the patterns in the report by order of declaration or by name. By default, the command sorts the patterns in the order of declaration. If you do not specify this switch, but do specify a pattern_set_list, the command reports pattern sets in the order specified by the list.

- *pattern_set_list*

  An optional Tcl list of pattern set names. If this list is present, the report is restricted to the pattern sets in the list. Otherwise, the report contains all pattern sets.

Return Values

None

Examples

Example 1

The following example creates a report of all pattern sets sorted by name:

```
report_pattern_sets -sort_by name
```

Example 2

The following example creates a report of the pattern sets returned by the lsearch command:

```
report_pattern_sets [lsearch -inline -all [get_pattern_set_list] p*]
```

For a detailed example, refer to "Create Pattern Sets" in the *Tessent IJTAG User's Manual*.

Related Topics

close_pattern_set

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_open_pattern_set

get_pattern_set_list

open_pattern_set

reset_open_pattern_set

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_pattern_trigger

Context: patterns -silicon_insight

Mode: all modes

Reports pattern triggers set with the add_pattern_trigger command.

Usage

report_pattern_triggers [-pattern_name *pattern_name*]

Arguments

- -pattern_name *pattern_name*

  An optional switch and pattern name that reports the triggers only for the specified pattern. If you do not specify this, the report includes all pattern triggers.

Examples

report_pattern_triggers -pattern_name mini_chain_0_setup.stil

```
Trigger vector address(es) for pattern 'mini_chain_0_setup.stil' = '11 25 5'.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_patterns

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: analysis

Displays information about patterns.

Usage

report_patterns [-EXTernal] [-PATtern_index *pattern_index*]
[-PATH_faults] [-CHain_test]
[-UNLoad_x]
[-INClude_always_pulsed_clocks]
[{> | >>} *file_pathname*]

Additional switches available only when EDT is on:

report_patterns
[-EDT [[-COnstant_chains] [-Verbose]]

Description

You can display information for all patterns or use the -Pattern_index switch to report on a single pattern. When you use the set_pattern_filtering command to create a temporary subset of sampled patterns, the report_patterns command reports only on patterns in the filtered subset. By default, information for internal patterns is reported, but you can report on patterns in an external pattern set by first specifying the external pattern file with the read_patterns command.

Also, by default, the report information shows pulse_always and pulse-in-capture clocks as an *. In that case, a note follows the report to identify the clock information.

The following information is reported for each pattern when the -EDT switch is not used:

- Pattern # — Pattern index
- Pre-filtering pattern # — Pattern index in the original unfiltered pattern set (reported only when filtering by the set_pattern_filtering command is in effect, causing “Pattern #” to show pattern indexes in the filtered subset).
- Pattern type — Basic, Clock sequential, MacroTest, Multiple load, or Iddq
- Cycles — An integer indicating how many capture cycles there are in the pattern
- Loads — An integer indicating how many scan loads there are in the pattern
- Observe procedure — Name of the observe procedure (Master_Observe or Shadow_Observe), if applied in the pattern
- Capture procedure — Name of the named capture procedure, if applied in the pattern
- Capture clock sequence — Sequence the clocks are pulsed in during capture

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When you use the -EDT switch to report a pattern created with Tessent TestKompress (EDT On), the information content of the report changes to display the following EDT-specific information:

- EDT block name if applicable
- EDT channel index number
- EDT channel names
- EDT input channel pin names
- EDT output channel pin names
- Observed core scan chain name or names

If a channel is masked in the reported pattern, the name of the single scan chain observed for that channel is listed. If the channel is not masked in the pattern, this means that all connected chains are observed and the "<all connected chains>" string displays.

Arguments

- -EXTernal

  An optional switch that displays pattern information for external patterns instead of internal patterns. Before using this switch, you must specify the name of the external pattern file with the read_patterns command.

- -PATtern_index *pattern_index*

  An optional switch and integer pair that displays information only for the pattern with the specified index, not all patterns.

- -PATH_faults

  An optional switch that displays the path faults detected by the internal patterns. This switch cannot be used with the -external switch and is only valid when the current fault type is set to path_delay.

- -CHain_test

  An optional switch that, together with -pattern_index switch, reports a particular chain test pattern in the pattern set. Without this switch, the tool reports scan test pattern information. Note that you can use this switch only when chain test patterns exist. Chain test patterns are created by the write_patterns command or if loaded as part of external patterns. If no chain test patterns exist, the tool issues an error message.

- -EDT **(Available only when EDT is on)**

  An optional switch that (1) reports which of the internal scan chains connected to each scan channel are observed by the specified EDT pattern when used in conjunction with the -Pattern_index switch, or (2) reports one line per pattern when used without the -Pattern_index switch. On each line, the pattern ID is followed by the pattern masking status

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

for this pattern. This switch is valid only for Tessent TestKompress patterns created with Tessent TestKompress (EDT On).

- -Constant_chains **(Available only when EDT is on)**

  An optional switch that, together with the -EDT and -Pattern_index switches, lists the actual data source (decompressor or constant 0) for the specified internal pattern. Use this switch to display test pattern information for the low-power decompressor.

- -Verbose **(Available only when EDT is on)**

  An optional switch that, together with the -EDT and -Pattern_index switches, lists the actual names of all the connected chains in the observed_chains column. Without this switch, the summary message "<all connected chains>" is displayed rather than the names of all the connected chains.

- -UNLoad_x

  An optional switch that reports patterns that contain some scan cells with X unload values. For EDT patterns, X values due to EDT masking are not considered as unload X. When you specify this switch with the -pattern_index switch, the command also reports the scan cells that capture X values (see "Example 8" on page 2032).

  You can use the -unload_x switch for debugging purposes if you issue a "set_simulation_option -report_x_capturing_cells on" command and notice some patterns that capture X on scan cells. You should first use the -unload_x switch to see patterns that unload X, and then add the -pattern_index switch to further see which scan cells capture X of the specified pattern.

  Use the -x_cell_report_limit switch to control the number of cells that get reported.

- -INClude_always_pulsed_clock

  An optional switch that enables the report to include the full names of the pulse-always and pulse-in-capture clocks. This changes the behavior from the default, which is to represent these clocks as "*" in the report and follow the report with a note to identify the clocks.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

Example 1a

The following example of scan pattern reporting displays all patterns in the current internal pattern set. Table 7-2 summarizes the display notation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 7-2. Notation Used in report_patterns Display

Notation	Column	Meaning
master_obs	observe_proc	Pattern applies a master_observe procedure.
shadow_obs	observe_proc	Pattern applies a shadow_observe procedure.
-	observe_proc	No master_observe or shadow_observe procedure is applied.
-	capture_proc	No named capture procedure is applied.
-	capture_clock_sequence	No clock is pulsed in the capture window.
[-]	capture_clock_sequence	No clock is pulsed in the capture window, but primary inputs are being forced.
[*clock1*][*clock2*] …	capture_clock_sequence	Sequence (left to right) in which *clock1* and *clock2*, etc. are pulsed.
[*clock1*, *]	capture_clock_sequence	By default, pulse-always and pulse-in-capture clocks are shown as *. A note is added at the end of the report to identify these clocks.
[*clock1*, *clock2*, …]	capture_clock_sequence	Shows *clock1* and *clock2*, etc. are pulsed at the same time.
S	capture_clock_sequence	A scan load occurs in the corresponding cycle.[1]

1. The first cycle always has a scan load, so no S is displayed for the first cycle.

report_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
patt #  type            cycles loads observe_proc cap_proc    cap_clk_seq
------- --------------- ------ ----- ------------ ----------- -----------
   0    basic             1      1   master_obs     -         [clk1]
   1    basic             1      1     -            -         [clk1]
   2    basic             1      1     -            -         [clk1]
   3    clock_seq         2      1   shadow_obs   named_cap1  [clk1]
                                                               [clk1]
   4    multiple_load     3      3     -            -         [ramclk]
                                                              S[clk1]
                                                              S[clk2]
   5    clock_seq         2      1     -            -         [clk1,clk2]
                                                              [-]
   6    clock_seq         2      1     -            -         [clk1][-]
   7    multiple_load     3      2     -            -         [clk1]
                                                              S[clk1]
                                                              [clk1]
   8    basic             1      1     -            -         [clk1]
   9    basic             1      1     -            -         [-]
  10    basic             1      1     -            -         [scn_rst]
```

Pattern 0 is a basic pattern that applies a master_observe procedure. Pattern 3 is a clock sequential pattern that uses the named capture procedure “named_cap1” in which the clock sequence is [clk1] [clk1]. Pattern 4 is a multiple load pattern. The S in the capture clock sequence indicates there is a scan load in the corresponding cycle (the first cycle always has a scan load, so no S is displayed there). Pattern 5 is a clock sequential pattern that pulses two clocks simultaneously in the first capture cycle, then in the cycle that follows, forces primary inputs without pulsing any clocks. Pattern 6 differs from pattern 5 in that it pulses just one clock in the first capture cycle.

Example 1b

Another example of report_patterns showing the case with pulse-always and pulse-in-capture clocks.

report_patterns

```
// pattern #  type           cycles  loads    observe_proc  cap_proc    cap_clock_sequence
// ------  ---------------  ------  -----  ------------  ------------  ------------------
// 0       clock_sequential    2      1    -        -       [core_clk200,*] [core_clk200,*]
// 1       clock_sequential    2      1    -        -       [core_clk200,*] [core_clk200,*]
//
//Note: [*] = "int_shift_reg_clk1,int_shift_reg_clk2,int_shift_reg_clk3", which are
//pulse-always clocks.
```

The “Note” line only appears if there are pulse-in-capture or pulse-always clocks in the design.

Example 2

The following example shows how you can use the set_pattern_filtering command to display a subset of patterns. The example first displays just the non-basic patterns of the preceding example, then displays just the patterns that use ramclk as a capture clock:

set_pattern_filtering -range 2 7

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Internal pattern filtering is on: 6 selected pattern(s), 11 original
//  pattern(s).
```

report_patterns

```
       pre-
       filter
patt # patt # type          cycles loads observe_proc cap_proc  cap_clk_s
------ ------ ------------- ------ ----- ------------ --------- ---------
   0      2   basic           1      1     -            -       [clk1]
   1      3   clock_seq       2      1   shadow_obs   named_cap [clk1]
                                                                [clk1]
   2      4   multiple_load   3      3     -            -       [ramclk]
                                                               S[clk1]
                                                               S[clk2]
   3      5   clock_seq       2      1     -            -     [clk1,clk2]
                                                                [-]
   4      6   clock_seq       2      1     -            -       [clk1][-]
   5      7   multiple_load   3      2     -            -       [clk1]
                                                               S[clk1]
                                                                [clk1]
```

set_pattern_filtering -clocks ramclk

```
//  Warning: The previous filter options have been reset.
//  Internal pattern filtering is on: 1 selected pattern(s), 11 original
//  pattern(s).
```

report_patterns

```
       pre-
       filter
patt # patt # type          cycles loads observe_proc cap_proc cap_clk_s
------ ------ ------------- ------ ----- ------------ -------- ---------
   0      4   multiple_load   3      3     -            -      [ramclk]
                                                              S[clk1]
                                                              S[clk2]
```

Example 3

The following EDT pattern reporting example reports EDT chain test pattern 9. The design is a Modular Compressed ATPG design with two EDT blocks, p1 and p2. Block p1 has one channel and all chains connected to this channel are observed with this pattern. Block p2 has three channels. For each of them, only one chain is observed with this pattern.

report_patterns -pattern_index 9 -edt -chain_test

```
// block ch# channel_name channel_input_pin channel_output_pin obs_chains
// ----- --- ------------ ----------------- ------------------ ----------
// p1     1  p1_channel1  core1_edt_ch_in1  core1_edt_ch_out1  <all conn.
//                                                             chains>
// ---------------------------------------------------------------------
// p2     1  p2_channel1  core2_edt_ch_in1  core2_edt_ch_out1 core2_chain1
// "      2  p2_channel2  core2_edt_ch_in2  core2_edt_ch_out2 core2_chain4
// "      3  p2_channel3  core2_edt_ch_in3  core2_edt_ch_out3 core2_chain7
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

The following example is the same as the preceding example except that the -Verbose switch is included in the command. This expands the “<all connected chains>” summary information to the names of all chains connected to p1_channel1.

report_patterns -pattern_index 9 -edt -chain_test -verbose

```
// block ch# channel_name channel_input_pin channel_output_pin obs_chains
// ----- --- ------------ ----------------- ------------------ ----------
// p1     1  p1_channel1  core1_edt_ch_in1  core1_edt_ch_out1 core1_chain1
//                                                            core1_chain2
//                                                            core1_chain3
//                                                            core1_chain4
//                                                            core1_chain5
// ---------------------------------------------------------------------
// p2     1  p2_channel1  core2_edt_ch_in1  core2_edt_ch_out1 core2_chain1
// "      2  p2_channel2  core2_edt_ch_in2  core2_edt_ch_out2 core2_chain4
// "      3  p2_channel3  core2_edt_ch_in3  core2_edt_ch_out3 core2_chain7
```

Example 5

The following example reports on a particular pattern in an external pattern set for a design that has both uncompressed scan chains (direct access chains) and compressed scan chains. The direct access chains are reported in lines that begin “<uncompressed chain>”. As the design consists of just one block, the “block” column is unnecessary and it is not displayed.

read_patterns my_edt_patts
report_patterns -external -pattern_index 199 -edt -chain_test

```
// ch# channel_name channel_input_pin channel_output_pin obs_chains
// --- ------------ ----------------- ------------------ ------------
//  1  pic_channel1 core1_edt_ch_in1  core1_edt_ch_out1  core1_chain1
//  2  pic_channel2 core1_edt_ch_in2  core1_edt_ch_out2  core1_chain4
//  3  pic_channel3 core1_edt_ch_in3  core1_edt_ch_out3  core1_chain7
// ---------------------------------------------------------------------
// <uncompressed chain> core2_edt_ch_in1  core2_edt_ch_out1  core2_chain1
// <uncompressed chain> core2_edt_ch_in2  core2_edt_ch_out2  core2_chain2
```

Example 6

The following example sets the fault type to path_delay, loads path data into the system, and creates patterns to test the path delay faults. It uses the report_patterns command to check which path faults are detected by the internal patterns.

set_fault_type path_delay
read_fault_sites paths.txt
set_pattern_type -sequential 2
create_patterns
report_patterns -path_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Pattern 0 detects following path fault(s):
//    Detected path   = path1, edge = rise, detection = robust
//    Detected path   = path44, edge = fall, detection = robust
//  Pattern 1 detects following path fault(s):
//    Detected path   = path3, edge = rise, detection = robust
//    Detected path   = path12, edge = fall, detection = robust
//    Detected path   = path16, edge = fall, detection = robust
//    Detected path   = path21, edge = fall, detection = robust
//    Detected path   = path33, edge = fall, detection = robust
//    Detected path   = path34, edge = fall, detection = robust
//    Detected path   = path45, edge = fall, detection = robust
//    Detected path   = path52, edge = fall, detection = robust
//  Pattern 2 detects following path fault(s):
//    Detected path   = path5, edge = fall, detection = robust
//    Detected path   = path44, edge = rise, detection = robust
...
```

Example 7

The following example shows the data source for scan chains associated with the 101 test pattern index.

report_patterns -edt -constant_chains -pattern_index 101

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//
//   block                data source statistics
//  -------  ------------------------------------------------
//  piccpu1  8 of 8 scan chains are driven by constant value
//  -------------------------------------------------------
//  piccpu2  3 of 8 scan chains are driven by constant value
//  -------------------------------------------------------
//  piccpu3  8 of 8 scan chains are driven by constant value
//
//  command: rep patterns  -edt -constant_chains  -patt 101 -verbose
//
//   block       chain      data source
//  -------  ------------  ------------
//  piccpu1  core1_chain1  constant 0
//     "     core1_chain2  constant 0
//     "     core1_chain3  constant 0
//     "     core1_chain4  constant 0
//     "     core1_chain5  constant 0
//     "     core1_chain6  constant 0
//     "     core1_chain7  constant 0
//     "     core1_chain8  constant 0
//  -----------------------------------
//  piccpu2  core2_chain1  decompressor
//     "     core2_chain2  decompressor
//     "     core2_chain3  constant 0
//     "     core2_chain4  constant 0
//     "     core2_chain5  decompressor
//     "     core2_chain6  constant 0
//     "     core2_chain7  decompressor
//     "     core2_chain8  decompressor
//  -----------------------------------
//  piccpu3  core3_chain1  constant 0
//     "     core3_chain2  constant 0
//     "     core3_chain3  constant 0
//     "     core3_chain4  constant 0
//     "     core3_chain5  constant 0
//     "     core3_chain6  constant 0
//     "     core3_chain7  constant 0
//     "     core3_chain8  constant 0
//
```

Example 8

The following example shows how to report patterns that contain some scan cells with X unload values:

report_patterns -unload_x

```
//  pattern #   type  cycles  loads  unload_X observe_proc  capture_proc  capture_clock_seq
//  ---------  -----  ------  -----  -------- ------------  ------------  -------------------
//        0    basic     1      1      yes        -             -        [clock]
//        1    basic     1      1      yes        -             -        [clock]
//        4    basic     1      1      yes        -             -        [clock]
//        5    basic     1      1      yes        -             -        [clock]
//        7    basic     1      1      yes        -             -        [clock]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can then further investigate one of these patterns and report the scan cells that capture X values:

report_patterns -pattern_index 1 -unload_x

```
// pattern # type  cycles loads unload_X observe_proc capture_proc capture_clock_seq
// --------- ----- ------ ----- -------- ------------ ------------ -------------------
//       0   basic    1     1     yes       -             -        [clock]
// Scan cell dff1 (233) captures X
// Scan cell dff3 (235) captures X
```

Example 9

The following example shows the default limit of 20 X captured cells reported:

report_patterns -pattern_index 0 -unload_x

```
// pattern # type  cycles loads unload_x observe_proc capture_proc capture_clock_sequence
// --------- ----- ------ ----- -------- ------------ ------------ ----------------------
//      0    basic  1       1      Yes      -           cap1           [clk1,clk4]
//
// Scan cell '/reg_result_add_clk4_29_/Q' (1274) captures X
// Scan cell '/reg_result_add_clk4_18_/Q' (1262) captures X
...
// Scan cell '/reg_result_add_clk4_7_/Q' (1170) captures X
// Only the first 20 out of 98 X captured cells are reported. Add "-X_cell_report_limit
// unlimited" to show all cells capturing X.
```

Related Topics

read_cpf

read_upf

add_faults

delete_faults

read_faults

report_faults

set_power_metrics

add_primary_inputs

delete_primary_inputs

report_primary_outputs

add_primary_outputs

delete_primary_outputs

report_primary_inputs

report_capture_procedures

add_scan_groups

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_procfile
report_timeplates
write_patterns
write_procfile
Multiprocessing to Reduce Runtime
add_processors
delete_processors
add_read_controls
analyze_control_signals
delete_read_controls
add_register_value
Serial Register Load and Unload for LogicBIST and ATPG
set_lpct_controller
delete_register_value
set_register_value
set_pattern_buffer
report_processors

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_power_data

Context: all contexts

Mode: all modes

Reports the power data currently loaded in the system from either a read_cpf or read_upf command.

Usage

report_power_data {[-Summary | -All | -DOmains [*domain_name* …] | -STates [*state_name* …]} [-ISolation_cells] [-LEvel_shifters] [-Always_on_cells] [-REtention_cells]

Description

Reports the power data currently loaded in the system from either a read_cpf or read_upf command.

Arguments

- -Summary

  An optional switch that specifies the content of power data to be reported. By default, the command reports the summary of loaded power data.

- -All

  An optional switch that reports the details of the loaded power data, including all power domains and all power modes. When executing the command in non-setup mode, the tool reports the count of low power cells associated with each power domain and the gate count of every power domain.

- -DOmains [*domain_name* …]

  An optional switch and optional repeatable string that reports only the specified power domains (*domain_name*) and the data associated with the specified power domains. If no power domain is specified, all power domains are reported.

- -STates [*state_name* …]

  An optional switch and optional repeatable string that reports only the specified power mode and power data related to the specified power modes. If no power mode is specified, all power modes are reported.

- -ISolation_cells

  An optional switch that reports the detailed information of the isolation cells.

- -LEvel_shifters

  An optional switch that reports the detailed information of the level shifters.

- -Always_on_cells

  An optional switch that reports the detailed information of the always-on cells.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -REtention_cells

  An optional switch that reports the detailed information of the retention cells.

Examples

The following example displays detailed power data for isolation cells:

```
report_power_data -isolation_cells
```

Related Topics

add_faults

delete_faults

read_faults

read_cpf

report_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_power_metrics

Context: dft -edt, patterns -scan

Mode: analysis

Displays shift and capture power metrics for specified test patterns.

Usage

report_power_metrics [-patterns {all | pad | range *first_pattern_index last_pattern_index* | *pattern_index…* } ...]
[-sort_capture [wsa_average | wsa_peak | st_average | st_peak] | -sort_shift [load | response]]
[-ascending | -descending]
[-internal | -external]
[-shift [-scan_test | -chain_test | -all_test]
[-cycles {all | peak | range *first_cycle_index last_cycle_index* | *cycle_index*}...]]
[-clock_domains *object_spec*...]
[-instances *object_spec*...] [-modules *object_spec*...] [-partitions *object_spec*...]
[-every_capture_cycle]

Description

Use this information to identify the test patterns that may cause chip failure due to power issues. Enter this command without any arguments to display a summary of shift and capture power metrics for all test patterns.

Note

You can use this command when the set_power_metrics command is set to -Shift off -Capture off and the test patterns are automatically re-simulated to calculate shift and capture power if necessary.

Arguments

- -patterns {all | pad | range *first_pattern_index last_pattern_index* | *pattern_index* } ...

  An optional switch and literal, list of positive integers, or literal and list of positive integers that display the shift power and capture power for specified test patterns. Options include:

  all — Displays the shift power and capture power for all patterns.

  pad — Displays the shift power consumed by the unloading of the last test pattern in the specified test patterns. This value is only available when "set_power_metrics -composite_shift_power" is enabled.

  range *first_pattern_index last_pattern_index* — Displays the shift power and capture power for all patterns from *first_pattern_index* to *last_pattern_index*.

  pattern_index — Displays the shift power and capture power for the specified patterns. Pattern index starts from 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -sort_capture [wsa_average | wsa_peak | st_average | st_peak]

  An optional switch and literal pair that sorts specified test patterns based on their switching activity during capture. Literal options include:

  wsa_average — Average of weighted switching activity.

  wsa_peak — Peak of weighted switching activity.

  st_average — Average of state transitions.

  st_peak — Peak of state transitions. Default.

  For more information on WSA and ST values, refer to "Example 1" on page 2039 below and the set_power_metrics command. Weighted Switching Activity is defined in the *Tessent Scan and ATPG User's Manual*.

- -sort_shift [load | response]

  An optional switch and literal pair that sorts the specified test patterns based on their function during test pattern application. This option is only valid when the -SHIFT option is specified. Literal options include:

  load — Switching activity during the loading of test pattern stimuli.

  response — Switching activity during the unloading of test responses.

- -ascending | -descending

  An optional switch that sorts the specified test patterns in an ascending or descending order based on switching activity. By default, test patterns with the lowest switching activity are listed first (ascending order).

- -internal | -external

  An optional switch that specifies whether power metrics are reported for the internal or external test pattern set. By default, the internal test pattern set is used.

- [-shift {-scan_test | -chain_test | -all_test}]

  An optional pair of switches that reports shift and capture power metrics for a specific pattern type. By default, the power metrics for scan test patterns are reported. Options include:

  -scan_test— Scan test patterns. Default.

  -chain_test — Chain test patterns.

  -all_test — Scan and chain test patterns.

- -cycles {all | peak | range *first_cycle_index last_cycle_index* | *cycle_index...*}

  An optional switch and literal, list of integers, or a literal and a pair of integers that reports shift power for the specified test pattern cycles. You must enable the composite shift power calculation with set_power_metrics to calculate these values. Options include:

  all — Shift transitions for all cycles.

  peak — Shift transitions for the cycle with the peak shift power.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

range *first_cycle_index last_cycle_index* — Shift transitions for a specified range of cycles.

cycle_index — Shift transitions for the specified cycles.

- -clock_domains *object_spec*...

  An optional switch and repeatable string pair that reports the shift and capture power usage for one or more specified clock domains. An *object_spec* consists of one or more clock pin pathnames.

- -instances *object_spec*...

  An optional switch and repeatable string pair that reports the shift and capture power usage for one or more specified design instances. An *object_spec* consists of one or more pathnames.

- -modules *object_spec*...

  An optional switch and repeatable string pair that reports the shift and capture power usage for all instances of one or more specified design modules. An *object_spec* consists of one or more module names.

- -partitions *object_spec*...

  An optional switch and repeatable string pair that reports the shift and capture power usage for all instances of one or more specified design partitions. An *object_spec* consists of one or more partition names.

- -every_capture_cycle

  An optional switch that reports the weighted switching activity and the "state element transition" for every capture cycle on a per pattern basis. By default, the tool reports only the average and peak capture activity for the test pattern.

Examples

Example 1

The following example displays the shift power and capture power for pattern 15.

report_power_metrics -patterns 15

```
//                  Shift                          Capture
//  Pattern    Load    Response          WSA          State Element Tran.
//  -------  --------  --------  -------------------  -------------------
//  15       45.29%    40.56%    13.89% (15.49%)      14.32% (18.13%)
```

- The Shift Load number (45.29 percent) is the percentage of the number of scan cell transitions during the loading of the test stimuli over the worst case. The worst case is calculated by assuming that every scan cell is toggled in every shift cycle.
- The Shift Response number (40.59 percent) is the percentage of the number of scan cell transitions during the unloading of the test response over the worst case. The worst case is calculated by assuming that every scan cell is toggled in every shift cycle.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The first Capture WSA (Weighted Switching Activity) number (13.89 percent) is the average of the weighted switching activity per capture cycle. The second number (15.49 percent) is the weighted switching activity for the peak capture cycle. The peak capture cycle is the cycle with the highest weighted switching activity among all capture cycles in a test pattern.

  The WSA calculation includes the switching activity of the combinational logic, state elements, as well as factoring in gate output fanout for capture cycles.

- The first Capture State Element Tran. number (14.32 percent) is the average number of state element transitions per capture cycle. The second number (18.13 percent) is the number of state element transitions in the peak capture cycle. The peak capture cycle is the cycle with the highest number of state element transitions among all capture cycles in a test pattern.

Example 2

The following example displays a summary of capture power and shift power for all patterns:

report_power_metrics

```
Power Metrics                 Min.   Average   Max.
--------------------------  ------  -------  ------
WSA                          9.03%   18.74%  36.82%
State Element Transitions    8.99%   25.44%  47.33%
--------------------------  ------  -------  ------
Load Shift Transitions      39.06%   44.62%  51.66%
Response Shift Transitions   6.45%   39.38%  50.54%
```

Example 3

The following example calculates and reports the power metrics for external test patterns, *mypat.ascii*:

read_patterns mypat.ascii
report_power_metrics -external

```
Power Metrics                 Min.   Average   Max.
--------------------------  ------  -------  ------
WSA                         10.62%   12.54%  13.65%
State Element Transitions   13.59%   18.71%  20.94%
--------------------------  ------  -------  ------
Peak Cycle
--------------------------
WSA                         10.62%   12.54%  13.65%
State Element Transitions   13.59%   18.71%  20.94%
--------------------------  ------  -------  ------
Load Shift Transitions      15.38%   25.00%  30.77%
Response Shift Transitions  17.95%   21.15%  25.64%
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

The following example reports the power metrics with composite shift for both scan tests and chain tests for all test patterns in the internal test pattern set:

set_power_metrics -composite_shift on
report_power_metrics -shift -all_test -patterns all

```
// Chain test :
//
//           Pattern            Shift
//  Pat. loaded Pat. unloaded   Average
//  ----------- -------------   -------
//  0           -               32.89%
//  1           0               48.60%
//  2           1               54.20%
//  pad         2               49.56%
//
// Scan test :
//
//           Pattern            Shift
//  Pat. loaded Pat. unloaded   Average
//  ----------- -------------   -------
//  0            -              10.60%
//  1            0               8.60%
//  2            1              14.20%
//  3            2               8.20%
//  pad          3               9.56%
```

Example 5

The following example shows the composite shift power for a specified range of cycles in the second and third test pattern in the internal test pattern set. In this example, the peak shift power for test pattern 1 occurs in cycle 6 as indicated by the asterisk cycle index:

set_power_metrics -composite_shift on
report_power_metrics -pattern 1 2 -shift -cycle range 4 6

```
//          Pattern           Cycle   | Shift Transitions
// Pat. loaded Pat. unloaded
// ----------- -------------  ------ | -----------------
// 1           0               4     |            43.82%
//                             5     |            40.76%
//                             6  (*)|            47.64%
// 2           1               4     |            38.82%
//                             5     |            39.54%
//                             6     |            37.56%
//
// (*) indicates the cycle with the peak shift power.
```

Example 6

The following example reports the peak shift power for test pattern 1 and 2:

set power metric -composite_shift on
report_power_metrics -pattern 1 2 -shift -cycle peak

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Pattern                    Peak Cycle | Shift Transitions
// Pat. loaded Pat. unloaded
// ----------- ------------- -----------| -----------------
// 1            0             5         |            40.76%
// 2            1             114       |            42.82%
```

Example 7

The following example shows the switching activity for capture when the -every_capture_cycle argument is specified. In this example, there are three capture cycles in pattern 10.

report power metrics -pattern 10 -every_capture_cycle

```
//  Pattern                      Capture
//                    WSA            State Element Tran.
//  -------  --------------------  ---------------------
//  10       8.23%, 25.21%, 24.68%  1.69%, 22.16%, 19.53%
```

The following example has patterns with different numbers of capture cycles. This occurs when NCPs or clock control definitions are used. This example shows patterns with 1, 2, and 3 capture cycles.

report power metrics -pattern range 12 15 -every_capture_cycle

```
//  Pattern                      Capture
//                    WSA            State Element Tran.
//  -------  --------------------  ---------------------
//  12       8.16%, 21.91%, 23.22%  1.54%, 16.64%, 19.08%
//  13       8.22%, 25.06%,         1.74%, 21.28%
//  14       8.17%, 21.47%, 23.40%  1.52%, 15.78%, 14.42%
//  15       8.21%,                 1.61%,
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 8

The following example shows a power metric report when you have specified capture power control for the clock domain fclk_1:

```
//
// Global
//
Power Metrics                 Min.   Average   Max.
--------------------------  ------  -------  ------
WSA                          9.42%   10.45%  11.56%
State Element Transitions   11.90%   16.17%  18.49%
--------------------------  ------  -------  ------
Peak Cycle
--------------------------
WSA                          9.26%   10.45%  11.56%
State Element Transitions   11.59%   16.17%  18.49%
--------------------------  ------  -------  ------
Load Shift Transitions      12.38%   23.10%  28.47%
Response Shift Transitions  15.95%   19.15%  23.25%
//
// Clock Domain
//   'fclk_1'
//
Power Metrics                 Min.   Average   Max.
--------------------------  ------  -------  ------
WSA                         10.62%   12.54%  13.65%
State Element Transitions   13.59%   18.71%  20.94%
--------------------------  ------  -------  ------
Peak Cycle
--------------------------
WSA                         10.62%   12.54%  13.65%
State Element Transitions   13.59%   18.71%  20.94%
--------------------------  ------  -------  ------
Load Shift Transitions      15.38%   25.00%  30.77%
Response Shift Transitions  17.95%   21.15%  25.64%
```

Related Topics

set_power_metrics

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_power_control

The report_power_control command generates a report on power control settings for the entire design or for instances, clock domains, and partitions with power control enabled.

Usage

report_power_control [-all_settings]
[-global | -instances *object_spec* … | -modules *object_spec* … |
-clock_domains *object_spec* … | -partitions *object_spec* …]
[-verbose]

Arguments

- -all_settings

  An optional switch that reports all power control settings. This is the default.

- -global

  An optional switch that reports only the global power control settings for the top-level instance.

- -instances *object_spec*

  An optional switch and repeatable string pair that reports power control settings for every instance specified by *object_spec*.

- -modules *object_spec*

  An optional switch and repeatable string pair that reports power control settings for every instance of the module or modules specified by *object_spec*.

- -clock_domains *object_spec*

  An optional switch and repeatable string pair that reports power control settings for every clock domain specified by *object_spec*.

- -partitions *object_spec*

  An optional switch and repeatable string pair that reports power control settings for every partition specified by *object_spec*. You can define design partitions with the add_design_partition command.

- -verbose

  An optional switch that reports more detailed information about capture power control settings based on clock gating, which provides insight into whether the capture power of the design can be controlled effectively by the ATPG engine.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example shows the current low power ATPG settings from the set_power_control command:

report_power_control

```
report_power_control
//  ====================================================
//  Control capture power by ATPG globally
//  ====================================================
//    -switching_threshold_percentage                 : 30.00%
//    -rejection_threshold_percentage                 : 35.00% -st_peak
//
//  ====================================================
//  Control shift power by ATPG globally
//  ====================================================
//    -switching_threshold_percentage                 : 15.00%
//    -rejection_threshold_percentage                 : off
```

Example 2

The following example uses the -verbose switch to report the statistics of sequential elements controlled by the clock gater groups for each clock domain:

report_power_control -verbose

```
report_power_control -verbose
//  ====================================================
//  Control capture power by ATPG globally
//  ====================================================
//    -switching_threshold_percentage                 : 30.00%
//    -rejection_threshold_percentage                 : 35.00% -st_peak
//
//  Clock       Scan Elements  Non-scan Elements  Under Clock Gater Control
//  ---------  -------------  -----------------  -------------------------
//  '/CLK1' (1)          18               1                     63.16%
//  '/CLK1_GATED'(10)    25               1                     46.15%
//
//  ====================================================
//  Control shift power by ATPG globally
//  ====================================================
//    -switching_threshold_percentage                 : 15.00%
//    -rejection_threshold_percentage                 : off
```

Example 3

The following example shows a power control report for a design with separate power control for the core_instance_1 instance and the domain fclk_1 clock.

report_power_control

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  ==================================================
//  Control capture power by ATPG globally
//  ==================================================
//    -switching_threshold_percentage                : 15.00%
//    -rejection_threshold_percentage                : off
//
//  ===================================================
//  Control shift power by ATPG globally
//  ===================================================
//    -switching_threshold_percentage                 : 15.00%
//    -rejection_threshold_percentage                 : 20.00%
//
// ==================================================
//  Control capture power by ATPG for instance
//      'core_instance_1'
//  ==================================================
//    -switching_threshold_percentage                : 10.00%
//    -rejection_threshold_percentage                : off
//
// ==================================================
//  Control capture power by ATPG for clock domain
//      'fclk_1'
//  ==================================================
//    -switching_threshold_percentage                : 18.00%
//    -rejection_threshold_percentage                : off
```

Example 4

The following example shows the more detailed report generated with the -verbose option for the same setup as in Example 3. Additional tables list the number of scan and non-scan elements, and the percentage of elements under clock gater control for each clock.

report_power_control -verbose

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  ================================================
//  Control capture power by ATPG globally
//  ================================================
//    -switching_threshold_percentage                : 15.00%
//    -rejection_threshold_percentage                : off
//
//  Clock           Scan Elements  Non-scan Elements  Under Clock Gater Control
//  --------------  -------------  -----------------  -------------------------
//  '/fclk_1' (1)       5929             7278                  97.68%
//  '/fclk_2' (2)       6303             1792                  54.81%
//  '/fclk_3' (3)       362              271                   0.00%
//  ================================================
//  Control shift power by ATPG globally
//  ================================================
//    -switching_threshold_percentage                 : 15.00%
//    -rejection_threshold_percentage                 : 20.00%
//
// ================================================
//  Control capture power by ATPG for instance
//      'core_instance_1'
//  ================================================
//    -switching_threshold_percentage                : 10.00%
//    -rejection_threshold_percentage                : off
//
//  Clock           Scan Elements  Non-scan Elements  Under Clock Gater Control
//  --------------  -------------  -----------------  -------------------------
//  '/fclk_2' (2)       203               12                   81.23%
//  '/fclk_3' (3)        67                1                   12.10%
//
// ================================================
//  Control capture power by ATPG for clock domain
//      'fclk_1'
//  ================================================
//    -switching_threshold_percentage                : 18.00%
//    -rejection_threshold_percentage                : off
//
//  Clock           Scan Elements  Non-scan Elements  Under Clock Gater Control
//  --------------  -------------  -----------------  -------------------------
//  '/fclk_1' (1)       5929             7278                  97.68%
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_primary_inputs

Context: all contexts

Mode: analysis, setup

Displays a list of the primary inputs of a circuit.

Usage

report_primary_inputs [-All | *net_pathname*... | *primary_input_pin*...]
[-Class {Full | User | System}] [-Verbose] [{> | >>} *file_pathname*]

Description

You can choose to display either the user class, system class, or full classes of primary inputs. Additionally, you can display all of the primary inputs or a specific list of primary inputs. If you issue the command without specifying any arguments, the tool displays all the primary inputs.

Arguments

- -All

  An optional switch that displays all the primary inputs. This is the default.

- *net_pathname*

  An optional repeatable string that specifies the circuit connections whose user-class primary inputs to display.

- *primary_input_pin*

  An optional repeatable string that specifies a list of system-class primary input pins to display.

- -Class Full | User | System

  An optional switch and literal pair that specifies the source (or class) of the primary input pins to display. The literal choices are as follows:

  Full — Displays all the primary input pins in the user and system class. This is the default.

  User — Displays only the user-entered primary input pins.

  System — Displays only the netlist-described primary input pins.

- -Verbose

  An optional switch that displays all the nets grouped under a primary input.

- > *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list for creating or replacing the contents of *file_pathname*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- >> *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list for appending to the contents of *file_pathname*.

Examples

Example 1

The following example displays the full classes of primary inputs:

add_primary_inputs indata2 indata4
report_primary_inputs -class full

Example 2

The following example displays the user classes of primary inputs:

add_primary_inputs /dut/*/CLK_X-pseudo_port_name MYCLKX

```
//Note: Adding primary input MYCLKX merging 224 nets.
```

report_primary_inputs -class user

```
USER: '/MYCLKX' (merged internal pin)
Merges 224 nets
```

Example 3

The following dft -scan example displays all of the primary inputs:

report_primary_inputs -all

```
SYSTEM:    '/clk'
SYSTEM:    '/datain'
```

The label “system” means that these are primary inputs that the tool automatically recognizes because they were in the netlist.

Related Topics

add_primary_inputs

delete_primary_inputs

report_primary_outputs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_primary_outputs

Context: all contexts

Mode: analysis, setup

Displays the specified primary outputs.

Usage

report_primary_outputs [-All | *net_pathname*... | *primary_output_pin*...]
[-Class {Full | User | System}] [{> | >>} *file_pathname*]

Description

The report_primary_outputs command displays a list of primary outputs of a circuit. You can choose to display either the user class, system class, or full classes of primary outputs. Additionally, you can display all of the primary outputs or a specific list of primary outputs. If you issue the command without specifying any arguments, the tool displays all the primary outputs.

Arguments

- -All

  An optional switch that displays all primary outputs. This is the default.

- *net_pathname*

  An optional, repeatable string that specifies the circuit connections whose user-class primary outputs you want to display.

- *primary_output_pin*

  An optional, repeatable string that specifies a list of system-class primary output pins that you want to display.

- -Class Full | User | System

  An optional switch and literal pair that specifies the source (or class) of the primary output pins that you want to display. The literal choices are as follows:

  Full — A literal that displays all of the primary output pins in the user and system class. This is the default.

  User — A literal that displays only the user-entered primary output pins.

  System — A literal that displays only the netlist-described primary output pins.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example displays all primary outputs in the user class:

```
add_primary_outputs outdata1 outdata3 outdata5
report_primary_outputs -class user
```

Example 2

The following dft -scan example displays all of the primary outputs:

```
report_primary_outputs -all
```

```
SYSTEM:     '/dataout'
SYSTEM:     '/dataout1'
```

The label “system” means that these are primary outputs that the tool automatically recognizes because they were in the netlist.

Related Topics

add_primary_outputs

delete_primary_outputs

report_primary_inputs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_procedures

Context: dft -edt, dft -scan, patterns -scan, patterns -ijtag, patterns -scan_retargeting, patterns -failure_mapping

Mode: setup, analysis

Displays the specified procedure.

Usage

report_procedures {***procedure_name*** [*group_name*] | **-All**} [-COre *core_name* [-MOde *mode_name*]]
[{> | >>} *file_pathname*] [-EXpand_iprocs]

Description

Displays the specified procedure.

The report_procedures command displays all procedures or the specified procedure.

Arguments

- ***procedure_name***

  A string that specifies which procedure to display.

- *group_name*

  An optional string that specifies a particular scan group from which to display the specified procedure.

- -**All**

  A switch that specifies for the tool to display all procedures. This is the default.

- -COre *core_name*

  An optional switch and string pair that specifies that the tool reports the procedures for a given core. The core must exist, or an error is issued.

- -MOde *mode_name*

  An optional switch and string pair that specifies that the tool reports the procedures for the given mode of a core defined by the -core *core_name* switch. The tool reports an error if this switch is specified without the -core switch. This switch is not necessary when there is only one mode for a core. If a core has more than one mode, an error is issued if the mode is not specified.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of file_pathname.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of file_pathname.

- -EXpand_iprocs

  An optional switch that reports the final solution that is used in the test_setup or test_end procedures for any embedded iCalls and iMerge in those procedures. If an iCall is used within an iMerge, then that entire iMerge is expanded. The start and end of the expanded cycles is marked by comments in the reported procedure or procedure file. Use this switch to when debugging E14 DRC violations.

Examples

Example 1

The following example displays all procedures:

report_procedures -all

Example 2

The following example shows using the -EXpand_iprocs switch to work with the set of used ports and vector callbacks in:

ANALYSIS> report_procedure test_setup -expand_iprocs

```
procedure test_setup =
  timeplate tp1 ;
  iReset;
  // Begin expanded iCall m8051_B1_edt_i.setup edt_low_power_shift_en on ;
  // cycle 1 start at time 40
  cycle =
    annotate "+ Targets: …"
     …
    end;
  // End expanded iCall
  // cycle 5 starts at time 200
  cycle =
    …
  end;
end;
```

Example 3

This is an example of using -core and -mode.

report_procedures -core piccpu_maxlen16_1 -mode

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
procedure capture my_cap =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force_pi ;
        force core_1_edt_clock 0 ;
        force core_1_ramclk 0 ;
        pulse core_1_clk ;
    end;
 end;
 procedure capture =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force_pi ;
        measure_po ;
        pulse_capture_clock ;
    end;
 end;
 procedure clock_sequential =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force_pi ;
        pulse_capture_clock ;
        pulse_read_clock ;
        pulse_write_clock ;
    end;
 end;
 procedure shift =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force_sci ;
        force core_1_edt_update 0 ;
        measure_sco ;
        pulse core_1_clk ;
        pulse core_1_edt_clock ;
    end;
 end;
 procedure load_unload =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force core_1_clk 0 ;
        force core_1_edt_clock 0 ;
        force core_1_edt_update 1 ;
        force core_1_ramclk 0 ;
        force core_1_reset 0 ;
        force core_1_scan_enable 1 ;
        pulse core_1_edt_clock ;
    end ;
    apply shift 16;
 end;
 procedure test_setup =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
            force core_1_edt_clock 0 ;
            force core_1_reset 0 ;
            force core_1_scan_enable 0 ;
        end;

    end;
```

Example 4

This is an example of using -code with a specific procedure.

report_procedures load_unload -core piccpu_maxlen16_1

```
 procedure load_unload =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force core_1_clk 0 ;
        force core_1_edt_clock 0 ;
        force core_1_edt_update 1 ;
        force core_1_ramclk 0 ;
        force core_1_reset 0 ;
        force core_1_scan_enable 1 ;
        pulse core_1_edt_clock ;
    end ;
    apply shift 16;

    end;
```

Related Topics

add_scan_groups

read_procfile

report_capture_procedures

report_timeplates

write_patterns

write_procfile

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_processors

Context: unspecified, all contexts

Mode: setup, analysis, insertion (dft -edt context only)

Displays information about the processors used for the current multiprocessing environment.

Usage

report_processors [-Verbose]

Description

The report_processors command prints a table containing information about the processors used for the current multiprocessing environment.

For more information, refer to "Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual.*

When multithreading is enabled, the report_processors command displays the information listed in the following table for the current multiprocessing environment:

Table 7-3. Default report_processors Information Display

Column Name	Description
hosts	Network names of host machines
threads	The number of threads launched on that host
arch	Architecture of host machines (similar to entering "uname -m" on each host)
CPU(s)	Number of hardware processors and their speed for each host machine
%idle	Current CPU idle percentage (a small interval of time is sampled)
free RAM	A machine's total free memory
process size	Current memory usage of the primary or secondary process

Note

The column names change when multithreading is disabled.

This information enables you to determine whether to use the current set of host machines for secondary processes or whether to use different machines. For example, a host machine with a low %idle is not as likely to provide as much CPU time to secondary processes as one that is 100 percent idle. The free RAM may also help you decide which machines to avoid at the present time.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Be sure to consider the number of processors on the host machine when evaluating the idle percentage (%idle), as a partially idle machine may have one or more completely idle processors available to run secondary processes.

Arguments

- -Verbose

  An optional switch that adds the data listed in Table 7-4 to the normal display.

  The primary CPU time reported is cumulative. The secondary CPU time reported is the sum of the CPU time for all the threads reported for the most recent distribution session.

Table 7-4. Additional Data Displayed by -Verbose

Column Name	Description
free swap	Total available swap space minus the swap space currently in use
total RAM	Total available RAM
total swap	Total available swap space
user CPU	Current sum of the time spent executing the primary or secondary process
sys CPU	Current sum of the time spent in system routines on behalf of the primary or secondary process

Examples

Example 1

The following example summarizes the performance statistics for the current multiprocessing environment:

report_processors

```
// hosts          threads arch      CPU(s)       %idle     free RAM    process size
// ------------- ------- ----    -----------  -------   -----------  ------------
// nemo (primary)      1 x86-64  24 x 2.8 GHz      6%     70229.83 MB   19320.18 MB
// ahab                4 x86-64  16 x 3.0 GHz     39%     15463.78 MB     458.61 MB
// Primary process with 1 thread and 1 secondary process with 4 threads running.
```

Example 2

This example uses the -verbose switch for this command. The first “report_processors -verbose” command reports that the cumulative CPU time for the primary process prior to running create_patterns is 18m.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When you subtract the 18m CPU time reported by that command from the 11h 51m CPU time for the primary process reported by the second instance of the command, you have the total primary CPU time for the create_patterns command, 11h 33m.

The CPU time reported for the secondary threads is the sum of the thread CPU times for the latest distribution session.

The tabular output of this example is wrapped, as indicated with an ellipsis (…), for easy reading.

report_processors -verbose

```
//  sub-command: report_processors -verbose
//  hosts           threads    arch       CPU(s)    %idle   free RAM     …
//  --------------  -------  -------  -----------  -----  -----------   …
//  nemo (primary)      1    x86-64  24 x 2.8 GHz     6%  70229.83 MB   …
//  ahab                4    x86-64  16 x 3.0 GHz    39%  15463.78 MB   …

      free swap    total RAM    total swap   process size   user CPU   sys CPU
    ------------ ------------  ------------  ------------  ---------  -------
    307199.70 MB  258086.16 MB  307199.99 MB   19320.18 MB  18m  5.3s   24.6s
     47688.60 MB  129012.48 MB   47999.99 MB     458.61 MB       0.3s    0.1s
//  Primary process with 1 thread and 1 secondary process with 4 threads running.

//  sub-command: create_patterns
….

//  sub-command: report_processors -verbose
//  hosts           threads   arch       CPU(s)    %idle   free RAM    …
//  --------------  -------  -------  -----------  -----  -----------  …
//  nemo (primary)      1    x86-64  24 x 2.8 GHz    87%  97755.77 MB  …
//  ahab                4    x86-64  16 x 3.0 GHz    93%  35058.28 MB  …

      free swap     total RAM    total swap    process size   user CPU     sys CPU
    ------------  ------------  ------------  ------------  -----------  ---------
    307199.70 MB  258086.16 MB  307199.99 MB   33636.34 MB  11h 51m 37s   2m 51.9s
     47487.77 MB  129012.48 MB   47999.99 MB   30711.03 MB  18h  5m  6s  10m  8.1s
//  Primary process with 1 thread and 1 secondary process with 4 threads running.
```

Related Topics

add_processors

delete_processors

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_read_controls

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays all of the currently defined read control lines.

Usage

report_read_controls [{> | >>} *file_pathname*]

Description

Displays all of the currently defined read control lines.

The report_read_controls command displays all the read control lines specified using the add_read_controls command. The display also includes the corresponding off-state with each read control line.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of file_pathname.

Examples

The following example displays a list of the current read control lines:

```
add_read_controls 0 r1 r3
add_read_controls 1 r2 r4
report_read_controls
```

Related Topics

add_read_controls

analyze_control_signals

delete_read_controls

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_read_verilog_options

Context: unspecified, all contexts

Mode: all modes

Reports options specified with the set_read_verilog_options command.

Usage

report_read_verilog_options

Description

Reports options specified with the set_read_verilog_options command.

Arguments

None

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_register_value

Context: all contexts

Mode: setup, analysis

Reports register information for each register name.

Usage

report_register_value [(> | >>) *file_pathname*]

Description

Reports register information for each register name.

In the report, the columns contain the following data:

- Register Value Name — Name of the register specified by the add_register_value command in the pattern generation dofile.
- Value string — Binary value loaded into the register, if known. The register is loaded when you run the write_patterns command. Before write_patterns is run, this column contains Xs for registers with dynamic contents. For register values with a width greater than 200, only the first 200 bits of the value are displayed.
- Width — Number of bits in the register.
- Dynamic/Static — Shows the DRC impact.
  - Dynamic: The contents of the register is not known/not used during DRC and can be changed.
  - Static: The contents of the register is known and used during DRC and cannot be changed.
- Source — The source of the register contents: user-defined or tool-determined.
- LSB/MSB First — Whether the register is applied from MSB > LSB or LSB > MSB.
- Pin Inversion — Indicates whether any inversions are present. Contains two values that refer to input_pin_inversion and output_pin_inversion: F = false and T = true.

See "Serial Register Load and Unload for LogicBIST and ATPG" in the *Tessent Shell User's Manual* for complete information.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following example reports the values of LPCT configuration data fields before and after the write_patterns command is run. In this example, if you issue "set_lpct_controller -max_shift 1000", the lpct_config_shift_length register has the width of 10, as shown below.

report_register_value

Report output before the write_patterns command is run:

```
Register value name                  Value    Width   Dynamic/ Source    LSB/      Pin
                                     string           Static             MSB       Inversion
                                                                         First
------------------------------------ ------   -----   -------- ------    -------   ------
lpct_config_capture_depth            xx       2       dynamic  capture   LSB       FF
                                                               width     first
lpct_config_chain_pattern_load_count XXXX     10      dynamic  chain
                                     XXXX                      patterns  LSB       FF
                                     XX                        load/     first
                                                               unload
                                                               count
lpct_config_edt_bypass               0        1       static   user      MSB       FF
                                                               specified first
lpct_config_reset_control            0        1       static   user      MSB       FF
                                                               specified first
lpct_config_scan_en_control          0        1       static   user      MSB       FF
                                                               specified first
lpct_config_scan_pattern_load_count XXXX      20      dynamic  scan      LSB       FF
                                     XXXX                      patterns  first
                                     XXXX                      load/
                                     XXXX                      unload
                                     XXxx                      count
lpct_config_shift_length             XXXX     10      dynamic  shift     LSB       FF
                                     XXXX                      length    first
                                     XX
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Report output after the write_patterns command is run:

```
Register value name                  Value    Width   Dynamic/ Source    LSB/      Pin
                                     string           Static             MSB       Inversion
                                                                         First--------------
--------------------- ------   -----   -------- ------    -------   ------
lpct_config_capture_depth            10       2       dynamic  capture   LSB       FF
                                                               width     first
lpct_config_chain_pattern_load_count 000001   10      dynamic  chain     LSB       FF
                                             0110              patterns  first
                                                               load/
                                                               unload
                                                               count
lpct_config_edt_bypass               0        1       static   user      MSB       FF
                                                               specified first
lpct_config_reset_control            0        1       static   user      MSB       FF
                                                               specified first
lpct_config_scan_en_control          0        1       static   user      MSB       FF
                                                               specified first
lpct_config_scan_pattern_load_count  000000   20      dynamic  scan      LSB       FF
                             000001                patterns  first
011111                     load/
                                     11                       unload
                                                              count
lpct_config_shift_length             000010   10      dynamic  shift     LSB       FF
                                     1101                     length    first
```

Related Topics

add_register_value

delete_register_value

set_register_value

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_resources

Context: unspecified, all contexts

Mode: all modes

Displays the machine resources used by the active tool session.

Usage

report_resources

Description

Displays the machine resources used by the active tool session.

The report_resources command displays the cumulative processing time and the process memory usage for the machine on which the tool is running.

The tool also reports CPU time that is the cumulative CPU time of the master process since you invoked the run.

Arguments

None

Examples

The following example reports on machine resources:

report_resources

```
// Machine Name: bongo
// User Name: smith
// User CPU Time: 27.8 seconds
// System CPU Time: 1.6 seconds
// Memory Used: 141.739 MB
```

Related Topics

set_pattern_buffer

report_processors

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_rtl_to_gates_mapping

Context: all contexts

Mode: all modes

Reports the rules for mapping names from RTL to post-synthesis/post-layout.

Usage

report_rtl_to_gates_mapping

Description

Reports the rules for mapping names from RTL to post-synthesis/post-layout. This reports both default rules and rules added with the add_rtl_to_gates_mapping command.

See the add_rtl_to_gates_mapping command description for information on these mappings and how to customize them.

Arguments

None.

Examples

The following example calls the report_rtl_to_gates_mapping before any RTL to gates mappings were added using the add_rtl_to_gates_mapping command. In this case, the report shows the built-in default mappings. The add_rtl_to_gates_mapping command is used to add an instance name mapping, a port name mapping, and a port name substring mapping. The second report_rtl_to_gates_mapping invocation shows the added mappings.

report_rtl_to_gates_mapping

```
//  Index  Register Name Prefix  Register Name Suffix
    -----  --------------------  --------------------
    1                                          _reg
    2         rtlcreg_
    3
```

add_rtl_to_gates_mapping -instance_name_map_list {corea/coreb corea}
** -port_name_map_list {joe1.jim1 joey1.jimmy1}**
**add_rtl_to_gates_mapping **
** -port_name_substring_map_list {zz yy}**
report_rtl_to_gates_mapping

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Index  Register Name Prefix  Register Name Suffix
//  -----  --------------------  --------------------
//   1                                        _reg
//   2       rtlcreg_
//   3
//  Index  Original instance name  Mapped instance name
//  -----  ----------------------  --------------------
//  1         corea/coreb                  corea
//  Index  Original Port Name        Mapped port name
//  -----  ------------------        ----------------
//  1         joe1.jim1                    joey1.jimmy1
//  Index  Original Port Name Substring  Mapped Port Name Substring
//  -----  ----------------------------  --------------------------
//  1         zz                           yy
```

Related Topics

add_rtl_to_gates_mapping

delete_rtl_to_gates_mapping

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_run_synthesis_options

Context: unspecified, dft, patterns

Mode: setup, analysis, insertion

Reports the values for all options set with the set_run_synthesis_options command.

Usage

report_run_synthesis_options
[-synthesis_tool {dc_shell | genus}]

Description

Reports the values for all options set with the set_run_synthesis_options command.

Arguments

- -synthesis_tool dc_shell | genus

  An optional switch and value pair that selects for which synthesis tool the report is produced. The default value is “dc_shell”.

  Only the options specific to that tool are reported.

  Note

  When that switch/value pair is not specified, only the options that apply to all synthesis tools, such as ‘default_synthesis_tool’ and ‘synthesis_output_directory’, are reported.

Examples

The following example demonstrates using of this command:

```
SETUP> set_run_synthesis_options dc_shell -startup_file .synopsys_dc.setup
SETUP> report_run_synthesis_options
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
SETUP> report_run_synthesis_options

// Synthesis options
// =================
// --------------------------  ------------------
// Option                      Value
// --------------------------  ------------------
// default_synthesis_tool      dc_shell
// synthesis_output_directory  ./synthesis_outdir
//
SETUP> report_run_synthesis_options -synthesis_tool genus

// Synthesis options for 'genus'
// ============================
// -------------------------  ------------
// Option                     Value
// -------------------------  ------------
// compilation_options        {}
// pre_compilation_commands   {}
// post_compilation_commands  {}
// startup_file               ./genus.init
// command_name               genus
//
SETUP> set_run_synthesis_options -default_synthesis_tool genus genus -startup_file ./
genus.init
SETUP> report_run_synthesis_options

// Synthesis options
// =================
// --------------------------  ------------------
// Option                      Value
// --------------------------  ------------------
// default_synthesis_tool      genus
// synthesis_output_directory  ./synthesis_outdir
//
SETUP> report_run_synthesis_options -synthesis_tool genus

// Synthesis options for 'genus'
// ============================
// -------------------------  ------------
// Option                     Value
// -------------------------  ------------
// compilation_options        {}
// pre_compilation_commands   {}
// post_compilation_commands  {}
// startup_file               ./genus.init
// command_name               genus
//
SETUP>
```

Related Topics

check_synthesis

get_run_synthesis_options

run_synthesis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_run_synthesis_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_scan_cells

Context: dft -edt, dft -scan, dft -test_points, patterns, patterns -scan, patterns -scan_diagnosis

Mode: analysis, insertion (dft -scan context only)

Displays a report on the scan cells that reside in the specified scan chains.

Usage

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

report_scan_cells [-All | {*chain_name* [{-Range *cell_id1 cell_id2*} | *cell_id*…]}...]
[-VERilog] [-Power_domains [*power_domain_name*...]] [-SUbchain]
[-Display [FLAt_schematic | HIErarchical_schematic]] [{> | >>} *file_pathname*]

Context: dft -scan

report_scan_cells [-All | *chain_name*…] [-SHift_register_flops]
[-VERilog] [-Power_domains [*power_domain_name*...]]
[-SCan_mode *scan_mode*] [-Filename *filename* [-Replace]]
[{> | >>} *file_pathname*]

Context: dft -test_points -no_rtl

report_scan_cells [-All | *chain_name*…] [-SHift_register_flops]
[-Power_domain] [-Filename *filename* [-Replace]] [{> | >>} *file_pathname*]

Description

The report_scan_cells command provides a report on the scan cells within specific scan chains. The report provides the following information for each scan cell:

- Chain cell index number (cell ID), where 0 is the scan cell closest to the scan-out pin.
- Scan chain in which the scan cell resides.
- Scan group in which the scan cell resides. (The scan group data is hidden if the design has only scan group.)
- Sequential element type, primitive type, and latch triggering type during shifting. The latch triggering type can be one of the following: LE, TE, AH, AL.

Table 7-5. Sequential Element Types

Cell Type	Summary
MASTER	primary sequential gate of a scan cell holding the load and unload values
SLAVE	sequential gate on the scan path that loads the same (inverted) value as the master cell by a different shift clock pulse
SHADOW	sequential gate off the scan path that loads the same (inverted) value as the master gate

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 7-5. Sequential Element Types (cont.)

Cell Type	Summary
OBS_SHADOW	shadow cell that is also observable through a shadow_observe procedure
COPY	sequential gate on the scan path that loads the same (inverted) value as the master cell in the same shift clock pulse
EXTRA	sequential gate on the scan path between the master and the slave cell that loads the same (inverted) value as the master cell by a different shift clock pulse
TLA	transparent latch on scan path not driven directly by a scan input pin
SCI TLA	transparent latch on scan path driven directly by a scan input pin

- Inversion data for each sequential element.
- Gate index number for each sequential element.
- Shift clock name and inversion status at the clock input of each sequential element. ("dft -edt" and "patterns -scan" only)
- Top-level library model cell name of each sequential element.
- Instance name of each sequential element.
- Library instance name, if one is defined.
- Cell input and output pins.

For more information, see "Scan Terminology" in the *Tessent Scan and ATPG User's Manual*.

If you issue the command without specifying any arguments, the tool displays a report on the scan cells for all scan chains.

Tip

If you are saving a log of the tool session, reporting all scan cells fills the logfile with information for every scan cell—a lot of information if the design is large! To reduce the impact on the logfile in this case, consider reporting on only one scan chain.

Note

Although scan cells are listed in the order of nearest-to-output first, the sequential elements inside one cell are not listed in that order (the master is always listed first).

Be aware that when reading hierarchical names into its internal database, the tool hides backslashes (\), converts periods (.) to slashes (/), and ignores spaces. For example, the tool

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reads the cell name, “FFH.\E/MY_SCAN_FF .SCAN_FF5” in a WGL pattern, as “FFH/E/MY_SCAN_FF/SCAN_FF5”, storing the latter in its internal database. To report on this cell, you must reference it by the name that is stored internally.

Note

The difference between the internal hierarchical name and the name in the WGL file of the preceding example does not alter the correctness of the tool. That is, the output pattern file remains consistent with the original netlist.

dft -scan only: If you issue the command with the -Filename switch, then the tool writes the scan cells to a file in the format that can be read by the insert_test_logic command. The format of the written file is different than the format of the viewed report. You can optionally edit the scan cell order in the file before reading the file with the insert_test_logic command.

Arguments

- -All

  An optional switch that displays information for all the scan cells in all the scan chains. This is the default.

- *chain_name*

  An optional, repeatable string that specifies a scan chain whose scan cell data you want to display. When you use this argument without accompanying cell IDs, the tool displays information for all the scan cells in the scan chain. If you follow the chain_name argument with the -Range switch (and beginning and ending cell IDs for the range), the tool displays information for every cell in the range. If you provide one or more cell IDs without the -Range switch, the tool displays information for just those cells in that chain.

- -Range *cell_id1 cell_id2*

  An optional switch and pair of integers that specify the beginning and ending cell IDs of a range of scan cells in *chain_name* whose data you want to display.

- *cell_id*

  An optional repeatable integer that specifies the index number (cell ID) of a scan cell and, together with the chain_name argument, identifies a specific scan cell whose data you want to display.

- -VERilog

  An optional switch that outputs the cell instance and pin pathnames in Verilog syntax, rather than using the default netlist independent format.

- -SHift_register_flops (dft -scan only)

  An optional switch that specifies to print only the shift register flip-flops that are stitched into scan chains.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Filename *filename* [-Replace] (dft -scan only)

  An optional switch and string that specifies that the tool writes the list of scan cells to a file. The format of the written file is different than the format of the viewed report; see the insert_test_logic filename argument for detailed information on the file format. The -Replace switch specifies that the file should be overwritten if it already exists.

- -Power_domains *power_domain...*

  An optional switch and repeatable string pair that creates a "Power Domain" column in the report when applicable.

- -Scan_mode *scan_mode*

  An optional switch and string pair that specifies the scan mode for which you want to tool to report scan cell information.

- -SUbchain

  An optional switch that, in analysis mode only, displays the scan cells in the tool-learned subchains. The tool-learned subchains are segments of scan cells inside a library model and must perform serial shifts during the parallel Verilog simulation.

- -Display FLAt_schematic | HIErarchical_schematic

  An optional switch and string pair that displays the scan cells in the Tessent Visualizer Flat or Hierarchical Schematic window.

  Note

  This switch is not supported with the -All switch.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

This example displays a list of the scan cells for scan chain0. The design only has one scan group; therefore, the scan group column is not shown.

```
report_scan_cells chain0
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
cell# chain   memory_type    inv gate# shift_clock inv cell_name instance_name (ext.pin
names)
----- ------ -------------- ---- ----- --------------- --------- ------------- ----------
0     chain0 MASTER (LA-AH) FFTT 24848 /clk        F   DFF3      /Q_I4  ""     (TI,QN)
1     chain0 MASTER (LA-AH) FFFF 25040 /clk        F   DFF1      /H_14  ""     (TI,QN)
             SLAVE  (FF-TE) FFFF 24849 /clk        T   DFF2      /Q_I4  ""     (TI,QN)
2     chain0 MASTER (LA-AH) FFFF 24546 /clk        F   DFF1      /R_I1  ""     (TI,QN)
             SLAVE  (FF-TE) FFFF 25041 /clk        T   DFF2      /Q_I4  ""     (TI,QN)
3     chain0 MASTER (LA-AH) FFFF 24542 /clk        F   DFF1      /R_I1  ""     (TI,QN)
             SLAVE  (FF-TE) FFFF 24547 /clk        T   DFF2      /Q_I4  ""     (TI,QN)
4     chain0 MASTER (LA-AH) FFFF 24914 /clk        F   DFF1      /S_I1  ""     (TI,QN)
             SLAVE  (FF-TE) FFFF 24543 /clk        T   DFF2      /Q_I4  ""     (TI,QN)
5     chain0 MASTER (LA-AH) FFFF 24544 /clk        F   DFF1      /Z_I4  ""     (TI,QN)
             SLAVE  (FF-TE) FFFF 24915 /clk        T   DFF2      /Q_I4  ""     (TI,QN)
6     chain0 MASTER (LA-AH) FFFF 24540 /clk        F   DFF2      /N_I3  ""     (TI,QN)
             SLAVE  (FF-TE) FFFF 24545 /clk        T   DFF2      /Q_I4  ""     (TI,QN)
```

The first column in the report displays the chain cell index number, where 0 is the scan cell closest to the scan-out pin.

The second column displays the name of the scan chain the scan cell is in.

The third column displays the scan cell's sequential element type(s), primitive type, and latch triggering type during shifting. The latch triggering type can be any of the following: LE, TE, AH, AL.

The fourth column (inv) contains four flags that show inversion data using F (false) to indicate no inversion difference and T (true) to indicate inversion difference.

The following table illustrates what the value of each flag indicates, depending on whether library cell pins are present or absent at the input and output for each scan cell. An unused value for a position shows a false (F) inversion flag, but this value is not used.

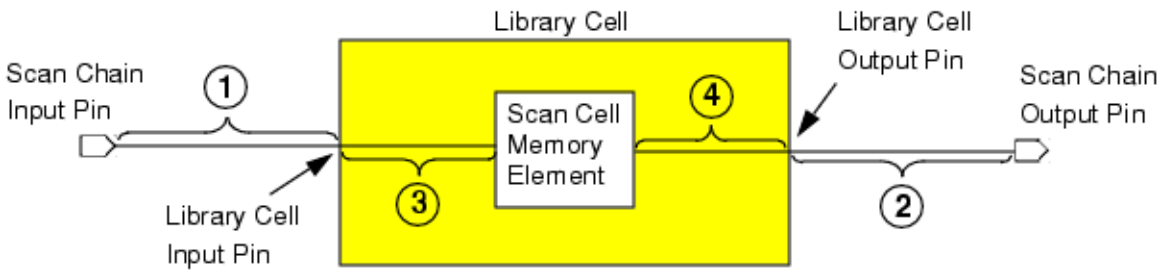
Table 7-6. Library Cell Pins and Inversion Flag Position Values

Library Cell Pin Connection		Inversion Flag Position Value			
Input	**Output**	**1**	**2**	**3**	**4**
Present	Present	Figure 7-4	Figure 7-4	Figure 7-4	Figure 7-4
Present	Absent	Figure 7-4	Figure 7-5	Figure 7-4	Unused value
Absent	Present	Figure 7-5	Figure 7-4	Unused value	Figure 7-4
Absent	Absent	Figure 7-5	Figure 7-5	Unused value	Unused value

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

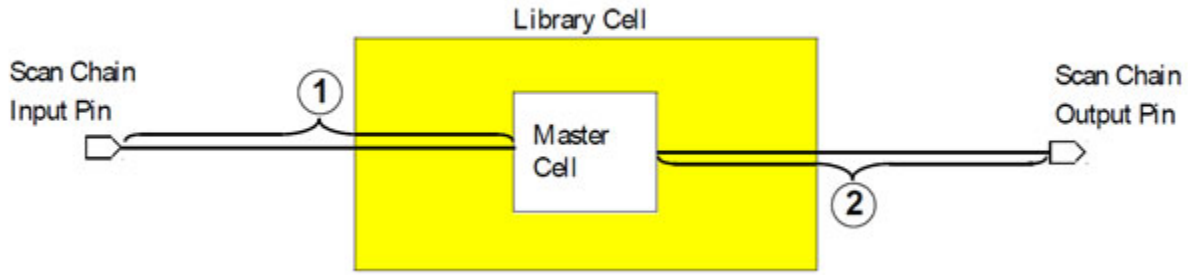
The following diagram illustrates the four parts of the “inv” column in cases where library cell input and output pins exist:

Figure 7-4. Inversion Flags With Library Cell Pins



In cases where the master cell is not directly connected to one or both library cell input/output pins, parts 1 and 2 of the “inv” column refer to the entire length between the scan chain pin (input or output) and the master cell. In these cases, there are intervening elements between the library cell input or output pin and the master cell, so that the master cell is not connected directly to the library cell pin:

Figure 7-5. Inversion Flags Without Library Pins



Where library cell pins are directly connected, the report presents these details (left-to-right) as follows:

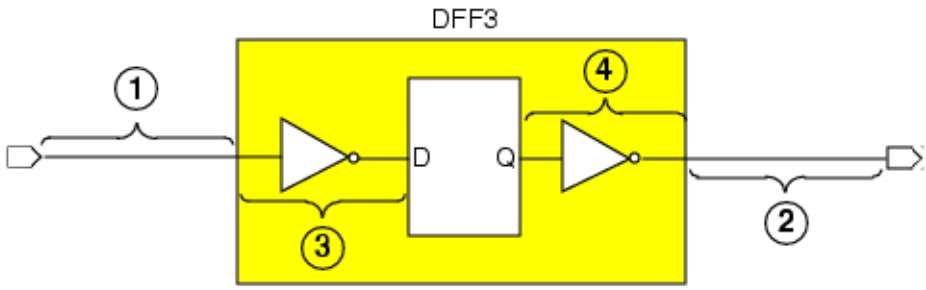
- Inversion of the signal between the scan chain input pin and the library cell input pin.
- Inversion of the signal between the library cell output pin and the scan chain output pin.
- Inversion of the signal between the library cell input pin and the scan cell memory element (primitive).
- Inversion of the signal between the scan cell memory element (primitive) and the library cell output pin.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In cases where library cell pins are absent (that is, not directly connected to the master cell), the report presents the following details instead:

- **Library Cell Input Pin Absent —** Inversion of the signal between the scan chain input pin and the master cell data port.
- **Library Cell Output Pin Absent —** Inversion of the signal between the master cell output port and the scan chain output pin.

For example, scan cell 0 in the previous scan cell report for chain0 shows inversion data FFTT. This indicates that library cell DFF3 contains an inverter on the input data path as well as one on the output path as shown here:



The inversion data is useful when comparing scan cell data reported in the tool ("set_gate_report -pattern_index <*n*>") to values stored in the pattern file. Since different pattern formats handle inversions differently, the scan cell inversion data can also be useful in understanding differences between pattern formats.

Refer also to "Example 7" on page 2080 for more examples of this column.

The fifth column displays the gate index number.

The sixth column displays the name of the shift clock for the scan cell.

Note

In the rare case where a single scan cell's clock port is driven by multiple shift clocks at the same time, the tool reports the data for only one of the clocks. An example would be two different shift clocks with identical waveforms ANDed together to drive the scan cell's clock port (not recommended due to the potential for clock skew).

The seventh column shows whether the shift clock's defined inactive state corresponds to the cell's clock input value that for level-sensitive devices results in the cell becoming inactive, or for edge-triggered devices is the initial value of a capturing transition. If they correspond (no inversion), an F is listed. If they do not correspond (inversion), a T is listed. For more information on the meaning of "inactive" as it applies to scan clocks, refer to the add_clocks command description.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

If the scan cell is a hierarchical model, inversion is indicated with respect to the clock ports of sequential element primitives instantiated within the model, not with respect to the clock input(s) at the top level of the model.

The eighth column displays the cell name of the sequential element.

The ninth column displays the instance name of the sequential element.

The tenth column displays the library instance name. If there is no library instance name, the report shows two double-quotes in the library instance name field.

Note

To see the name of the library model corresponding to a particular scan cell, open Tessent Visualizer with the open_visualizer command. Open the Instance Browser tab and find the instance name of the scan cell. The library model is listed as “Module name” in the table.

The last column displays the library cell input and output pins of the scan cell. If the scan cell input or output pin does not directly connect to the library cell boundary pin, the report shows a dash (-) in the corresponding pin field.

Example 2

The following example displays a list of all scan cells in the analysis system mode:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 outdata4
set_system_mode analysis
report_scan_cells
```

```
-----------------------------------------------------------------------
         Chain    Group                                            Clock
CellNo   Name     Name     Pathname   CellName   ScanOut  Clock    Polarity
-----------------------------------------------------------------------
0        chain1   group1   /MQ_I400   sffr       Q        clk2      (+)
1        chain1   group1   /FH_I400   sffr       QB       clk2      (+)
2        chain1   group1   /FQ_I10    sffr       QB       clk2      (+)
-        chain1   group1   /lckup1    latch      Q        clk1      (-)
3        chain1   group1   /RP_I10    sffr       Q        clk1      (+)
4        chain1   group1   /IS_I10    sffr       Q        clk1      (+)
5        chain1   group1   /CZ_I400   sffr       QB       clk1      (+)
-----------------------------------------------------------------------
```

- The first column displays the chain cell index number, where 0-0 is the scan cell closest to the scan-out pin.
- The second column displays the chain name where the scan cell resides.
- The third column displays the group name where the scan cell resides.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The fourth column displays the hierarchical path of the scan cell.
- The fifth column displays the library model name for the scan cell.
- The sixth column displays the scan out port of the scan cell.
- The seventh column displays the clock for the scan cell.
- The eighth column displays the polarity of the clock of the scan cell.

Example 3

The following dft -scan example adds the new column ShiftRegID/CellNo to the report when it identifies a shift register in the netlist. This column contains a tool-assigned number for the shift register ID and a cell number that indicates the order in which the flip-flops were originally connected in the shift register structures. The column contains "-/-" for those cells that are not part of a shift register.

report_scan_cells

```
-----------------------------------------------------------------------
        Chain   Group             ShiftReg                            Clock
CellNo  Name    Name    Pathname  ID/CellNo  CellName  ScanOut Clock
Polarity
-----------------------------------------------------------------------
0       chain1  grp1    /ud5      -/-        sff       Q       clk    (+)
1       chain1  grp1    /ud6      -/-        sff       Q       clk    (+)
2       chain1  grp1    /ud4      1/4        dff       Q       clk    (+)
3       chain1  grp1    /ud3      1/3        dff       Q       clk    (+)
4       chain1  grp1    /ud2      1/2        dff       Q       clk    (+)
5       chain1  grp1    /ud1      1/1        sff       QB      clk    (+)
-----------------------------------------------------------------------
```

Example 4

The following dft -scan example uses the -Shift_register_flops switch to print only the shift register flops in the report:

report_scan_cells -shift_register_flops

```
-------------------------------------------------------------------------------
        Chain   Group             ShiftReg                            Clock
CellNo  Name    Name    Pathname  ID/CellNo  CellName  ScanOut  Clock  Polarity
-------------------------------------------------------------------------------
2       chain1  grp1    /ud4      1/4        dff       Q        clk    (+)
3       chain1  grp1    /ud3      1/3        dff       Q        clk    (+)
4       chain1  grp1    /ud2      1/2        dff       Q        clk    (+)
5       chain1  grp1    /ud1      1/1        sff       QB       clk    (+)
-------------------------------------------------------------------------------
```

Example 5

The following example shows use of the -Power_domain switch to report on power domains.

report_scan_cells -power_domain

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
-----------------------------------------------------------------------------------------
CellNo ChainName GroupName Pathname      CellName ScanOut  Clock ClockPolarity  PowerDomain
-----------------------------------------------------------------------------------------
0      chain1    grp1     /I2/df_0001_1    sff      QB     clk       (+)         PD2
1      chain1    grp1     /I2/df_0002_1    sff      QB     clk       (+)         PD2
2      chain1    grp1     /I2/df_0003_1    sff      QB     clk       (+)         PD2
0      chain2    grp1     /I10/df_0003_1   sff       Q     clk       (+)         PD3
1      chain2    grp1     /I10/df_0001_1   sff      QB     clk       (+)         PD3
2      chain2    grp1     /I10/df_0002     sff      QB     clk       (+)         PD3
0      chain3    grp1     /I6              sff      QB     clk       (+)         PD1
-----------------------------------------------------------------------------------------
```

Example 6

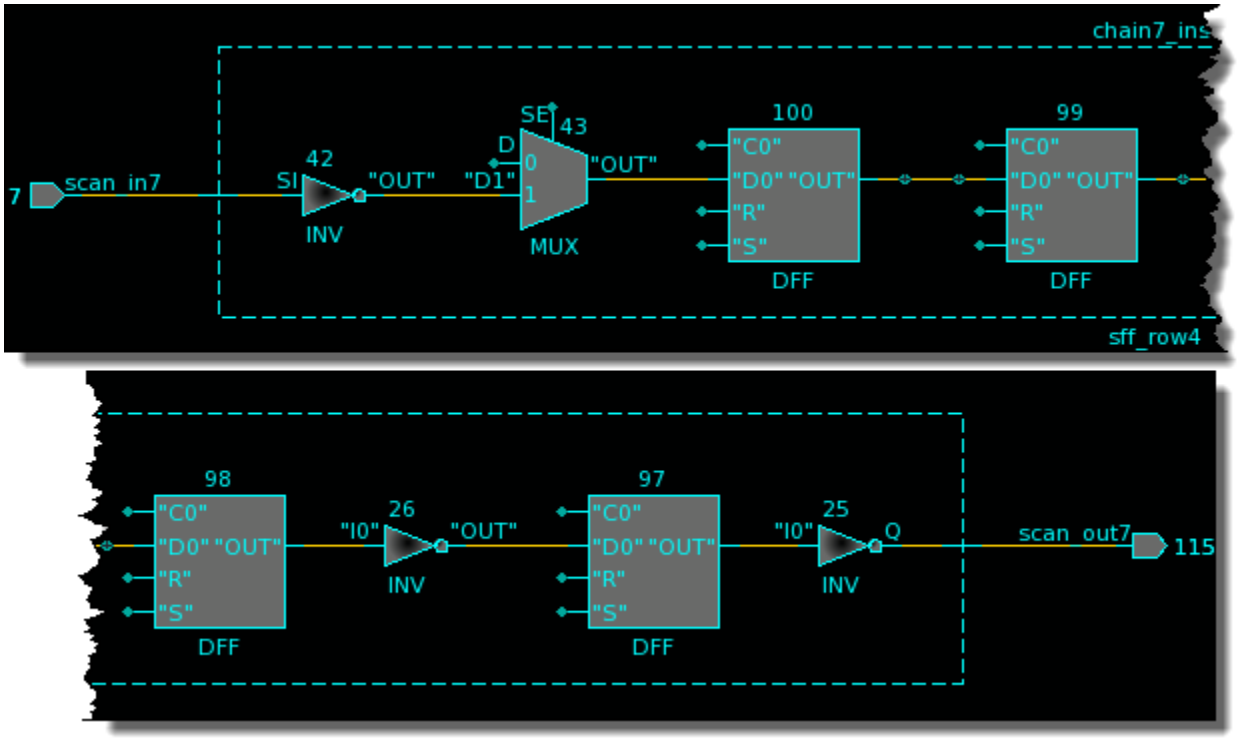
The following dft -scan example shows the additional output that is reported when sub-chains are encountered. Specifically, the starting and ending cell in a sub-chain are listed as a range value in the CellNo column, the clock for the first and last cell in the sub-chain are listed in the Clock column, and the polarity of each clock is listed in the Clock Polarity column.

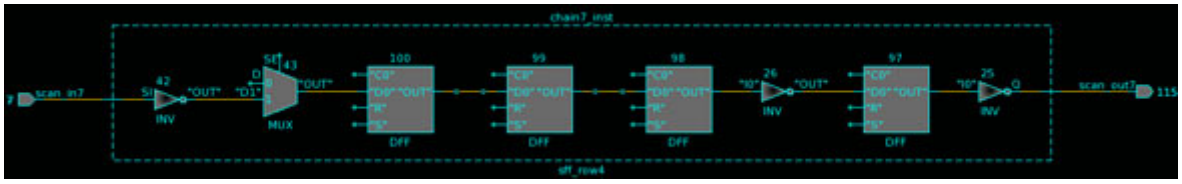
```
-------------------------------------------------------------------------
         Chain    Group                                          Clock
CellNo   Name     Name  Pathname    CellName    ScanOut  Clock   Polarity
-------------------------------------------------------------------------
0       chain1   grp1  /usf2       SCIFTD11S10 SO       sysCLK       (+)
1       chain1   grp1  /usf1       SCIFTD11S10 SO       sysCLK       (+)
0-3     chain2   grp1  /um1        &subchain1  SO    sysCLK,sysCLK (+,+)
4-7     chain2   grp1  /uB1/um1    &subchain1  SO    sysCLK,sysCLK (+,+)
8-9     chain2   grp1  /uC1        &subchain2  so    sysCLK,sysCLK (+,+)
10      chain2   grp1  /uff2       SCIFTD11S10 SO       sysCLK       (+)
11      chain2   grp1  /uB1/uff2   SCIFTD11S10 SO       sysCLK       (+)
0-3     chain3   grp1  /um2        &subchain1  SO    MYTCLK,MYTCLK (+,+)
4-7     chain3   grp1  /uB2/um2    &subchain1  SO    MYTCLK,MYTCLK (+,+)
8-11    chain3   grp1  /uB2/um1    &subchain1  SO    MYTCLK,MYTCLK (+,+)
12-15   chain3   grp1  /uB1/um2    subchain1   SO    MYTCLK,MYTCLK (+,+)
16-17   chain3   grp1  /uC2        &subchain2  so    MYTCLK,MYTCLK (+,+)
18      chain3   grp1  /uff1       SCIFTD11S10 SO       MYTCLK       (+)
19      chain3   grp1  /uB1/uff1   SCIFTD11S10 SO       MYTCLK       (+)
20      chain3   grp1  /uB2/uff1   SCIFTD11S10 SO       MYTCLK       (+)
21      chain3   grp1  /uB2/uff2   SCIFTD11S10 SO       MYTCLK       (+)
-------------------------------------------------------------------------
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 7

The following example shows how the inversion flags appear in cases where the library input or output pins are not present. Consider the following scan chain:





The fourth column of the output of the report_scan_cells shows the inverters in relation to the scan cells in the chain. The inversion flags in this column that are used are shown in green. Observe that all of the unused inversion flags (the ones not in green) have a value of “F”.

report_scan_cells chain7 -display

```
cell#   chain    memory_type      inv   gate#    ...     (ext. pin names)
-----  ------  ---------------  ----  -------   ...     ----------------
0      chain7  MASTER   (FF-LE)  FFFT       97   ...     (-,Q)
1      chain7  MASTER   (FF-LE)  TFFF       98   ...     (-,-)
2      chain7  MASTER   (FF-LE)  TFFF       99   ...     (-,-)
3      chain7  MASTER   (FF-LE)  FFTF      100   ...     (SI,-)
```

Cell 0 is the rightmost cell in the chain. It is connected to a library cell output pin but no library cell input pin. According to Table 7-6 on page 2074 from Example 1, this means that the first flag corresponds to the connection from the scan chain input to the master cell. This is inverted twice, by inverters 42 and 26, so there is no inversion present and this is “F”. The second flag

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

corresponds to the connection from the library cell output pin to the scan chain output pin. Again, there is no inversion present, so this is also “F”. The third flag contains an unused value because there is no library cell input pin for this case. Finally, the fourth flag corresponds to the connection from the cell to the library cell output pin. This is inverted by inverter 25, so this is “T”.

Similarly, cell 1 is not connected to a library cell input or output pin. Therefore, only the values flags 1 and 2 are used. The input to cell 1 is inverted once by inverter 42, so the first flag is “T”. The output from cell 1 is inverted twice by inverters 26 and 25, so there is no overall inversion, and the second flag is “F”.

The remaining cells are similarly notated in the report.

Related Topics

add_scan_chains

add_scan_groups

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_scan_chains

Context: dft -scan, dft -test_points -no_rtl, dft (with no sub-context)

Mode: all

Displays a report on all the current scan chains.

Usage

patterns -scan Usage

report_scan_chains [-All_blocks] [-SUbchains [-Verbose]] [-Inversion] [-INVAlid_scan_pins] [{> | >>} *file_pathname*]

dft -scan Usage

report_scan_chains [{> | >>} *file_pathname*]

Description

The report shows the following information for each scan chain:

- Name of the scan chain
- Name of the scan chain group
- Scan chain input and output pins
- Length of the scan chain
- Names of the scan clocks (but only after you enable the E8 DRC in setup mode)

If the DRC E8 handling is set to anything other than Ignore when leaving Setup system mode, the command additionally lists the clocks specified to be pulsed in the shift procedure in the test procedure file. This enhanced report is only available in non-Setup modes. The clocks reported are likely a superset of the clocks that are actually used to clock the scan cells. For a precise report of the shift clocks, use the report_scan_cells command.

If your design has pre-existing scan chains, the command reports the pre-existing scan chains in separate groups according to their associated scan modes (all/some/no scan mode). Pre-existing scan chains not associated with any mode are displayed in a separate section of the report.

Arguments

- -All_blocks

  Note

  This switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks.

  An optional switch that displays information for all scan chains in every modular EDT block. Without this switch, only information defined for the current EDT block displays. If a current EDT block is not defined, the information for all EDT blocks displays. Use the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

current EDT block using either the add_edt_blocks or set_current_edt_block command. "EDT block" is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the "Modular Compressed ATPG" chapter of the *Tessent TestKompress User's Manual*.

When using the TCD-based automated flow, the tools adds all EDT blocks and does not set the current block. To report the scan chains in this case, you must use the -all_blocks switch or use set_current_edt_block to identify the blocks.

- -SUbchains

  An optional switch that displays information about the subchains of the scan chains, including the number of required serial shifts and the number of scan cells in each subchain. Only the scan chains with a subchain display.

- -Verbose

  An optional switch that displays detailed information about the scan cells in each subchain.

- -Inversion

  An optional switch that displays information about whether there is any net inversion across the entire length of the scan chain, between the scan input pin and scan output pin.
  For EDT only, this switch also displays whether there is any inversion between the decompressor output and the input of the scan chain and whether there is any inversion between the scan chain output and the compactor output. This switch can be used in all modes except Setup mode.

- -INVAlid_scan_pins

  An optional switch that displays only scan chains that have invalid scan pin definitions (input pin or output pin). When using this option, in addition to filtering the chains, the tool also adds "(invalid)" next to every non-existent pin. See "Example 7" on page 2087.

  In some cases when EDT Finder is on, you may only end up with invalid pins because the tool relaxes the pin checks when a chain is added. Having any invalid pins means EDT Finder cannot be turned off; this switch enables you to find invalid pins that prevent EDT Finder from being disabled.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example displays a report of all the scan chains and their scan clocks. Note that because the tool ignores the E8 DRC by default, the command does not display scan clock information.

ATPG> report_scan_chains

chain = chain1 group = grp1 input = /scan_in1 output = /scan_out1 length = 27
chain = chain2 group = grp1 input = /scan_in2 output = /scan_out2 length = 28
chain = chain3 group = grp1 input = /scan_in3 output = /scan_out3 length = 28

However, when you enable the E8 DRC in setup mode, the tool stores and reports scan clock information:

SETUP> set_drc_handling e8 note

SETUP> set_sytstem_mode atpg

ANALYSIS> report_scan_chains

chain = chain1 group = grp1 input = /scan_in1 output = /scan_out1 length = 27
master_clock(s) = /SCK1
chain = chain2 group = grp1 input = /scan_in2 output = /scan_out2 length = 28
master_clock(s) = /SCK2
chain = chain3 group = grp1 input = /scan_in3 output = /scan_out3 length = 28
master_clock(s) = /SCK3

Example 2

The following example displays a report of all the subchains.

report_scan_chains -subchains

```
chain = c1   group = g1   input = /SI0  output = /SO0  length = 11
  subchain-1 from chain "c1" cell 0 to cell 4, length = 5
  1 sub-chain in chain "c1", maximum subchain length within the chain is 5
chain = c3   group = g1   input = /SI2  output = /SO2  length = 7
  sub-chain-1 from chain "c3" cell 1 to cell 5, length = 5
  1 sub-chain in chain "c3", maximum subchain length within the chain is 5

The following 2 library models include sub-chains:
   library_name  max_subchain_length
   ------------  -------------------
   macro_sff1                      5
   reg5                            5

Total 2 sub-chains in 2 library models, which cause 4 extra parallel
shifts.
```

Example 3

The following example displays a report for a design with pre-existing scan chains associated with specific scan modes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_scan_chains

```
========================
Pre-existing Scan Chains
========================

   Pre-existing scan mode 'edt_mode' scan chains:
   ----------------------------------------------
      chain = ext_edt_mode/coreC_i1/
coreC_edt_occ_insertion1_tessent_edt_c1_ext_inst/edt_scan_out[0]   group
= grp1  input = /coreC_i1/
coreC_edt_occ_insertion1_tessent_edt_c1_ext_inst/edt_scan_in[0]  output =
/coreC_i1/coreC_edt_occ_insertion1_tessent_edt_c1_ext_inst/
edt_scan_out[0]  length = 16


      chain = ext_edt_mode/coreC_i1/
coreC_edt_occ_insertion1_tessent_edt_c1_ext_inst/edt_scan_out[1]   group
= grp1  input = /coreC_i1/
coreC_edt_occ_insertion1_tessent_edt_c1_ext_inst/edt_scan_in[1]  output =
/coreC_i1/coreC_edt_occ_insertion1_tessent_edt_c1_ext_inst/
edt_scan_out[1]  length = 16

   Pre-existing scan chains not associated with any scan mode:
   -----------------------------------------------------------
      chain = int_edt_mode/coreC_i1/
coreC_edt_occ_insertion1_tessent_edt_c1_int_inst/edt_scan_out[0]   group
= grp1  input = /coreC_i1/
coreC_edt_occ_insertion1_tessent_edt_c1_int_inst/edt_scan_in[0]  output =
/coreC_i1/coreC_edt_occ_insertion1_tessent_edt_c1_int_inst/
edt_scan_out[0]  length = 0


===============================
Scan Chains Created by the Tool
===============================

   Scan mode 'edt_mode' scan chains:
   ---------------------------------
      chain = edt_mode_chain0   group = grp1  input = /
top_insertion1_tessent_edt_c1_int_inst/edt_scan_in[0]  output = /
top_insertion1_tessent_edt_c1_int_inst/edt_scan_out[0]  length = 4
      chain = edt_mode_chain1   group = grp1  input = /
top_insertion1_tessent_edt_c1_int_inst/edt_scan_in[1]  output = /
top_insertion1_tessent_edt_c1_int_inst/edt_scan_out[1]  length = 4
```

Example 4

The following example indicates there is net inversion across the entire length of the scan chain, there is no inversion between the decompressor output and the input of the scan chain, and there is inversion between the scan chain output and the compactor output.

report_scan_chains -inversion

```
chain = chain1 group = grp1 input = /cpu_i/scan_in1 output =
/cpu_i/scan_out1 length = 152 inversion = Yes decompressor_inversion = No
compactor_inversion = Yes
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 5

The following example shows reporting when power domains are present in the design:

report_scan_chains

```
Power domain 'PD2' chains:
---------------------------------------------------------------------------
chain = chain1  group = grp1   input = /scan_in1  output = /scan_out1
  length = 3  scan_enable = /scan_en  clock = /clk

Power domain 'PD3' chains:
---------------------------------------------------------------------------
chain = chain2   group = grp1   input = /scan_in2  output = /so2 length = 3
  scan_enable = /scan_en  clock = /clk

Power domain 'PD1' chains:
---------------------------------------------------------------------------
chain = chain3   group = grp1   input = /scan_in3  output = /scan_out2
  length = 1  scan_enable = /scan_en  clock = /clk
```

Example 6

The following example displays a detailed report of the scan cells in the subchains.

report_scan_chains -sub -verbose

```
chain = c1   group = g1  input = /SI0  output = /SO0  length = 11
  sub-chain-1 from chain "c1" cell 0 to cell 4, length = 5
----- ----- ------------- ----- ------------ ------------- --------------
cell# chain memory_type   gate# library_name instance_name (ext.pinnames)
----- ----- ------------- ----- ------------ ------------- --------------
0     c1    MASTER (FF-LE)  147 macro_sff1   /macro1  ""   (-,Q)
1     c1    MASTER (FF-LE)  148 macro_sff1   /macro1  ""   (-,-)
2     c1    MASTER (FF-LE)  149 macro_sff1   /macro1  ""   (-,-)
3     c1    MASTER (FF-LE)  150 macro_sff1   /macro1  ""   (-,-)
4     c1    MASTER (FF-LE)  151 macro_sff1   /macro1  ""   (SI,-)
1 sub-chain in chain "c1", maximum sub-chain length within the chain is 5
chain = c3   group = g1  input = /SI2  output = /SO2  length = 7
  sub-chain-1 from chain "c3" cell 1 to cell 5, length = 5
----- ----- ------------- ----- ------------ ------------- --------------
cell# chain memory_type   gate# library_name instance_name (ext.pinnames)
----- ----- ------------- ----- ------------ ------------- --------------
1     c3    MASTER (FF-LE)  164 reg5         /macro5/reg0 s_5   (-,SO)
2     c3    MASTER (FF-LE)  165 reg5         /macro5/reg0 s_4   (-,-)
3     c3    MASTER (FF-LE)  166 reg5         /macro5/reg0 s_3   (-,-)
4     c3    MASTER (FF-LE)  167 reg5         /macro5/reg0 s_2   (-,-)
5     c3    MASTER (FF-LE)  168 reg5         /macro5/reg0 x_1   (SI,-)
1 sub-chain in chain "c3", maximum sub-chain length within the chain is 5

The following 2 library models include sub-chains:
   library_name  max_subchain_length
   ------------  -------------------
   macro_sff1                      5
   reg5                            5

Total 2 sub-chains in 2 library models, which cause 4 extra parallel
shifts.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 7

In the following example, the -INVAlid_scan_pins switch is used to display scan chains that have invalid scan pin definitions:

SETUP> report_scan_chains -all_blocks -invalid_scan_pins

```
EDT block = 'm1_28x16' chain = m1_chain5 group = grp1 input =
 'invalid_scanin1' (invalid) output = 'invalid_scanout1' (invalid) length
 = unknownEDT block = 'm2_4x32' chain = m2_chain1 group = grp1
input = 'invalid_scanin2' (invalid)
output = '/core2_4x32/retimetest_i/edt_so1' length = unknownEDT
block = 'm2_4x32' chain = m2_chain2 group = grp1
input = 'invalid_scanin3' (invalid)
output = '/core2_4x32/retimetest_i/edt_so2' length = unknown
```

Related Topics

add_scan_chains

delete_scan_chains

report_scan_cells

set_parallel_load_subchains

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_scan_elements

Context: dft -scan, dft -test_points -no_rtl, dft (with no sub-context)

Mode: all modes

Reports the scan elements that are created by the tool for scan insertion purposes.

Usage

report_scan_elements [*object_spec*] [-columns *column_names*]
[-add_columns *column_names*]

Description

Use this command to report the scan elements that are created by the tool for scan insertion purposes.

The report is printed in a table format where each column represents an attribute specified by the attribute_name list. Only segment and leaf_cell type of scan_segments are reported unless specified by the object_spec argument. Use the -columns switch to specify which columns to show and the order in which they appear. The "name" column is mandatory and is always shown. You can add additional columns after the default columns (or after the columns specified with the -columns switch) using the -add_columns switch.

Column names can be any of the following:

- Special Columns
  - si_so_clocks — the si,so clock domains. Each clock domain is preceded by +/- to indicate pos_edge/neg_edge accordingly.
- Scan Element Attribute Columns

  All scan element built-in attributes listed in the "Scan Element Object Type" on page 3813"Built in attributes on Scan Element (SE) Object Type" on page 2.
- Module Attribute Columns

  All module built-in attributes listed in "Module" on page 3654
- Instance Attribute Columns

  All instance built-in attributes listed in "Instance" on page 3668

Arguments

- *object_spec*

  An optional value that specifies a Tcl list of one or more object names or a collection of one or more scan element objects. Only existing scan_segment objects are shown.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -columns *column_names*

  An optional list of attributes or connector values to display for each scan element specified, including composite attributes such as “si_so_clocks”.

- -add_columns *column_names*

  An optional list of columns to append after the columns specified by the -columns switch (or after the default columns).

Examples

Example 1

The following example shows the default columns printed when the command is issued with no arguments.

report_scan_elements

```
=================================================================
name         type       length  si_so_clocks  state     is_non_scannable_reason
-----------  ---------  ------  ------------  -------   ----------------------
sch1/f5/SO1  segment    2       -clk1,-clk2   usable    --
sch2/f5/SO2  segment    2       +clk1,+clk3   usable    --
sch3/f6/so   segment    2       -clk2,+clk5   usable    --
/f           leaf_cell  1       +test_clk     ignored   inferred_from_drc
/f2          leaf_cell  1       +test_clk     ignored   user_specified
/f3          leaf_cell  2       -clk1,-clk1   usable    --
```

Example 2

The following example includes user-specified columns.

report_scan_elements [get_scan_elements -state unusable] -columns {type id state is_valid}

```
=================================================
name                 type            id  state     is_valid
-------------------  --------------  --  --------  --------
shift_register#/f4   shift_register  10  unusable  true
chain0               chain           12  unusable  true
inferred#bucket_0_1  inferred        13  unusable  true
```

Example 3

The following example adds additional columns to the default output using the -add_columns switch.

report_scan_elements -add_columns {type id used}

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
==================================================================================
name        type      length  si_so_clocks state   is_non_scannable_reason   id   used
----------- --------- ------  ------------ ------- -----------------------   --   ----
sch1/f5/SO1 segment   2     -clk1,-clk2    usable    --                        15  true
sch2/f5/SO2 segment   2      +clk1,+clk3   usable    --                        16 true
sch3/f6/so  segment   2      -clk2,+clk5   usable    --                        17 true
/f          leaf_cell 1      +test_clk     ignored   inferred_from_drc         19 false
/f2         leaf_cell 1      +test_clk     ignored   user_specified            20 false
/f3         leaf_cell 2      -clk1,-clk1   usable    --                        21 false
```

Example 4

The following example illustrates how you can define the columns of the report using a Tcl variable.

set cord { type state is_non_scannable_reason name}
report_scan_elements -column $cord [get_scan_element -state ignored]

```
=====================================================================================
type       state    is_non_scannable_reason  name
---------  -------  -----------------------  --------------------------------------------
leaf_cell  ignored  missing_scan_model     /tessent_persistent_cell_shift_capture_clock
leaf_cell  ignored  missing_scan_model     /tessent_persistent_cell_edt_clock
leaf_cell  ignored  imported_from_icl      /chip_top_gate_tessent_tap_main_inst/
leaf_cell  ignored  imported_from_icl      /chip_top_gate_tessent_tap_main_inst/instr_reg_2_
leaf_cell  ignored  imported_from_icl      /chip_top_gate_tessent_tap_main_inst/instr_reg_1_
```

Related Topics

add_scan_segments

delete_scan_elements

get_scan_elements

report_scan_segments

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_scan_enable

Context: dft -scan, dft -test_points

Mode: analysis, insertion

Reports on scan_enable signals and associated scan chains.

Usage

report_scan_enable [-scan_mode *scan_mode*] [{> | >>} *file_pathname*]

Description

The following details are reported for each scan enable signal:

- Primary input port (top-level scan enable port)
- Internal connection node (internal instance pin designated as the scan enable signal driver)
- Associated scan chain(s)

Arguments

- -scan_mode *scan_mode*

  An optional switch and string pair that specifies the scan mode for which you want to tool to report information.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of file_pathname.

Examples

Example 1

The following example reports the scan_enable details for all modes.

report_scan_enable

```
Scan mode 'int_mode' scan enables:
-------------------------------------------------------------
Primary Input        Internal Connection Node  Scan Chain
-------------------------------------------------------------
/spi_top/scan_en_out --                        int_mode_chain0
                                               int_mode_chain1
                                               int_mode_chain2
                                               int_mode_chain3
                                               int_mode_chain4
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Scan mode 'ext_mode' scan enables:
-------------------------------------------------------------
Primary Input        Internal Connection Node  Scan Chain
-------------------------------------------------------------
/spi_top/scan_en_out --                        ext_mode_chain0
```

Example 2

The following example reports the scan_enable details for int_mode only:

report_scan_enable -scan_mode int_mode

```
Scan mode 'int_mode' scan enables:
-------------------------------------------------------------
Primary Input        Internal Connection Node  Scan Chain
-------------------------------------------------------------
/spi_top/scan_en_out --                        int_mode_chain0
                                               int_mode_chain1
                                               int_mode_chain2
                                               int_mode_chain3
                                               int_mode_chain4
```

Related Topics

set_scan_enable

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_scan_groups

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays a report on all the current scan chain groups.

Usage

report_scan_groups [{> | >>} *file_pathname*]

Description

The report_scan_groups command provides the following information in a report for each scan chain group:

- Name of the scan chain group
- Number of scan chains within the scan chain group
- Number of shifts
- Name of the test procedure file, which contains the information for controlling the scan chains in the reported scan chain group

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of file_pathname.

Examples

The following example displays a report of all the scan groups:

```
add_scan_groups group1 scanfile
add_scan_groups group2 scanfile1
report_scan_groups
```

Related Topics

add_scan_groups

delete_scan_groups

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_scan_models

Context: dft -scan

Mode: setup, analysis

Displays the sequential scan models currently in the scan model list.

Usage

report_scan_models

Description

The report_scan_models command displays sequential models which you previously added to the scan model list by using the add_scan_instances command.

Arguments

None

Examples

The following example displays all the sequential scan models from the scan model list:

```
add_scan_instances -modules d_flip_flop1 d_flip_flop2
report_scan_models
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_scan_modes

Context: dft -edt, dft -logic_bist, patterns -scan, patterns -scan_diagnosis

Mode: setup

Reports all available scan modes after they are inserted by Tessent Scan.

Usage

report_scan_modes [-core *core_name*]

Description

Reports all available scan modes after they are inserted by Tessent Scan.

Arguments

- -core *core_name*

  An optional switch that specifies the core name for which the scan mode should be reported.

Examples

Example 1

The following example displays the scan modes before a scan mode was imported.

report_scan_modes

```
//  command: report_scan_modes
//            Mode Name  Mode Type  EDT Instruments  OCC Instruments
//      ---------------  ---------  ---------------  ---------------
//      int_edt_mode      internal         1                10
//      ext_mode          external                          2
//      int_single_mode   internal                          10
```

Example 2

The following example displays the scan modes after a scan mode was imported.

report_scan_modes

```
//            Mode Name  Mode Type  EDT Instruments  OCC Instruments
//      ---------------  ---------  ---------------  ---------------
//      int_edt_mode      internal         1                10
//   *  ext_mode          external                          2
//      int_single_mode   internal                          10
//
//   * = last imported scan mode
```

Related Topics

import_scan_mode

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_scan_path

Context: patterns -scan, patterns -scan_diagnosis

Mode: analysis

Traces scan paths and reports the scan cells and combinational logic cells present in the given scan chain.

Usage

report_scan_path [*chain_name*]
{ *cell_id* | -range *start_cell_id end_cell_id* | ([-start *start_cell_id*] [-end *end_cell_id*])}
[-report_past_sci_sco]
[-report_only_memory_elements]
[-report_complex_cell_internals] [-report_layout_data]
[-tag_mbff_internal_pins [-mbff_tag_sci_template *template_string*]
[-mbff_tag_sco_template *template_string*]]
[-skip_mbff_logic]
[-output_file *file_name* [-replace]]
[-add_top_hierarchy *top_level_hierarchy_prefix*]

Description

All sequential memory elements in a scan cell are described by their instance names and library internal names. Combinational logic elements between scan cells are described only by their instance names. However, if a logic element belongs to a multi-bit flip-flop instance, its internal library name is also provided.

The following cell_type names are used for different elements:

- “MASTER”, “COPY”, “SHADOW”, “TLA” — all sequential elements in scan chains (scan cells).
- “CELL” — all library-level logic instances (with cell model) between adjacent scan cells.
- “PRIMITIVE” — all Verilog primitives on the scan path (not contained within any library cell) between adjacent scan cells.
- “MBFF_LOGIC” — used for all logic elements between adjacent scan cells that are part of Multi-Bit Flip-Flop instance

By default, for each scan cell, the tool reports the combinational logic elements existing on the scan path between the given cell and its predecessor. If a group of logic elements forms a complex cell, only this library-level cell instance is reported. Its type is set to “CELL”. The reported gate ID for “CELL” elements is the boundary gate inside the complex logic cell at its output. In addition, the tool also reports any combinational logic elements that may be present on the scan-in and scan-out paths.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Each chain is reported separately in the form of a data table preceded by chain information, as shown in the examples below. Logic elements between scan cells are listed only if there is no ambiguity in the scan path. If there is ambiguity, the logic elements between adjacent cells are listed up to the point where the scan path diverges and then continue again after the point where the path re-converges. See Example 9 for details.

By default, the report_scan_path command provides the following information for each scan cell and combinational logic cell in a specified scan chain:

- Chain cell index number (cell ID), where 0 is the scan cell closest to the scan-out pin.
- Scan chain in which the cell resides.
- The cell type.
- Inversion data.
- Gate index number.
- Shift clock name and inversion status at the clock input.
- Top-level library model cell name.
- Instance name.
- Library instance name, if one is defined.
- Cell input and output pins.

Arguments

- *chain_name*

  Optional string that specifies the name of the chain for which the report should be generated. By default, all chains are reported.

- *cell_id* | -range- *start_cell_id end_cell_id* | ([-start *start_cell_id*] [-end *end_cell_id*])

  Optional choice that enables you to narrow the scope of the report for the specified chain. These options are available when you have specified *chain_name*.

  - *cell_id* — Index of a cell in the specified chain.
  - -range- *start_cell_id end_cell_id* — Closed interval of cell indices in the specified chain.
  - [-start *start_cell_id*] [-end *end_cell_id*] — Indices of a cell in the specified chain from which the command should start reporting and end reporting.

- -report_past_sci_sco

  An option that specifies to include single-port logic elements before the scan in or after the scan out.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -report_only_memory_elements

  An option that specifies to not include inter-scan logic elements between scan cells in the report.

- -report_complex_cell_internals

  For non-sequential cells, a verbose option that specifies to report the internal elements within library-level instances. For these elements, the cell type reported is "CELL". By default, the tool reports scan chain elements between scan cell instances at the library level. See Example 4.

- -report_layout_data

  An option that reports the physical layout coordinates of cells and their scan pins' polygons if an LDB is open. See Example 2. For pin polygons, the layers are reported in layout stack order.

  Note

  When you have a chip-mapped core-level LDBs, prior to using this switch, specify the set_layout_core_instance command. The resulting scan path report displays chip-level instance names, clock paths, and layout coordinates for the specified core. See Example 8.

- -tag_mbff_internal_pins

  An option that turns on custom tagging of internal pins for multi-bit flip-flops in the format (__sci_*N*__, __sco_*N*__), where *N* is the index of a master element within the multi-bit flip-flop, starting with 0. See Example 5 for details. For additional information, refer to "Multi-Bit Flip-Flop Handling" in the *Tessent Diagnosis User's Manual*.

- -mbff_tag_sci_template *template_string*

  An option and string pair that specifies to override the default internal pin tag naming convention for SCIs for multi-bit flip-flops in the format "*template_string N*", where *N* is the index of a master element within the multi-bit flip-flop, starting with 0. See Example 5 for details. For additional information, refer to "Multi-Bit Flip-Flop Handling" in the *Tessent Diagnosis User's Manual*.

- -mbff_tag_sco_template *template_string*

  An option and string pair that specifies to override the default internal pin tag naming convention for SCOs for multi-bit flip-flops in the format "*template_string N*", where *N* is the index of a master element within a multi-bit flip-flop, starting with 0. See Example 5 for details. For additional information, refer to "Multi-Bit Flip-Flop Handling" in the *Tessent Diagnosis User's Manual*.

- -skip_mbff_logic

  An option that specifies to ignore logic elements located within the multi-bit flip-flop. See Example 6 for details.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -output_file *file_name*

  An option that produces a report in JSON-based format and saves it to a compressed file with extension .spr.gz. See Example 7 for details.

- -replace

  An option that specifies to overwrite the existing file of the same name as specified with the -output_file option.

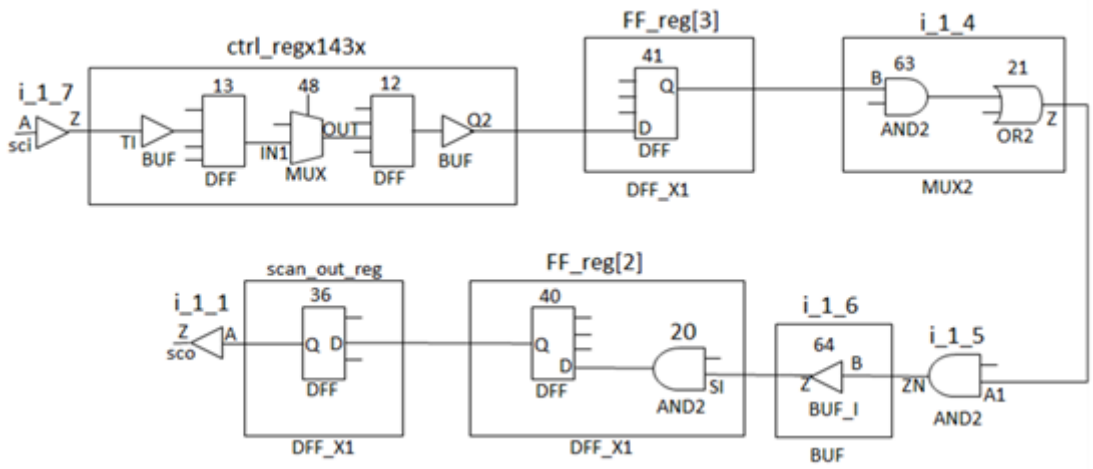
- -add_top_hierarchy *top_level_hierarchy_prefix*

  An option that enables you to report core instance names with respect to the top chip hierarchy when you do not have an LDB. Specify this option to manually add a hierarchy prefix (for example: /top/core_instance1) to the core instance names.

Examples

Examples one through six pertain to Figure 7-6.

Figure 7-6. report_scan_path Example



Example 1

The report provides information for each scan cell starting from scan output (refer to report_scan_cells command for details). The report consists of list of sequential memory elements belonging to each scan cell in the scan path (starting from scan out) and the list of combinational logic elements occurring between adjacent scan cells. Logic elements residing inside scan cells are not reported unless the scan cell is inside a multi-bit flip-flop instance.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
> report_scan_path
//  CHAIN NAME: 'chain1'
//  CHAIN LENGTH: 4
//  SCAN INPUT: '/top/scan_in[0]'
//  SCAN OUTPUT: '/top/scan_out[0]'
cell# chain   cell_type  inv   gate#  shift_clock       inv  cell_name  i
nstance_name  \
lib_internal_pathname  pin_names
---------------------------------------------------------------------------
-----------------
--            CELL        --    2       --              --
BUF         /cpu_top/i_1_1                                  (A,Z)
0      chain1  MASTER      TFFF  36      /shift_capture_clock  F
DFF_X1     /cpu_top/scan_out_reg   mlc_dff                  (D,Q)
--            CELL        --    64      --                    --
BUF /cpu_top/i_1_6                                          (A,Z)
--            AND         --    91      --                    --
AND2       /cpu_top/i_1_5                                   (A1,ZN)
--            CELL        --    21      --                    --
MUX2       /cpu_top/i_1_4                                   (B,Z)
1      chain1  MASTER      TFFF  41      /shift_capture_clock   F
DFF_X1     /cpu_top/FF_reg[3]      mlc_dff                  (D,Q)
2      chain1  MASTER      FFFF  12      /clk_sys               T
HS55       /cpu_top/ctrl_regx143x  dff2                     (-,Q2)
--            MBFF_LOGIC  --    48      --                    --
HS55       /cpu_top/ctrl_regx143x  d_mux2                   (IN1,OUT)
3      chain1  MASTER      FFFF  13      /clk_sys               T
HS55       /cpu_top/ctrl_regx143x  dff1                     (TI,-)
--            CELL        --    100      --                 --
BUF        /cpu_top/i_1_7                                   (A,Z)
```

Example 2

When you specify report_scan_path with the -report_layout_data option, the tool includes two additional columns labeled “cell_coordinates” and “pin_polygons” in the report. The format is:

```
...pin_names   cell_coordinates                            pin_polygons
-------------------------------------------------------------------------------------
...(D,-)       { 5129.0000 18414.0000 5329.5000 18534.0000 } ({route_2 3675.0000 1019.0000
3679.0000 1023.0000},{ route_2 3859.0000 1019.0000 3863.5000 1023.0000 })
...(-,Q)       { 5329.5000 18414.0000 5530.0000 18534.0000 } ({route_2 3474.5000 1019.0000
3478.5000 1023.0000},{ route_2 3659.0000 1019.0000 3663.5000 1023.0000 })
...(A1,ZN)     { 5454.5000 19071.5000 5665.0000 19191.5000 } ({route_2 3339.5000 361.5000
3343.5000 365.5000},{ route_2 3524.000 361.5000 3528.0000 365.5000 })
...(B,Z)       { 5649.0000 19280.0000 5849.5000 19400.0000 } ({route_2 3338.5000 153.0000
3342.5000 157.0000},{ route_2 3154.0000 153.0000 3158.5000 157.0000 })
...
```

The following modified example shows how the tool report displays when there is no polygon data for a pin (“-”) and also when there are polygons on different layers for a pin.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
...pin_names   cell_coordinates                              pin_polygons
-------------------------------------------------------------------------------------
...(D,-)       { 5129.0000 18414.0000 5329.5000 18534.0000 } ({route_2 3675.0000 1019.0000
3679.0000 1023.0000},{ route_2 3859.0000 1019.0000 3863.5000 1023.0000 })
...(-,Q)       { 5329.5000 18414.0000 5530.0000 18534.0000 } (-,{ route_2 3659.0000
1019.0000 3663.5000 1023.0000 })
...(A1,ZN)     { 5454.5000 19071.5000 5665.0000 19191.5000 } ({route_2 3339.5000 361.5000
3343.5000 365.5000},-)
...(B,Z)       { 5649.0000 19280.0000 5849.5000 19400.0000 } ({route_2 3338.5000 153.0000
3342.5000 157.0000},{ route_2 3154.0000 153.0000 3158.5000 157.0000, metal_2 3154.0000
153.000 3158.5000, 157.000 })
...
```

Example 3

The following example shows a report filtered for cells from 0 to 1 in chain1.

```
> report_scan_path chain1 -start 0 -end 1
//  CHAIN NAME: 'chain1'
//  CHAIN LENGTH: 4
//  SCAN INPUT: '/top/scan_in[0]'
//  SCAN OUTPUT: '/top/scan_out[0]'
cell# chain   cell_type  inv   gate#  shift_clock       inv  cell_name  instance_name    \
lib_internal_pathname  pin_names
-------------------------------------------------------------------------------------------
--            CELL      --    2   --                   --  BUF    /cpu_top/i_1_1     \
                        (A,Z)
0     chain1  MASTER    TFFF  36   /shift_capture_clock F   DFF_X1  /cpu_top/scan_out_reg\
mlc_dff                 (D,Q)
--            CELL      --    64   --                   --  BUF    /cpu_top/i_1_6     \
                        (A,Z)
--            AND       --    91   --                   --  AND2    /cpu_top/i_1_5   \
                        (A1,ZN)
--            CELL      --    21   --                   --  MUX2    /cpu_top/i_1_4   \
                        (B,Z)
1     chain1  MASTER    TFFF  41   /shift_capture_clock F   DFF_X1  /cpu_top/FF_reg[3]  \
mlc_dff                 (D,Q)
```

Example 4

In the following example, specifying -report_complex_cell_internal causes the tool to expand the complex instance /cpu_top/i_1_4 (cell type CELL, cell name MUX2) into its two component logic cells: cell types AND and OR, respectively. In addition, the report displays the internal component BUF within instance /cpu_top/i_1_6.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
> report_scan_path -start 0 -end 1 -report_complex_cell_internal
//  CHAIN NAME: 'chain1'
//  CHAIN LENGTH: 4
//  SCAN INPUT: '/top/scan_in[0]'
//  SCAN OUTPUT: '/top/scan_out[0]'
cell# chain   cell_type  inv   gate#  shift_clock       inv  cell_name  instance_name  \
lib_internal_pathname  pin_names
-------------------------------------------------------------------------------------------
--            CELL      --    2   --                   --  BUF    /cpu_top/i_1_1     \
                       (A,Z)
0     chain1  MASTER    TFFF   40   /shift_capture_clock F   DFF_X1  /cpu_top/FF_reg[2]  \
mlc_dff                (D,-)
              COPY      TFFF   36   /cell_clock          T   DFF_X1  /cpu_top/scan_out_reg \
mlc_dff                (-,Q)
--            BUF      --    64     --                   --  BUFF    /cpu_top/i_1_6     \
                       (B,Z)
--            AND       --     91   --                   --  AND2    /cpu_top/i_1_5   \
                       (A1,ZN)
--            OR        --     21   --                   --  MUX2    /cpu_top/i_1_4    \
                       (-,Z)
--            AND       --     63   --                   --  MUX2    /cpu_top/i-1_4   \
                       (B,-)
1     chain1  MASTER    TFFF   41   /shift_capture_clock F   DFF_X1  /cpu_top/FF_reg[3]  \
mlc_dff                (D,Q)
```

Example 5

The following reports show variations for reporting multi-bit flip-flops.

By default, internal pins for non-boundary master cells in multi-bit flip-flops are reported as blank. For example: (SI,-), (-,-) and (-,Q). The -tag_mbff_internal_pins overrides this behavior so that instance-unique names are generated and assigned as names for internal pins in the format "__sci_*N*__" and "__sco_*N*__".

Additionally, you can customize these names by specifying the -mbff_tag_sci_template and -mbff_tag_sco_template options.

```
> report_scan_path chain1 -start 2 -end 3 -tag_mbff_internal_pins
//  CHAIN NAME: 'chain1'
//  CHAIN LENGTH: 4
//  SCAN INPUT: '/top/scan_in[0]'
//  SCAN OUTPUT: '/top/scan_out[0]'
cell#  chain   cell_type  inv   gate#   shift_clock    inv  cell_name   instance_name \
lib_internal_pathname  pin_names
-------------------------------------------------------------------------------------------
2     chain1   MASTER     FFFF  12      /clk_sys       T    HS55   /cpu_topctrl_regx143x \
dff2                   (__sci_2__,Q2)
--             MBFF_LOGIC --    48      --             --   HS55   /cpu_top/ctrl_regx143x \
d_mux2                 (IN1, OUT)
3     chain1   MASTER     FFFF  13     /clk_sys        T    HS55   /cpu_top/ctrl_regx143x \
dff1                   (T1,__sc0_1__)
--               CELL               100     --                 --    BUF    /
cpu_top/i_1_7\
                                                  (A,Z)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
> report_scan_path chain1 -start 2 -end 3 -tag_mbff_internal_pins -mbff_tag_sci_template
SCI__ -mbff_tag_sco_template SCO__
//  CHAIN NAME: 'chain1'
//  CHAIN LENGTH: 4
//  SCAN INPUT: '/top/scan_in[0]'
//  SCAN OUTPUT: '/top/scan_out[0]'
cell#  chain   cell_type  inv   gate#   shift_clock    inv  cell_name   instance_name \
lib_internal_pathname  pin_names
-----------------------------------------------------------------------------------------
2     chain1   MASTER     FFFF  12      /clk_sys       T    HS55   /cpu_topctrl_regx143x \
dff2                  (SCI__2,Q2)
--            MBFF_LOGIC --    48      --             --   HS55   /cpu_top/ctrl_regx143x \
d_mux2                (IN1,OUT)
3     chain1   MASTER     FFFF  13     /clk_sys        T    HS55   /cpu_top/ctrl_regx143x \
dff1                  (TI,SCO_1)
--               CELL               100     --                 --    BUF    /
cpu_top/i_1_7\
                      (A,Z)
```

Example 6

In the following example, specifying -skip_mbff_logic causes the tool to exclude combinational logic elements within the MBFF and only report the scan cells.

```
> report_scan_path chain1 -start 2 -end 3 -skip_mbff_logic
//  CHAIN NAME: 'chain1'
//  CHAIN LENGTH: 4
//  SCAN INPUT: '/top/scan_in[0]'
//  SCAN OUTPUT: '/top/scan_out[0]'
cell#  chain   cell_type  inv   gate#   shift_clock    inv  cell_name   instance_name \
lib_internal_pathname  pin_names
-----------------------------------------------------------------------------------------
2     chain1   MASTER     FFFF  12      /clk_sys       T    HS55   /cpu_topctrl_regx143x \
dff2                  (__sci_2__,Q2)
3     chain1   MASTER     FFFF  13     /clk_sys        T    HS55   /cpu_top/ctrl_regx143x \
dff1                  (TI,__sco_1__)
--               CELL               100     --                 --    BUF    /
cpu_top/i_1_7\
                                                     (A,Z)
```

Example 7

In the following example, specifying -output_file causes the tool to generate the scan path report in JSON format.

```
> report_scan_path chain1 -output_file report
//  Generating scan path report
//  Scan path report file 'report.spr.gz' generated for 1 chain(s)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{
  "version": 1,                    // Version of file structure
  "source" : {                     // Design and LDB sources
    "design": "data/design.flat.gz", "design_md5" : "6570f9dec801b85f013e6076e4e624f6",
    "ldb": "data/design.ldb/layout.ladb", "ldb_md5" : "b09c72503ebee0699fcf42237198a27f"
  },
  "descriptors": { ... },          // Description of data as a key (see example that follows)
  "metadata": { }                  // Reserved field
  "chains":                        // Start data for scan chain
  [
    {
      "cells": [                   // Start scan cell data
        {
          "cell_id": 0,
          "memory_elements": [
            { 40, "MASTER", "TFFF", "/shift_capture_clock", "F", "DFF_X1", "/cpu_top/
FF_reg[2]", "mlc_dff", ["D", []], ["", []], [] },
            { 36, "COPY", "TFFF", "/cell_clock", "T", "DFF_X1", "/cpu_top/scan_out_reg",
"mlc_dff", ["", []], ["Q", []], [] }
          ],
          "logic_elements": [
            { 64, "CELL", "BUF", "/cpu_top/i_I_6", "", ["B", []], ["Z", []], [] },
            { 91, "AND", "AND2", "/cpu_top/i_1_5", "", ["A1", []], ["ZN", []], [] },
            { 21, "CELL", "MUX2", "/cpu_top/i_1_4", "", ["B", []], ["Z", []], [] }
          ]
        },
        {
          "cell_id": 1,
          "memory_elements": [
            { 41, "MASTER", "TFFF", "/shift_capture_clock", "F", "DFF_X1", "/cpu_top/
FF_reg[3]", "mlc_dff", ["D", []], ["Q", []], [] }
          ],
          "logic_elements": []
        },
        {
          "cell_id": 2,
          "memory_elements": [
            { 12, "MASTER", "FFFF", "/clk_sys", "T", "HS55", "/cpu_top/ctrl_regx143x",
"dff2", ["", []], ["Q2",[]], [] }
          ],
          "logic_elements": [
            {48, "MBFF_LOGIC", "HS55", "/cpu_top/ctrl_regx143x", "d_mux2", ["", []], ["",
[]], [] }
          ]
        },
        {
          "cell_id": 3,
          "memory_elements": [
            { 13, "MASTER", "FFFF", "/clk_sys", "T", "HS55", "/cpu_top/ctrl_regx143x",
"dff1", ["TI", []], ["", []], [] }
          ],
          "logic_elements": [
            { 2, "CELL", "BUF", "/cpu_top/i_1_1", "", ["A", []], ["Z", []], [] }
          ]
        }
      ]
      "group": "group1",
     "length": 4,
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
        "name": "chain1",
        "scan_input": "/top/scan_in[0]",
        "scan_output": "/top/scan_out[0]",
        "scan_output_inv": "F",
        "scan_output_logic_elements": [
           [ 100, "CELL", "BUF", "/cpu_top/i_1_1", "", [ "A",  [] ], [ "Z", [] ],  [] ]
        ]
      }
    ]
  }
```

Each chain in the report contains an ordered list of scan cells characterized by three parameters: *cell_id*, ordered list of memory elements and ordered list of logic elements (interconnect elements between the current and next scan cell). Order of all elements reflect their appearance in scan chain (tracing from scan chain output to scan chain input). The list of memory elements always contains at least the “MASTER” element. The list of logic elements can contain no elements.

The descriptors section of the report provides a key to help you read the JSON report. For example:

```
"descriptors: {
   "logic_element": [
      { "gate_id": "integer" },
      { "cell_type": "string" },
      { "cell_name": "string" },
      { "instance_name": "string" },
      { "input_pin": [
          { "layer_name": "string" },
          { "layout_polygons": [ [ "string", [ "float" ] ] ] }
        ]
      },
      { "output_pin": [
          { "layer_name": "string" },
          { "layout_polygons": [ [ "string", [ "float" ] ] ] }
        ]
      },
      { "cell_coordinates": [ "float" ] }
   ],
   "memory_element": [
      { "gate_id": "integer" },
      { "cell_type": "string" },
      { "inv": "string" },
      { "shift_clock": "string" }
      { "shift_clock_inv": "sstring" },
      { "cell_name": "string" },
      { "instance_name": "string" },
      { "lib_internal_pathname": "string" },
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
        { "input_pin": [
            { "layer_name": "string" },
            { "layout_polygons": [ [ "string", [ "float" ] ] ] }
          ]
        },
        { "output_pin": [
            { "layer_name": "string" },
            { "layout_polygons": [ [ "string", [ "float" ] ] ] }
          ]
        },
        { "cell_coordinates": [ "float" ] }
    ]
},
```

The layout_polygons and cell_coordinates fields are empty unless the LDB is open and you specified the -report_layout_data option. For example, for an input pin:

```
        [ "A", [                                        // input pin
            [ "metal1", [                               // layer name
                         341.6800000000001,             // left
                         214.7249999999999,             // bottom
                         341.81,                        // right
                         214.9000000000001,             // top
                        ]
             ]
        ]
    ],
    [                                                   // cell coordinates
      341.430000000001,                                 // left
      214.199999999999,                                 // bottom
      342.759999999999,                                 // right
      215.599999999999,                                 // top
    ]
```

Example 8

This example applies to chip-mapped, core-level LDBs and does not pertain to the figure shown previously. Due to space considerations, the report examples are truncated to only display the pertinent columns.

When you have a chip-mapped core-level LDB, specify the set_layout_core_instance command prior to the report_scan_path to display chip-level instance names, clock paths, and layout coordinates for a particular core. The following example shows a snippet of a report before setting the current layout core instance.

```
> report_scan_path chain24 -report_layout_data

cell# chain ... shift_clock ... instance_name ... cell_coordinates   ...
-------------------------------------------------------------------------------------
0   chain24 ... /CLK2       ... /cpu_i/ix276  ... {3668.5000 962.0000 3869.0000 1082.0000}
1   chain24 ... /CLK2       ... /cpu_i/ix286  ... {3468.0000 962.0000 3668.5000 1082.0000}
...
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

After opening a chip-mapped core-level LDB, you can report the layout core information, and choose one of the displayed instances by using the set_layout_core_instance command.

```
//  command: open layout chip_mapped_ldb/core_flat.ladb -chip_design_name  top_wrapper
//  Note: Loaded layout chip design 'top_wrapper' with 8 instances of core 'cpu_top'
//  command: report_layout_core_information
//  Top design name: 'top_wrapper'
//  Instances of 'cpu_top':
//   ---------------------------------------------------
//     Instance Path   PlaceX    PlaceY     Orient
//   ---------------------------------------------------
//     cpu_top_core1   2000      2000       N
//     cpu_top_core2   9500000   2000       W
//     cpu_top_core3   2000      9500000    S
//     cpu_top_core4   9500000   9500000    E
//     cpu_top_core5   2000      20002000   FN
//     cpu_top_core6   9500000   20002000   FW
//     cpu_top_core7   2000      29500000   FS
//     cpu_top_core8   9500000   29500000   FE
//   ---------------------------------------------------
//  command: set_layout_core_instance cpu_top_core3
//  command: get_layout_core_instance
//  Core instance path: cpu_top_core3
```

The following scan path report shows that when you specify a layout core instance, the tool displays chip-level instance names, clock paths, and layout coordinates.

```
> report_scan_path chain24 -report_layout_data

cell# chain ... shift_clock          ...     instance_name             ...   \
cell_coordinates   ...
-------------------------------------------------------------------------------------------
0   chain24 ... /cpu_top_core3/CLK2   ...     /cpu_top_core3/cpu_i/ix276  ...   \
{5129.0000 18414.0000 5329.5000 18534.0000}    ...
1   chain24 ... /cpu_top_core3/CLK2   ...     /cpu_top_core3/cpu_i/ix286  ...   \
{5329.0000 18414.0000 5530.000 18534.0000}     ...
...
```

Example 9

The following example shows how the scan path report indicates ambiguities in the scan path. When an ambiguity is detected, the report uses a special separator (**) after the gate with the ambiguous scan input to indicate the point of divergence.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// CHAIN NAME: 'chain12'
// CHAIN LENGTH: 2
// SCAN INPUT: 'scan_in'
// SCAN OUTPUT: 'scan_out'
cell# chain   cell_type  inv   gate#  shift_clock       inv  cell_name  instance_name  \
lib_internal_pathname  pin_names
-----------------------------------------------------------------------------------------
0     chain12  MASTER    FFFF  354065   /EXTCLK         F   ff02     /core/top/reg_42_  \
mlc_dff                (d,q)
--            NAND      --    261935   --              --  osi413   /core/top/uopt_123  \
                       (-,z)
**
--            NOR       --    209461   --              --  nor2    /core/top/nor2_1    \
                       (b,z)
1     chain12  MASTER    FFFF  354108   /EXTCLK         F   ff01    /core/top/d_reg_42_  \
mlc_dff                (d,q)
// ** - unique scan path between two gates cannot be determined (ambiguity)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_scan_polygons

Context: patterns -scan_diagnosis

Mode: setup, analysis

Traces scan chains and reports the cell and output net polygons for inter-scan cell logic located along the traces.

Usage

report_scan_polygons [-chain *chain_name* ...] [-start *cell_number*] [-stop *cell_number*] [-layout_marker *marker_file*] [-all_core_instances]

Description

The report_scan_polygons command traces from scan chain output to scan chain input for the specified chains and cell range. It reports the inter-scan cell logic found along the traces and the logics' bounding boxes. The report includes the logics' layers, critical areas, and polygons.

Scan cell numbering starts at 0 for the scan cell closest to the scan output and advances through the cells to the scan input. You can filter the report to include only specified chains and specified cells.

Refer to "Inter-Scan Cell Polygon Reporting for Chain Diagnosis" in the *Tessent Diagnosis User's Manual* for more information.

Arguments

- -chain *chain_name* ...

  An optional switch with strings that specifies a list of scan chain names as reported by the chain diagnosis report. By default, the tool reports all chains in the design.

- -start *cell_number*

  An optional switch and string pair that specifies to begin tracing inter-scan cell logic at the specified scan cell number. The default, 0, represents the scan cell closest to the scan output pin.

- -stop *cell_number*

  An optional switch and string pair that specifies to stop tracing inter-scan cell logic at the specified scan cell number. The default, -1, represents the scan cell closest to the scan input pin.

- -layout_marker *marker_file*

  An option switch and string pair that specifies to write a report in layout marker format supported by Calibre DESIGNrev.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -all_core_instances

Note

When you have a chip-mapped core-level LDB from hierarchical layout-aware diagnosis, prior to using this switch, you must specify the set_layout_core_instance command to set the current layout core instance for reporting purposes.

Only used in the context of hierarchical layout-aware diagnosis. An optional switch that specifies to generate an inter-scan cell logic report with scanned net pathnames at the chip level. By default, the tool reports the scanned net pathnames at the core level. This switch pertains to hierarchical designs and has no effect on non-hierarchical designs. Specifically, the tool generates the scanned net pathnames for each instance of a given core reported at the chip level. Refer to "Inter-Scan Cell Polygon Reporting for Chain Diagnosis" in the *Tessent Diagnosis User's Manual*.

Examples

Example 1

The following example reports the inter-scan cell logic for suspect 16 in the chain diagnosis report, which is suspect scan cell 333 on scan chain 78:

```
16    65  STUCK(IN+CELL+OUT) 1 /top_dut/io/data_out/SO FFTX1 \       /
top_dut/io/n18 333 chain78
```

Specifically, the following command specifies to trace and report the inter-scan cell logic between scan cell 333 and scan cell 334. The report shows that there is a buffer along the trace from scan cell 333 to scan cell 334.

report_scan_polygons -chain chain78 -start 333 -stop 334

For the scan cells and inter-scan cell logic, the report's header lines list the identifier name, its gate type, library model name, and pin instance for each scan cell and connecting logic gate. For each listed cell and cell logic, it then reports the layer name (for nets) or "cell" (where "cell" is the bounding box of the cell listed in the header line), the critical area (for nets only), and the layout defect bounding box coordinates for the cell and output net.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
333    LA  FD4TQHVTX1 /top_dut/io/data_out_regx0x/dff1
# The following lines list the polygons of the net connected to the
# scan output pin of cell 333
   cell{ 67.76 291.2 73.64 295.12 }
   M1 ca: 0.0378666 { 48.745 336.91 48.975 337.05 }
   M1 ca: 0.0378666 { 67.83 292.905 67.97 293.135 }
   M1 ca: 0.0740101 { 67.83 292.34 67.99 293.27 }
   M1 ca: 0.0289322 { 49.17 337.05 49.4 337.22 }
   M1 ca: 0.0703802 { 48.745 336.91 49.4 337.05 }
   M3 ca: 1.24997 { 48.79 335.51 64.05 335.65 }
   M3 ca: 0.438766 { 63.91 333.55 69.09 333.69 }
   M3 ca: 0.1233 { 67.83 314.79 69.09 314.93 }
   M3 ca: 0.0629425 { 63.865 333.55 64.375 333.69 }
   M3 ca: 0.026492 { 63.865 335.51 64.095 335.93 }
   M3 ca: 0.026492 { 48.745 335.51 48.975 335.93 }
   M3 ca: 0.026492 { 68.905 333.27 69.135 333.69 }
   M3 ca: 0.026492 { 68.905 314.51 69.135 314.93 }
   M3 ca: 0.026492 { 67.785 314.51 68.015 314.93 }
 BUF  BFHVTX1 /top_dut/io/iomuxc/iomuxc_registers/FE_OFC20932_n19
# The following lines list the polygons of the net connected to the
# scan output pin of the buffer, which also connects to the scan
# input pin of cell 333
   cell{ 82.32 173.6 83.44 177.52 }
   M1 ca: 0.1246 { 81.275 176.195 82.46 176.325 }
   M1 ca: 0.0592873 { 71.705 292.39 72.215 292.53 }
   M1 ca: 0.024477 { 81.225 176.19 81.455 176.61 }
   M2 ca: 0.0556996 { 77.91 231.35 78.33 231.49 }
   M2 ca: 0.5965 { 77.91 231.35 78.05 238.49 }
   M2 ca: 0.0404092 { 77.07 240.265 77.21 240.495 }
   M2 ca: 0.0404092 { 78.75 220.105 78.89 220.335 }
   M2 ca: 0.0404092 { 77.63 220.105 77.77 220.335 }
   M2 ca: 0.0629425 { 80.43 185.945 80.57 186.455 }
   VIA2 ca: 0.0117878 { 77.635 272.235 77.765 272.365 }
   VIA2 ca: 0.0117878 { 77.915 240.595 78.045 240.725 }
   VIA2 ca: 0.0117878 { 77.075 240.315 77.205 240.445 }
   VIA2 ca: 0.0117878 { 81.275 185.995 81.405 186.125 }
   M3 ca: 0.100766 { 80.43 186.27 81.41 186.41 }
   M3 ca: 0.0404092 { 77.585 272.23 77.815 272.37 }
   M3 ca: 0.0404092 { 77.865 240.59 78.095 240.73 }
   M3 ca: 0.026492 { 78.145 227.15 78.375 227.57 }
   M3 ca: 0.026492 { 77.585 227.15 77.815 227.57 }
   M3 ca: 0.026492 { 81.225 185.99 81.455 186.41 }
   M3 ca: 0.026492 { 80.385 185.99 80.615 186.41 }
334    LA  FD4TQHVTX1 /top_dut/io/data_out_regx0x/dff1
# The following lines list the polygons of the net connected to the scan
# output pin of cell 334, which also connects to the input of the
# buffer
   cell{ 84.84 173.6 90.72 177.52 }
   M1 ca: 0.1675 { 83.3 175.355 84.98 175.485 }
   M1 ca: 0.0740101 { 84.91 174.74 85.07 175.67 }
   M1 ca: 0.0202047{ 83.095 175.025 83.37 175.535 }
   ...
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example shows the output of the report_scan_polygons command when you have a multi-bit flip-flop (MBFF) as a scan element. The command traces the scan elements inside the MBFF and also shows the combinational elements between scan elements inside the flop.

report_scan_polygons -chain 931 -start 0 -stop 1

```
chain: 931
0    LA  t0hd_ln22_sdfarlcp3t01p00 /u_chip_core_top_u_apss_hm/u_kryo_apss/u_apss/
u_gic_asyncbridge_APC/u_d_c_gic_stream/u_adb400_dnstrm/rd_ptr_sync_reg_FTREG3_44662/
seqinst2
 MUX  t0hd_ln22_sdfarlcp3t01p00 /u_chip_core_top_u_apss_hm/u_kryo_apss/u_apss/
u_gic_asyncbridge_APC/u_d_c_gic_stream/u_adb400_dnstrm/rd_ptr_sync_reg_FTREG3_44662
1    LA  t0hd_ln22_sdfarlcp3t01p00 /u_chip_core_top_u_apss_hm/u_kryo_apss/u_apss/
u_gic_asyncbridge_APC/u_d_c_gic_stream/u_adb400_dnstrm/rd_ptr_sync_reg_FTREG3_44662/
seqinst1
```

Related Topics

get_layout_core_instance

set_layout_core_instance

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_scan_segments

Context: dft -scan, dft -test_points -no_rtl, dft (with no sub-context)

Mode: all modes

Reports the scan segment declarations that are defined either via add_scan_segments command or tcd_scan files.

Usage

report_scan_segments [*object_spec*] [-columns *column_names*]
[-add_columns *column_names*]

Description

Use this command to report the scan segment declarations that are defined either via add_scan_segments command or tcd_scan files.

The report is printed in a table format where columns represent the scan segment specifications, such as length, traceability, pin names on the segment container and so forth. The inversion information of the pins is represented with a tilde (~) in front of the pin names in the table.

Note

Pin columns show the full pin path in the event that the pin path is different than the name path or "./" to indicate the path is the same as the path of the scan element as indicated in the name field.

Column names can be any of the following:

- Special Columns
  - **from_tcd** — the from_tcd scan segment property.
  - **reset_pins** — the active_high_reset pins (preceded by % if it is an unbalanced clock, and ~ if the polarity is inverted).
  - **scan_en_pins** — the scan_enable pins (preceded by % if it is an unbalanced clock, and ~ if the polarity is inverted).
  - **set_pins** — the active_high_set pins (preceded by % if it is an unbalanced clock, and ~ if the polarity is inverted).
  - **si_pin** — the scan_in pins (preceded by % if it is an unbalanced clock, and ~ if the polarity is inverted).
  - **si_so_clock_pins** — the clock_in,clock_out pins (preceded by % if it is an unbalanced clock, and ~ if the polarity is inverted).
  - **si_so_clocks** — the si,so clock domains. Each clock domain is preceded by +/- to indicate pos_edge/neg_edge accordingly.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **ste_pins** — the test_enable pins (preceded by % if it is an unbalanced clock, and ~ if the polarity is inverted).
- **so_pin** — the scan_out pins (preceded by % if it is an unbalanced clock, and ~ if the polarity is inverted).

Note

When the tool identifies a clock as "unbalanced," there is clock gating logic inside the segment container, and the clock tree is most likely not balanced with respect to the pin at the container boundary. The tool uses flip-flops instead of latches when inserting retiming logic. Also, the tool drives the flip-flops from the shift-capture clock.

- Scan Element Attribute Columns

  All scan element built-in attributes listed in the "Built in attributes on Scan Element (SE) Object Type" table in "Scan Element Object Type" on page 3813

- Module Attribute Columns

  All module built-in attributes listed in "Module" on page 3654

- Instance Attribute Columns

  All instance built-in attributes listed in "Instance" on page 3668

Arguments

- *object_spec*

  An optional value that specifies a Tcl list of one or more object names or a collection of one or more scan element objects. Only existing scan_segment objects are shown.

- -columns *column_names*

  An optional list of attributes or connector values to display for each scan element, including composite attributes such as "si_so_clocks".

- -add_columns *column_names*

  An optional list of columns to append after the columns specified by the -columns switch (or after the default columns).

Examples

The following example shows a snippet of the default columns printed when the command is issued with no arguments.

report_scan_segments

```
>report_scan_segments==========================================================================
================
name  length  traceable  state     si_pin   so_pin  si_so_clocks  si_so_clock_pins  ...
----  ------  ---------  -------   ------   ------  ------------  ----------------  ...
so1   5       yes        ignored   ./si1    ./so1   --            ./clk1,./clk1     ...
so2   5       no         usable    ./si2    ./so2   +clk3n,+clk3n ~./clk3n,~./clk3n ...
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

add_scan_segments

delete_scan_elements

get_scan_elements

report_scan_elements

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_scan_volume

Context: dft -edt, patterns -scan

Mode: analysis

Displays the volume of scan data used by the pattern set.

Usage

report_scan_volume [-External] [{> | >>} *file_pathname*]

Description

The report_scan_volume command displays the volume of scan data used by the pattern set, including chain test patterns. The report format depends on the design and type of patterns, as illustrated in the examples. Because the pattern count is not a good measure of EDT compression for different scan configurations, this command gives you a metric that is equally good for EDT and regular ATPG, making it easier to compare their results.

Arguments

- -External

  An optional switch that displays the volume of scan data used by the external pattern set. Prior to using this switch, you must specify an external pattern set using the read_patterns command.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

Example 1

This basic EDT design example and the examples that follow show the volume display for various types of designs and patterns.

$$\text{Scan Volume} = (\#\ \text{Scan Channels}) \times (\#\ \text{Shift Cycles}) \times (\#\ \text{Scan Loads})$$

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// ---------------------------------------------------
// Scan volume report.
// -------------------
//    channels     : 1
//    shift cycles : 371+32
// ---------------------------------------------------
// pattern    # test  # scan                   volume
// type     patterns   loads  (cell loads or unloads)
// -------  --------  ------  -----------------------
// chain test      6       6                     2418
```

Example 2

The following example is a basic EDT design where multiple load patterns trigger an expanded format because of a difference in input and output volume:

$$\text{Input Volume} = (\#\ \text{Scan Channels}) \times (\#\ \text{Shift Cycles}) \times (\#\ \text{Scan Loads})$$

$$\text{Output Volume} = (\#\ \text{Scan Channels}) \times (\#\ \text{Shift Cycles}) \times (\#\ \text{Test Patterns})$$

```
// ---------------------------------------------------------------------
// Scan volume report.
// -------------------
//    channels     : 1
//    shift cycles : 371+32
// ---------------------------------------------------------------------
// pattern          # test  # scan  input volume  output volume  average
// type           patterns   loads  (cell loads) (cell unloads)   volume
// -------------  --------  ------  ------------  -------------  -------
// chain test            6       6          2418           2418     2418
// basic                34      34         13702          13702    13702
// clock_seq           304     304        122512         122512   122512
// multiple_load        12      24          9672           4836     7254
// -------------  --------  ------  ------------  -------------  -------
// total               356     368        148304         143468   145886
```

Example 3

The following example is an EDT design where the number of input channels differs from the number of output channels:

$$\text{Input Volume} = (\#\ \text{Input Channels}) \times (\#\ \text{Shift Cycles}) \times (\#\ \text{Scan Loads})$$

$$\text{Output Volume} = (\#\ \text{Output Channels}) \times (\#\ \text{Shift Cycles}) \times (\#\ \text{Test Patterns})$$

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// ---------------------------------------------------------------------
// Scan volume report.
// -------------------
//    input channels  : 8
//    output channels : 9
//    shift cycles    : 30
// ---------------------------------------------------------------------
// pattern          # test  # scan  input volume  output volume  average
// type           patterns   loads  (cell loads) (cell unloads)   volume
// -------------  --------  ------  ------------  -------------  -------
// chain test            4       4           960           1080     1020
// basic                80      80         19200          21600    20400
// multiple_load        10      30          7200           2700     4950
// -------------  --------  ------  ------------  -------------  -------
// total                94     114         27360          25380    26370
```

Example 4

The following example is a Tessent FastScan or Tessent TestKompress (EDT Off) design; no multiple load patterns:

$$\text{Scan Volume} = (\#\ \text{Scan Chains}) \times (\#\ \text{Shift Cycles}) \times (\#\ \text{Scan Loads})$$

```
// ---------------------------------------------------
// Scan volume report.
// -------------------
//    scan chains  : 10
//    shift cycles : 31
// ---------------------------------------------------
// pattern    # test  # scan                   volume
// type     patterns   loads  (cell loads or unloads)
// -------  --------  ------  -----------------------
// chain_test      1       1                      310
```

Example 5

The following example is a Tessent FastScan or Tessent TestKompress (EDT Off) design with multiple load patterns:

$$\text{Input Volume} = (\#\ \text{Scan Chains}) \times (\#\ \text{Shift Cycles}) \times (\#\ \text{Scan Loads})$$

$$\text{Output Volume} = (\#\ \text{Scan Chains}) \times (\#\ \text{Shift Cycles}) \times (\#\ \text{Test Patterns})$$

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// ---------------------------------------------------------------------
// Scan volume report.
// -------------------
//    scan chains  : 10
//    shift cycles : 31
// ---------------------------------------------------------------------
// pattern          # test  # scan  input volume  output volume  average
// type           patterns   loads  (cell loads) (cell unloads)   volume
// ----------  -----------  ------  ------------  -------------  -------
// chain test            1       1           310            310      310
// basic                42      42         13020          13020    13020
// clock_seq            53      53         16430          16430    16430
// multiple_load         3       6          1860            930     1395
// -------------  --------  ------  ------------  -------------  -------
// total                99     102         31620          30690    31155
```

Related Topics

report_edt_configurations

report_edt_pins

set_edt_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_seq_transparent_procedures

Context: dft -edt, patterns -scan

Mode: analysis

Displays a list of seq_transparent test procedures along with the associated data that you specify.

Usage

report_seq_transparent_procedures [-All | *procedure_name*...] [-CElls] [-Load_disturbs] [-CApture_disturbs] [{> | >>} *file_pathname*]

Description

The report_seq_transparent_procedures command displays a list of the specified seq_transparent test procedures. You can optionally display data associated with each seq_transparent test procedure by specifying the appropriate switches.

If you do not specify any of the data switches, the command does not provide any details about the procedures.

Arguments

- -All

  An optional switch that displays the associated data for all seq_transparent procedures. If you do not specify any other argument with the -All switch, the command simply lists the names of all the currently defined seq_transparent procedures. This is the default.

- *procedure_name*

  An optional repeatable string that specifies the names of the seq_transparent procedures whose data you want to display.

- -CElls

  An optional switch that displays the seq_transparent cells associated with the specified procedures. By default, the tools do not display these cells.

- -Load_disturbs

  An optional switch that displays the scan cells with load disturbs associated with the specified procedures. By default, the tools do not display these cells.

- -CApture_disturbs

  An optional switch that displays the scan cells, primary outputs, seq_transparent cells, and RAMs which had capture disturbs. By default, the tools do not display these items.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example displays all the seq_transparent test procedures, along with the associated scan cells that had load disturbs:

```
add_scan_groups g1 seqproc.g1
add_scan_chains c1 g1 si so
add_clocks 0 clk
set_system_mode analysis
report_seq_transparent_procedures -all -load_disturbs
```

Related Topics

report_gates

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_sequential_fault_depth

Context: dft -edt, patterns -scan

Mode: analysis

Displays estimates of maximum test coverage possible with different sequential depth settings.

Usage

report_sequential_fault_depth [{> | >>} *file_pathname*]

Description

The report_sequential_fault_depth command displays the maximum test coverage the tool estimates is possible if all relevant collapsed faults are detectable. Use this command to quickly assess the upper limits of coverage possible under optimal test conditions for various sequential depths.

The data is displayed in three columns as follows:

- Depth numbers for fifteen successive sequential depths, starting with the current depth setting. The default sequential depth upon invocation of the tool is zero, but you can change it using the -Sequential switch with the set_pattern_type command.
- Number of targeted faults for each depth
- Estimated maximum collapsed test coverage achievable under the assumption that all relevant collapsed faults can be detected. This figure is an upper bound of the coverage you can expect.

Note

Relevant faults include those the tool declared ATPG_untestable (AU) in a previous ATPG run—because the tool retargets these faults when you increase the depth. Accordingly, the tool includes these AU faults in the row corresponding to the current depth+1.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example shows the output of the report_sequential_fault_depth command for a sequential depth of 4:

report_sequential_fault_depth

```
// Structural heuristic analysis of the sequential depth of faults
//
//   Set    Number        Max.
// Depth    of target     possible
//    to       faults        TC[%]
//  this      (coll.)      (coll.)
//     4         1450        89.37
//     5         7002        93.59
//     6         7852        94.42
   ...
//    15        11897        98.34
//    16        12033        98.47
//    17        12099        98.53
//    18        12159        98.59
```

Related Topics

set_pattern_type

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_shift_registers

Context: dft -scan, dft - test_points

Mode: setup, analysis, insertion

Reports the identified shift registers in the design after switching to analysis mode.

Usage

report_shift_registers [-Verbose | -Summary]

Description

The tool tries to preserve the original connections inside the identified shift registers during stitching. Therefore, this command may report the shift registers in the design differently, before and after the execution of the insert_test_logic command.

For each identified shift register, this command reports the following information:

- ID, assigned by the tool
- Length
- First and last flip-flop properties (instance pathname, clock edge and name, library model name) unless the -verbose switch is specified in which case the properties of all flip-flops in the shift registers are reported

Arguments

- -Verbose

  An optional switch that reports all of the flip-flops identified in the shift registers.

- -Summary

  An optional switch that reports a summary text without printing the flip-flops identified in the shift registers.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example shows the output when neither switch is specified:

report_shift_registers

```
-----------------------------------------------------------------
                SequentialCell        Clock          Library
Id    Length    Instance PathName     Edge & Name    ModelName
-----------------------------------------------------------------
[1]   4         /ud1                  + clk        dff    ...
                /ud4                  + clk        dff
[2]   2         /uA/uf9               - clk        dff
                /uA/uf10              - clk        dff

// Number of sequential elements in design: 10
// Number of shift register flops recorded for scan insertion: 6
// => 60% of all sequential elements in design
// Number of shift registers recorded for scan insertion: 2
// Longest shift register has 4 flops.
// Shortest shift register has 2 flops.
// Potential number of nonscan flops to be converted to scan cells: 2
// Potential number of scan cells to be converted to nonscan flops: 0
```

Example 2

The following example shows the output when the -verbose switch is specified:

report_shift_registers -verbose

```
-----------------------------------------------------------------
                SequentialCell     Clock              Library
Id    Length    InstancePathName   Edge & Name        ModelName
-----------------------------------------------------------------
[1]   4         /ud1               + clk              dff
                /ud2               + clk              dff
                /ud3               + clk              dff
                /ud4               + clk              dff
[2]   2         /uA/uf9            - clk              dff
                /uA/uf10           - clk              dff

// Number of sequential elements in design: 10
// Number of shift register flops recorded for scan insertion: 6
// => 60% of all sequential elements in design
// Number of shift registers recorded for scan insertion: 2
// Longest shift register has 4 flops.
// Shortest shift register has 2flops.
// Potential number of nonscan flops to be converted to scan cells: 2
// Potential number of scan cells to be converted to nonscan flops: 0
```

Example 3

The following example shows the output when the -summary switch is specified:

report_shift_registers -summary

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Number of sequential elements in design: 6
// Number of shift register flops recorded for scan insertion: 4
// => 66.67% of all sequential elements in design
// Number of shift registers recorded for scan insertion: 1
// Longest shift register has 4 flops.
// Shortest shift register has 4 flops.
// Potential number of nonscan flops to be converted to scan cells: 1
// Potential number of scan cells to be converted to nonscan flops: 0
```

Related Topics

set_shift_register_identification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_silicon_insight_result

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Reports the summary diagnosis results for failing flops and pseudo failing flops for ATPG.

Usage

report_silicon_insight_result **-atpg_datalog** {**scan_cells** | **patterns** | **full**}

Arguments

- **-atpg_datalog** {**scan_cells** |**patterns** | **full**}

  A required switch and string choice that specifies the type of summary information to return:

  scan_cells—Reports a list of failing cells and pattern counts for the failing cells. This type of report is useful for determining how often each cell fails in all failing patterns.

  patterns—Reports a list of all failing patterns and their failing flop/cell count per failing pattern.

  full—Reports a list of all failing patterns, their failure counts, and a description of each failing cell in the pattern.

Examples

Example 1

The following example displays the format for the scan_cells summary diagnosis results. For failing flops, the report displays the cell name, channel ID, and cell index. For failing_pseudo_flops, the report displays the channel ID and cell index.

```
// Cells summary: Summary report for failing patterns count per cell:
// ---------------------------------------------------------------------------
// |                    CellName |    ChannelId | CellIndex | PatternCount |
// ---------------------------------------------------------------------------
// |                         PIN |   SFRWE_c2_2 |         0 |           80 |
// | my_glue/cpu_i/uPORT/ix1734 | edt_channel3 |        15 |            1 |
// | my_glue/cpu_i/uPORT/ix687  | edt_channel3 |        17 |            1 |
// | my_glue/cpu_i/uPORT/ix1754 | edt_channel3 |        18 |            1 |
// | my_glue/cpu_i/uPORT/ix1794 | edt_channel3 |         1 |            1 |
// | my_glue/cpu_i/uPORT/ix1814 | edt_channel3 |         2 |            1 |
// | my_glue/cpu_i/uPORT/ix1834 | edt_channel3 |         3 |            1 |
// | my_glue/cpu_i/uPORT/ix1854 | edt_channel3 |         4 |            1 |
// | my_glue/cpu_i/uPORT/ix701  | edt_channel3 |        10 |            1 |
// | my_glue/cpu_i/uPORT/ix729  | edt_channel3 |        12 |            1 |
```

Example 2

The following example displays the format for the patterns summary diagnosis results.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Patterns summary: Summary report for failing flop frequency per pattern:
// ------------------------
// | Pattern# | FlopCount |
// ------------------------
// |         0 |          1 |
// |         1 |          4 |
// |         2 |          5 |
// |         3 |          6 |
// |         4 |          1 |
```

Example 3

The following example displays the format for the full summary diagnosis results. For failing flops, the report displays the cell name, channel ID, and cell index. For failing_pseudo_flops, the report displays the channel ID and cell index.

```
// Full report: Full report for failing patterns:
// ------------------------------------------------------------------------------------------
// |Pattern# |                   CellName |    ChannelId | CellIndex |    Chain | CellId |
// ------------------------------------------------------------------------------------------
// |       0 |                        PIN |   SFRWE_c2_2 |         0 |          |        |
// |       1 |                        PIN |   SFRWE_c2_2 |         0 |          |        |
// |       1 | my_glue/cpu_i/uPORT/ix1734 | edt_channel3 |        15 |  chain18 |     15 |
// |       1 | my_glue/cpu_i/uPORT/ix687  | edt_channel3 |        17 |  chain18 |     17 |
// |       1 | my_glue/cpu_i/uPORT/ix1754 | edt_channel3 |        18 |  chain18 |     18 |
// |       2 |                        PIN |   SFRWE_c2_2 |         0 |          |        |
// |       2 | my_glue/cpu_i/uPORT/ix1794 | edt_channel3 |         1 |  chain19 |      1 |
// |       2 | my_glue/cpu_i/uPORT/ix1814 | edt_channel3 |         2 |  chain19 |      2 |
// |       2 | my_glue/cpu_i/uPORT/ix1834 | edt_channel3 |         3 |  chain19 |      3 |
// |       2 | my_glue/cpu_i/uPORT/ix1854 | edt_channel3 |         4 |  chain19 |      4 |
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_simdut_data

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Generates reports about specified design objects within designs simulated by SimDUT.

Usage

report_simdut_data ***object_type***

Arguments

- *object_type*

  A required literal that specifies a design object.

Description

Use this command to introspect nets, registers, and module instances within designs simulated by SimDUT. For more information, refer to "Simulating Desktop, ATE, and ATPG Behavior" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

Examples

Example 1

The following example shows a report for block4_I1, where block4_I1 is an instance of a module.

report_simdut_data SIMDUT_TB.DUT_inst.block4_I1

```
name    | SIMDUT_TB.DUT_inst.block4_I1
-- -----+-------------------------------------------------------------
type    | Module
value   | -
regs    | sel_shiftreg[0:1] sel_nr enB enC enD enE enF selA selB art art2
art3[4:0][3] art4[20]
nets    | tlr si en se ce ue tck so muxA muxB tdr_A_so tdr_B_so tdr_C_so
tdr_D_so tdr_E_so tdr_F_so tdr_G_so wart wart2 wart3[3:0][3] wart4[20]
modules | tdr_A tdr_B tdr_C tdr_D tdr_E tdr_F tdr_G
```

Example 2

The following example shows a report for block4_I1.art3, where art3 is a register array defined as: reg [4:0] art3 [0:2]

report_simdut_data SIMDUT_TB.DUT_inst.block4_I1.art3

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
name     | SIMDUT_TB.DUT_inst.block4_I1.art3
-- -----+-------------------------------------------------------------
type     | Array Reg[3]
range    | [4:0]
value    | [0] xxxxx
         | [1] xxxxx
         | [2] xxxxx
```

For the Type field, the integer inside the square brackets indicates the size of the array. The Range field indicates how specific bits are indexed.

Example 3

The following example shows a report for block4_I1.art2, where art2 is a register array defined as: reg [4:0] art2 = 5’b10110

report_simdut_data SIMDUT_TB.DUT_inst.block4_I1.art2

```
name     | SIMDUT_TB.DUT_inst.block4_I1.art2
-- -----+-------------------------------------------------------------
type     | Reg
range    | [4:0]
value    | 10110
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_simdut_faults

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Reports the SimDUT faults.

Usage

report_simdut_faults

Arguments

None

Examples

The following example reports a list of active stuck-at faults stored in the simulator:

report_simdut_faults

```
//  Net name                                                    | fault type
//  ---------------------------------------------------------------------
//  /SIMDUT_TB/DUT_inst/block1/ram1/\MEM_reg[32][1] /Q | 1
//  /SIMDUT_TB/DUT_inst/block1/ram1/\MEM_reg[32][2] /Q | 0
//  /SIMDUT_TB/DUT_inst/block1/ram1/\MEM_reg[32][4] /Q | 1
```

Related Topics

add_simdut_fault

delete_simdut_fault

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_simulation_contexts

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Lists the available simulation contexts and indicates the current simulation context.

Usage

report_simulation_contexts [-predefined | -user_defined]

Description

The report_simulation_contexts command lists all simulation contexts. Note that the command reports available simulation contexts, some of which are only available after the tool reads a procedure file.

If you issue the command without arguments, the command reports all simulation contexts as follows:

```
Simulation Context Report

Name                   Class         Populated
--------------------   ------------  ---------
  empty_context        User-defined  yes
  stable_after_setup   Predefined    no
  stable_capture       Predefined    no
* stable_load_unload   Predefined    no
  stable_shift         Predefined    no
  tie_value            Predefined    yes

* = current simulation context
```

The Populated column reports with a “yes” only those contexts that have correct simulation values. The simulation values are created in the DRCs.

Arguments

- -predefined

  An optional switch that reports predefined simulation contexts, which are listed in Table 8-6 on page 2434.

- -user_defined

  An optional switch that reports user-defined simulation contexts.

Related Topics

add_simulation_context

add_simulation_forces

copy_simulation_context

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

delete_simulation_contexts
delete_simulation_forces
get_current_simulation_context
get_simulation_context_list
get_simulation_value_list
report_simulation_forces
set_current_simulation_context
simulate_clock_pulses
simulate_forces

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_simulation_forces

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Lists the active forces on the specified gate_pin objects.

Usage

report_simulation_forces [***obj_spec***]

Description

The report_simulation_forces command lists the simulation forces on the specified gate_pin object(s). When you issue this command without arguments, the tool lists all simulation forces of the current simulation context.

Arguments

- ***obj_spec***

  An optional value that specifies one or more gate_pin objects.

Examples

The following example shows the output of the report_simulation_forces command:

```
ANALYSIS> add_simulation_forces [get_gate_pins I1] -value 0
ANALYSIS> add_simulation_forces [get_gate_pins I2] -value Z
ANALYSIS> add_simulation_forces [get_gate_pins reg12/Q] -value 1
ANALYSIS> report_simulation_forces
```

```
Gate Pin    Gate Pin ID   Value
--------   -------------  -----
'Din1[0]'       7.0          0
'Din1[1]'       6.0          Z
'Din1[2]'       5.0          1
```

Related Topics

add_simulation_context

add_simulation_forces

copy_simulation_context

delete_simulation_contexts

delete_simulation_forces

get_simulation_context_list

get_simulation_value_list

report_simulation_contexts

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

simulate_clock_pulses

simulate_forces

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_simulation_library_sources

Context: unspecified, all contexts

Mode: all modes

Reports the values specified with the set_simulation_library_sources command.

Usage

report_simulation_library_sources

Description

Reports the values specified with the set_simulation_library_sources command.

Arguments

None.

Examples

The following example uses the set_simulation_library_sources to define search paths, and to map a pre-compiled library name to its physical directory. The report_simulation_library_sources command is then used to report them.

```
set_simulation_library_sources \
   -y ../verilog ./../data/customPads \
   -v ../data/extra_lib.v \
   -extensions .v \
   -logical_library_map_list {my_lib ../data}
report_simulation_library_sources
```

```
// Simulation Library Source Paths
// ===============================
// type  path                     file extensions
// ----  ----------------------   ---------------
// dir   '../verilog'             'v v.gz'
// dir   './../data/customPads'   '.v.gz'
// file  '../data/extra_lib.v'
//
// Simulation Library Logical Library Map
// ======================================
//  name    dir path
//  ------  --------
//  my_lib  ../data
```

Related Topics

set_simulation_library_sources

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_ssn_configuration

Context: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: analysis

Reports the current SSN configuration settings.

Usage

report_ssn_configuration

Description

The report_ssn_configuration command displays a report describing the SSN configuration. It provides information about all active datapaths and SSHs, including bus width, bus clock period, shift clock period, and capture clock period.

Note

You may see minor differences between the clock period this command reports and measured period values in simulation. This command reports the smallest generated clock period. Any differences are due to jitter in the clock period caused by packet rotation in the bus and are expected as part of normal operation.

Arguments

None.

Examples

The following example splits the output of the report_ssn_configuration command across two blocks for formatting purposes. The ellipsis (…) indicates how the lines break across each block.

ANALYSIS> report_ssn_configuration

```
----------  ------------------------------------------------------ …
 Datapath                        ICL Instance                      …
                                                                   …
----------  ------------------------------------------------------ …
 1           *** datapath output ***                               …
             rng_3.MT_GET_occ_hybrid_tessent_ssn_scan_host_1_inst  …
             rng_1.MT_GET_occ_hybrid_tessent_ssn_scan_host_1_inst  …
             rng_2.MT_GET_occ_hybrid_tessent_ssn_scan_host_1_inst  …
             *** datapath input ***                                …
----------  ------------------------------------------------------ …
 2           *** datapath output ***                               …
             kekelo2.kekelo_rtl_tessent_ssn_scan_host_1_inst       …
             kekelo1.kekelo_rtl_tessent_ssn_scan_host_1_inst       …
             xtea_3.xtea_rtl_tessent_ssn_scan_host_1_inst          …
             xtea_2.xtea_rtl_tessent_ssn_scan_host_1_inst          …
             xtea_1.xtea_rtl_tessent_ssn_scan_host_1_inst          …
             *** datapath input ***                                …
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
… -----------  ----------------  ----------------  ----------------
…  Bus Width       Bus Clock        Shift Clock      Capture Clock
…                   Period            Period            Period
… -----------  ----------------  ----------------  ----------------
…      6        30ns (33.3MHz)
…      1        5ns (200.0MHz)    135ns (7.4MHz)    135ns (7.4MHz)
…      1        5ns (200.0MHz)    135ns (7.4MHz)    135ns (7.4MHz)
…      1        5ns (200.0MHz)    135ns (7.4MHz)    135ns (7.4MHz)
…      6        30ns (33.3MHz)
… -----------  ----------------  ----------------  ----------------
…      6        30ns (33.3MHz)
…      1        15ns (66.7MHz)    225ns (4.4MHz)    75ns (13.3MHz)
…      1        15ns (66.7MHz)    225ns (4.4MHz)    75ns (13.3MHz)
…      1        15ns (66.7MHz)    150ns (6.7MHz)    75ns (13.3MHz)
…      1        15ns (66.7MHz)    150ns (6.7MHz)    75ns (13.3MHz)
…      1        15ns (66.7MHz)    150ns (6.7MHz)    75ns (13.3MHz)
…      2        30ns (33.3MHz)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_static_dft_signal_settings

Context: dft patterns

Mode: setup, analysis, insertion

Prerequisites: ICL is elaborated

Reports the setting specified or implied with the set_static_dft_signal_values command.

Usage

report_static_dft_signal_settings
[-all | -current_design | -icl_instances *icl_instances* | -instances *instances*]

Description

Reports the settings specified or implied with the set_static_dft_signal_values command. All static DFT signals implemented with IJTAG can be set using the set_static_dft_signal_values command. See the add_dft_signals command for information on how to add the DFT signals. The DFT signals are grouped by each module in which you added the signals and ran the process_dft_specification command. These modules have the tessent_design_level attribute equal to “chip”, “physical_block”, and “sub_block” in their extracted ICL module view.

When setting a static DFT signal within an instance, the other DFT signals without an explicit setting are assigned an implicit value following the logic described in the set_static_dft_signal_values command description section.

Arguments

- -all

  An optional switch that reports the settings that exist on DFT signals found within all ICL instances that have the tessent_design_level attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition. The DFT signals found within the top module are also reported.

  The -all, -current_design, -icl_instances, and -instances switches are mutually exclusive. If unspecified, the -current_design switch is the default.

- -current_design

  An optional switch that reports the settings that exist on DFT signals found within the top module. The top-level module must have the tessent_design_level attribute set to “chip”, “physical_block”, and “sub_block” within the ICL module definition.

  The -all, -current_design, -icl_instances, and -instances switches are mutually exclusive. If unspecified, the -current_design switch is the default.

- -icl_instances *icl_instances*

  An optional switch that reports the settings that exist on DFT signals found within the specified ICL instances. These ICL instances must have the tessent_design_level attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The -all, -current_design, -icl_instances, and -instances switches are mutually exclusive. If unspecified, the -current_design switch is the default.

- -instances *instances*

  An optional switch that reports the settings that exist on DFT signals found within the specified design instances. These design instances must match the value of the "hierarchical_design_instance" attribute found on an ICL instance with the tessent_design_level attribute set to "chip", "physical_block", and "sub_block" within their ICL module definition.

  The -all, -current_design, -icl_instances, and -instances switches are mutually exclusive. If unspecified, the -current_design switch is the default.

Examples

The following example shows the report generated for a block where the set_static_dft_signal_values was explicitly called for the ltest_en DFT signal. The report shows the inferred values on the other signals.

```
// ICL Module      : sub_block_l3


// ---------------  -------------------  ---------  ----------
// DFT Signal Name  Usage                Set value  Set source
// ---------------  -------------------  ---------  ----------
// all_test         global_dft_control       1      Inferred
// ext_ltest_en     logic_test_control       -      -
// int_ltest_en     logic_test_control       -      -
// ltest_en         logic_test_control       1      Explicit
// ext_multi        scan_mode(external)      -      -
// int_edt          scan_mode(internal)      -      -
//
```

Related Topics

add_dft_signals

set_static_dft_signal_values

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_statistics

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Displays a detailed statistics report to the screen.

Usage

report_statistics
[[-DEtailed_analysis [DI] [-Threshold *threshold_number*]]
[-Instance *instance_pathname* | -Model {ALL | *model_name*} |
-CLOCK_domains {ALL | SUMMARY | *clock_pathname...*}] |
-Hierarchy [-FORmat "*format_code ...*"]
[-Sort "*format_code*" [-Ascending | -Descending]]
[-EXClude_unlisted_faults | -Grayboxes_only]
[-LESsthan *percentage*]
[-LEVel *integer*]
[-DETections *integer*]
[-INDent *integer* | [-USe_full_instance_path] [-CSV]]]
[-port_equivalent_faults]
[(> | >>) *file_pathname*]

Description

The tool also reports CPU time that is the cumulative CPU time of the master process since you invoked the run.

When you use the -Instance switch, the tool's statistics report lists the following three groups of information:

- The current number of collapsed or total faults, depending on the current fault mode, in each class. By default, the fault mode is full. The report does not display fault classes with no members.
- The percentage of test coverage, fault coverage, fault coverage loss, and ATPG effectiveness for collapsed or total faults. By default, the fault mode is full, however the tool can report either the collapsed number or full number depending on the current fault mode.
- The total numbers for the following:
  - Total patterns simulated in the preceding fault simulation process. This subgroup may additionally contain total numbers for the following internal patterns sets:

    Basic scan patterns

    Clock_po patterns

    Ram_sequential patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Clock_sequential patterns

- Total patterns currently in the test pattern set
- Total CPU time

If a pattern type has no patterns, the report does not display the count for that type. If all patterns are basic patterns, it does not display any count. And it counts clock_sequential patterns that are also clock_po only as clock_sequential patterns.

When you use the -Hierarchy switch, the statistics report lists the following two groups of information for each reported hierarchical instance:

- Instance leaf name (add -USe_full_instance_path switch to report entire instance path)
- A customizable list of statistics that, by default, consists of the following:
  - Module name
  - Total number of faults
  - Number of unobserved (UO) faults
  - Number of ATPG untestable (AU) faults
  - Test coverage

When you issue the "set_atpg_timing On" command prior to create_patterns, the statistics report also includes delay test coverage. For more information about how the tool calculates delay test coverage, refer to the *Tessent Scan and ATPG User's Manual*.

Arguments

- -DEtailed_analysis [DI] [-Threshold *threshold_number*]

  An optional switch that, by default, reports detailed information for the following fault sub-classes:

  - AU.CC (CELL_CONSTRAINTS)
  - AU.PC (PIN_CONSTRAINTS)
  - AU.TC (TIED_CELLS)
  - UD.AAB (ATPG_ABORT)
  - UD.EAB (EDT_ABORT)
  - UD.UNS (UNSUCCESS)

  For information about any of these fault sub-classes, refer to the discussion about "Fault Sub-Classes" in the *Tessent Scan and ATPG User's Manual*.

  For an example of reporting detailed statistics for AU or DI sub-classes, refer to "Example 2" on page 2149.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DI

An optional literal that enables the tool to report the DI fault sub-classes without performing the detailed analysis for AU faults, which can be time consuming. All of the DI sub-classes are listed under "Fault Sub-Classes" in the *Tessent Scan and ATPG User's Manual*. Note that the tool always performs analysis for all DI fault sub-classes, with the exception of DI.EDT, which you control using the set_fault_subclass_analysis command. For more information about DI.EDT analysis, refer to that command description.

-Threshold *threshold_number*

An optional switch and numerical pair that limits the list of untestable faults displayed. The *threshold_number* is the minimum coverage loss percentage to display. The default threshold is 0.1 percent. Faults having a coverage loss less than the threshold are summarized at the end of the list (see "Example 2" on page 2149).

- -Instance *instance_pathname*

  An optional switch and string pair that reports fault statistics for a specific instance. The instance_pathname is the name of a circuit block whose statistics you want to report. Only fault statistics are affected by this option; pattern count statistics apply to the entire design.

- -Model ALL | *model_name*

  An optional switch and string pair that specifies reporting statistics for all models or a single named model in a Tessent Cell library. When specifying a single model, *model_name* must match a cell model in the Tessent Cell library. This switch is only available for Tessent FastScan and Tessent TestKompress when invoked in model mode (-model). If you use this switch in other than model mode, the tool issues the following error:

  Error: -model reporting only available for -model tool invocations.

- -CLOCK_domains SUMMARY | ALL | *clock_pathname...*

  An optional switch and literal or repeatable string pair that you use for the following:

  - Reporting the summary of clock domain test coverage
  - Specifying a list of clocks to report the coverages for the specified clock domains

  Choose from the following options:

  SUMMARY — A literal that reports a summary of clock domain coverage and produces a Clock Domain Summary section. For each clock domain, the tool reports the percentage of faults in the clock domain with respect to the total fault population, and reports the relevant test coverage numbers for each clock domain.

  ALL — A literal that reports the detailed coverage information for all clock domains.

  clock_pathnames... — A repeatable string specifying a clock domain or clock domains. The tool reports the detailed coverage information for the specified clock domains.

- -Hierarchy

  An optional switch that enables fault statistics to be reported for each hierarchical instance in the design. The tool uses columns for all reported fields. For each reported instance, the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

tool displays the instance name and flags the instance if it is a gray box (GB) or black box (BB). If an instance is a gray box, the tool adds rows reporting statistics of faults for "Instances in netlist" and "Instances not in netlist". By default, the tool lists the following statistics for each reported instance:

- Module name
- Total number of faults
- Number of unobserved (UO) faults
- Number of ATPG untestable (AU) faults
- Test coverage
- Test coverage loss

-FORmat "*format_code…*"

An optional switch and a string of one or more format codes that, together with the -Hierarchy switch, specifies which data to display and the order in which to display it. Use this switch to tailor the content and arrangement of displayed data in hierarchical reports. By default, only the instance column is displayed when using the -format switch. Table 7-7 lists the format code literals this switch recognizes and the information displayed for each.

Note

Because it is not possible to determine the locations of representative faults for a specific level of hierarchy, only uncollapsed faults are reported with the -Hierarchy switch.

Table 7-7. Format Codes

Tessent FastScan Information (uncollapsed)	Format Code	Report Label
Fault coverage	%fc	Fault Coverage
Test coverage	%tc	Test Coverage
Test coverage loss	%tcl	TC Loss
Relevant test coverage	%rtc	Relevant TC
ATPG effectiveness	%ae	ATPG Eff.
Module name	mo	Module
Number of faults	#f	# faults
Fault category	<category>[.<subclass>]	<CAT>[.<SUB>]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The -format option reports statistics for fault subclasses specified using format codes with this format:

```
<fault category>[.<subclass>]
```

For example:

```
report_statistics -hierarchy -format "%tc %rtc AU AU.BB" -level 1
                 Hierarchical Statistics Report
                          Stuck-at Faults
-------------------------------------------------------------------------------
               Instance                 Test Coverage  Relevant TC     AU   AU.BB
-------------------------------------  -------------  -----------  -----  ------
-top-                                         59.12%       59.12%  14465     116
  CHIPJ_MGsg_INT_CD_FLOP_DECODER_INST         79.41%       79.41%      7       0
  DEF                                         89.85%       89.85%    166       0
  LVISION_JTAP_INST                            0.00%        0.00%      0       0
  LVISION_LOGICTEST_INST                       0.00%        0.00%      0       0
  LV_BURST_CLK_CTRL_I1                         0.00%        0.00%      0       0
  LV_BURST_CLK_CTRL_I2                         0.00%        0.00%      0       0
  LV_SE_CTRL_BL5_P_I1                         77.34%       77.34%     29       0
  LV_SE_CTRL_BL5_P_I2                          6.83%       76.83%     19       0
  LV_SHIFT_CLK_CTRL_INST                       0.00%        0.00%      0       0
  TLB_inst                                    57.69%       57.69%   4100     116
```

All format codes, including the fault codes, are case insensitive. For example:

```
report_statistics -hierarchy -format “%fc au Au.UNC Di.sCAn, DI.UNC”
```

When duplicated format codes exist in the format code list, only one displays in the reported statistics.

Information displays in the same order as the format codes in the command. The default for this switch is:

```
-format "mo #f uo au %tc"
```

The results display as shown in the following example:

```
                Hierarchical Statistics Report
                         Stuck-at Faults
-------------------------------------------------------------
Instance    Module      #faults  UO        AU   Test Coverage
----------- ------------ ------- ---- -------- --------------
-top-       my_core      46250   2316      768       93.16%
  u10       m3s018bo      6240   1038       99       81.36%
  u11       m3s015bo      4526    122       60       95.97%
  u12       m3s016bo_3     590      7        8       97.46%
    u1      m3s020bo      2146     87       33       94.40%
```

Note

You must include a space between each format code you specify, and begin and end the sequence of format codes with a double quotation mark (").

-Sort *format code* [-Ascending | -Descending]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

An optional switch and literal pair that enables the tool to sort the results by the column indicated by *format code*. Sorting begins with level 1 instances, proceeds with each level 1 instance's child (level 2), and continues hierarchically until complete. If -sort is not used, the report is sorted by the instance names in ascending alphabetical order.

-Ascending

An optional switch to sort in ascending order.

-Descending

An optional switch to sort in descending order. This is the default sort order.

The -sort option parser supports fault <category>[.<subclass>] format codes, case insensitive format codes, and ignores duplicated format column codes.

-EXClude_unlisted_faults

An optional switch that excludes unlisted faults from the report and calculations. It also omits the "Instances in netlist" and "Instances not in netlist" lines from the report. See also unlisted faults in the report_faults command description.

-Grayboxes_only

An optional switch that together with the -Hierarchy switch, limits the report to graybox instances, not including the top instance. This option automatically enables the "-use_full_instance_path" option.

When you use this switch, the header of the statistics report includes the text "(Grayboxes Only)".

-LESsthan *percentage*

An optional switch and real number pair that, together with the -hierarchy switch, specifies to report only module instances with test coverage less than or equal to the specified percentage. The default percentage is 100.0 and causes all module instances to be reported.

Specifying "-lessthan 90.0", as shown in this example, the tool reports statistics only for module instance u10 because u10 is the only instance with test coverage less than or equal to 90 percent:

```
report_statistics -hierarchy -lessthan 90.0

                  Hierarchical Statistics Report
                          Stuck-at Faults
-------------------------------------------------------------------
  Instance     Module       #faults   UO    AU  Test Coverage
---------      ------------  ------   ----  --  -------------
u10            m3s018bo       6240    1038   9       81.36%
```

-LEVel *integer*

An optional switch and integer pair that, together with the -Hierarchy switch, controls the hierarchy reporting depth. The tool reports all hierarchical levels down to and including the level you specify, counting from the top level, which is 0. If you specify "-level 1", for example, the tool displays data for the top level and the instances one

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

level of hierarchy beneath the top level. By using this switch to restrict reporting to the upper levels of a design's hierarchy, you can reduce the command's processing time on very large designs with deep hierarchy. The maximum permitted value is 1000.

-DETections *integer*

An optional switch and integer pair used with the -Hierarchy switch to represent the number of multiple detections. When specified, the DS fault count, test coverage, relevant coverage, fault coverage, test coverage loss, and ATPG efficiency metrics adjust to reflect the values with detections greater than or equal to the detection number specified.

By default, the tool runs as though you had specified "-detections 1".

-INDent *integer*

An optional switch and integer pair that, together with the -hierarchy switch, specifies the number of additional spaces to indent the information for each subsequent level of reported hierarchy. The default indentation is two spaces. Specifying "-indent 0" removes all indentation. The maximum permitted value is 10.

-USe_full_instance_path

An optional switch used to display the complete instance path. Without it, the report displays the leaf instance name and the instantiation level is shown by indentation.

-CSV

An optional switch that uses a comma as a delimiter between the columns in the report, making the data readily usable within spreadsheet programs. This option automatically enables the "-use_full_instance_path" option.

When you use this switch, the statistics report has no header.

- -port_equivalent_faults

  An optional switch to restrict the UDFM fault statistics reporting to library cell port faults. Use this switch only when loading UDFM files created by Tessent CellModelGen.

  In the uncollapsed fault mode, you can generate UDFM statistics reports equivalent to traditional stuck-at and transition fault statistics reports. Read in one or more UDFM files then use the "add_fault_sites -all" command followed by the "report_statistics -port_equivalent_faults" command.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example displays a statistics report for the top level instance in a design after performing an ATPG run:

set_system_mode analysis
create_patterns
report_statistics

```
                          Statistics Report
                          Transition Faults
---------------------------------------------------------------------------
Fault Classes                                  #faults          #faults
                                               (total)      (total relevant)
-------------------------------------    ------------- ------------------
  FU (full)                                     1446            1424
  -----------------------------------    ------------- ------------------
  DS (det_simulation)                        24 ( 1.66%)       same ( 1.69%)
  DI (det_implication)                      326 (22.54%)       same (22.89%)
  UU (unused)                               242 (16.74%)       same (16.99%)
  AU (atpg_untestable)                      854 (59.06%)        832 (58.43%)
---------------------------------------------------------------------------
Fault Sub-classes
  --------------------------------
  AU (atpg_untestable)
    BB  (black_boxes)                       176 (12.17%)       same (12.36%)
    PC*  (pin_constraints)                   22 ( 1.52%)       deleted
    TC*  (tied_cells)                       120 ( 8.30%)       same ( 8.43%)
    Unclassified                            536 (37.07%)       same (37.64%)
  *Use "report_statistics -detailed_analysis" for details.
---------------------------------------------------------------------------
Coverage
  --------------------------------
  test_coverage                                 29.07%            29.61%
  fault_coverage                                24.20%            24.58%
  atpg_effectiveness                           100.00%           100.00%
---------------------------------------------------------------------------
#test_patterns                                                          4
  #clock_sequential_patterns                                            4
#simulated_patterns                                                    64
CPU_time (secs)                                                       0.5
---------------------------------------------------------------------------
```

Here is how the statistics report would look if you used a “report_statistics -hierarchy -level 1” command:

report_statistics -hierarchy -level 1

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
                          Hierarchical Statistics Report
                                  Stuck-at Faults
-----------------------------------------------------------------------------------------
Instance                              Module                  #faults  UO     AU Test Coverage
--------------------------  ----------------------------   -------  -- ------ ------------
-top-                       CHPJ                              45084   0  14465     59.12%
  CHPJ_MGsg_FLP_DCDR_INST   CHPJ_MGsg_FLP_DCDR                   34   0      7     79.41%
  DEF                       CHPJ_LV_BGROUP_DEF                 1638   0    166     89.85%
  LVSN_JTAP_INST            CHPJ_LVISION_JTAP                     0   0      0      0.00%
  LVSN_LGCTST_INSTCHP       LVSN_LGCTST                           0   0      0      0.00%
  LV_BRST_CLK_CTRL_I1       CHPJ_LV_BRST_CLK_CTRL_BL5_P_LVU67     0   0      0      0.00%
  LV_BRST_CLK_CTRL_I2       CHPJ_LV_BRST_CLK_CTRL_BL5_P           0   0      0      0.00%
  LV_SE_CTRL_BL5_P_I1       CHPJ_LV_SE_CTRL_BL5_P_LVU69         356   0     29     77.34%
  LV_SE_CTRL_BL5_P_I2       CHPJ_LV_SE_CTRL_BL5_P               356   0     19     76.83%
  LV_SHIFT_CLK_CTRL_INST    CHPJ_LV_SHIFT_CLK_CTRL                0   0      0      0.00%
  TLB_inst                  TLB                               42512   0  14100     57.69%
```

Example 2

The following example shows a statistics report with detailed analysis:

report_statistics -detailed_analysis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
             Statistics Report
              Stuck-at Faults
-------------------------------------------
Fault Classes                     #faults
                                  (total)
---------------------------  --------------
  FU (full)                         144
  -------------------------  --------------
  UU (unused)                    14 ( 9.72%)
  RE (redundant)                  2 ( 1.39%)
  AU (atpg_untestable)          128 (88.89%)
-------------------------------------------
Fault Sub-classes
  -------------------------
  AU (atpg_untestable)
    TC   (tied_cells)           124 (86.11%)
      /u4/u2 (38) TX             23 (15.97%)
      /u4/u5/u2 (40) TX          18 (12.50%)
      /u5/u2 (42) TX             18 (12.50%)
      /u2 (36) TX                14 ( 9.72%)
      /u4/u5/u3 (41) TX          14 ( 9.72%)
      /u7 (37) TX                10 ( 6.94%)
      /u4/u3 (39) TX             10 ( 6.94%)
      (Combined tied cells)      17 (11.81%)
    Unclassified                  4 ( 2.78%)
-------------------------------------------
Coverage
  -------------------------
  test_coverage                      0.00%
  fault_coverage                     0.00%
  atpg_effectiveness               100.00%
-------------------------------------------
#test_patterns                           0
#simulated_patterns                      0
CPU_time (secs)                        1.8
-------------------------------------------
```

Use the -Threshold switch if you want to limit the detailed list of untested faults. Use the following command to show the detailed breakout of the DI fault sub-classes in addition to the AU and UD sub-classes:

report_statistics -detailed_analysis DI

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
             Statistics Report
              Stuck-at Faults
-----------------------------------------
...
-----------------------------------------
Fault Sub-classes
  -------------------------
  DI (det_implication)
   EDT  (edt_logic)        30 ( 0.06%)
   SCAN (scan_path)       600 ( 1.20%)
   SEN  (scan_enable     2146 ( 4.30%)
   CLK  (clock)           180 ( 0.36%)
   SR   (set_reset)       300 ( 0.60%)
   MEM  (memory)           30 ( 0.06%)
   DIN  (data_in)          30 ( 0.06%)
...
```

Example 3

The following example shows the statistics report for relevant coverage. A second column is added to the report to compare statistics. The excluded fault sub-classes are marked as “deleted”.

set_relevant_coverage -exclude fp
report_statistics

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
                     Statistics Report
                      Stuck-at Faults
-------------------------------------------------------------
Fault Classes                  #faults          #faults
                               (total)      (total relevant)
   ----------------------- ---------------- ----------------
   FU (full)                    8078584          8048951
   ----------------------- ---------------- ----------------
   UC (uncontrolled)             2079 ( 0.03%)    same ( 0.03%)
   UO (unobserved)              92608 ( 1.15%)    same ( 1.16%)
   DS (det_simulation)        4797074 (59.38%)    same (59.39%)
   DI (det_implication)        677274 ( 8.38%)    same ( 8.39%)
   PU (posdet_untestable)         245 ( 0.00%)    same ( 0.00%)
   PT (posdet_testable)             4 ( 0.00%)    same ( 0.00%)
   UU (unused)                  21254 ( 0.26%)    same ( 0.27%)
   TI (tied)                   602348 ( 7.46%)    same ( 7.47%)
   BL (blocked)                428144 ( 5.30%)    same ( 5.31%)
   RE (redundant)               12063 ( 0.15%)    same ( 0.15%)
   AU (atpg_untestable)       1445491 (17.89%) 1415858 (17.59%)
-------------------------------------------------------------
Fault Sub-classes
   -----------------------
   AU (atpg_untestable)
     BB (black_boxes)          148646 ( 1.84%)    same ( 1.90%)
     FP (false_paths)          165611 ( 2.05%)       deleted
     Unclassified              852523 (10.55%)    same (10.59%)
   UC+UO
     AAB (ATPG_abort)           88056 ( 1.09%)    same ( 1.09%)
     Unclassified                6631 ( 0.08%)    same ( 0.08%)
-------------------------------------------------------------
Coverage
   -----------------------
   test_coverage                     78.04%           79.67%
   fault_coverage                    67.77%           68.37%
   atpg_effectiveness                98.83%           98.83%
-------------------------------------------------------------
#test_patterns                                        17206
   #clock_sequential_patterns                         17206
#simulated_patterns                                   18880
CPU_time (secs)                                      5579.7
-------------------------------------------------------------
```

Example 4

The following example shows the Clock Domain Summary excerpted from a statistics report:

report_statistics -clock_domains summary

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
                    Statistics Report
                     Stuck-at Faults
-------------------------------------------------------------------------

…

Clock Domain Summary                      % faults      Test Coverage
                                           (total)     (total relevant)
----------------------------------------  ------------  -----------------
/pll_ref_clk                                10.06%            8.19%
/pll_ref_clk                                20.00%            0.00%
/tbcin                                       3.03%           32.76%
/reset_tbc20                                 0.01%            6.25%
/pt_clk                                      1.02%           78.73%
/refclkr                                     0.09%           37.57%
/reset_rbc                                   0.01%            6.67%
/func1                                      40.22%           25.25%
/func2                                      18.76%           13.18%
/klok1                                       0.09%           24.43%
/klok2                                       0.00%            0.00%
/klok3                                       0.00%            0.00%
/klok4                                       0.14%           27.20%
/RESET_ALL                                   0.99%            0.21%
/arm_clk2                                    0.02%           38.04%
/arm_clk3                                   27.61%           17.95%
/ddr_clk1x                                   2.57%           35.08%
/ddr_reset                                   0.27%            0.92%
/LSI_TCK                                     2.75%           19.28%
/in_hcm[12]                                  0.13%            0.33%
/in_hcm[10]                                  0.00%            0.00%
/in_hcm[7]                                   0.18%            5.07%
/in_hcm[6]                                   2.19%            1.29%
/in_hcm[5]                                   0.74%            2.06%
/in_hcm[4]                                  18.76%           13.18%
/in_hcm[3]                                   0.14%           27.20%
/in_hcm[2]                                   0.00%            0.00%
/in_hcm[1]                                   0.96%           21.75%
/in_hcm[0]                                   0.43%           21.80%
/bidi[15]                                    0.45%           21.29%
/bidi[14]                                    0.04%            2.00%
-------------------------------------------------------------------------
```

Example 5

The following example performs timing-aware ATPG using SDF timing data. Note that the statistics report includes delay test coverage:

```
set_system_mode analysis
read_sdf my_design.sdf
set_atpg_timing on -clock_waveform DEFAULT 3 1 1
create_patterns
report_statistics
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
                 Statistics Report
                  Transition Faults
-----------------------------------------------
                            #faults     #faults
Fault Classes               (coll.)     (total)
-----------------------     -------     -------
FU (full)                   3370612     4741152
-----------------------     -------     -------
UC (uncontrolled)               846        1339
UO (unobserved)               41993       63027
DS (det_simulation)         2518486     3799766
DI (det_implication)         494643      516429
PU (posdet_untestable)           34          52
UU (unused)                      30          52
TI (tied)                      4270        5308
BL (blocked)                   2654        2754
AU (atpg_untestable)         307656      352425
-----------------------     -------     -------
test_coverage                 89.58%      91.19%
fault_coverage                89.39%      91.04%
atpg_effectiveness            98.73%      98.64%
-----------------------     -------     -------
delay_test_coverage           68.09%      67.39%
-----------------------------------------------
#test_patterns                                3508
      #clock_sequential_patterns              3508
#simulated_patterns                          3616
CPU_time (secs)                           17372.7
-----------------------------------------------
```

Example 6

The following example shows the multiple detection test coverage report at the end of the normal report_statistics output when some unlisted faults are loaded using the “read_faults -graybox” command and AU.PC faults are excluded from the relevant test coverage.

```
read_faults top_faults.txt -retain
read_faults core1_faults.txt -graybox -retain
set_relevant_coverage -exclude au.pc
report_statistics
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following shows the left side of the command output:

```
                    Statistics Report
                     Stuck-at Faults
-----------------------------------------------------------------------
...
Multiple Detection Statistics
-------------------------------------------------------------------------
Detections                                          DS Faults
  (N)                              (Detections == N)  (total w. unlisted)
----------                         ----------------   ------------------
  1                                   0 (  0.00%)      15563 ( 17.22%)
  2                                   0 (  0.00%)      15563 ( 17.22%)
...
  8                                 135 (  0.11%)      15249 ( 16.89%)
  9                                 198 (  0.16%)      15127 ( 16.76%)
  10+                              5238 (  4.12%)      14950 ( 16.57%)
-----------------------------------------------------------------------
bridge_coverage_estimate
-----------------------------------------------------------------------
```

The following shows the right side of the command output, with the Detections column included for your reference.

```
Multiple Detection Statistics
-------------------------------------------------------------------------
Detections                 Test Coverage (Detections >= N)
(N)      (revelant w. unlisted)(total w/o unlisted)(revelant w/o unlisted)
------    --------------------  ------------------  ---------------------
1             17.30%             18220 ( 12.16%)                  12.17%
2             17.30%             18220 ( 12.16%)                  12.17%
...
8             16.97%             17870 ( 11.76%)                  11.78%
9             16.84%             17735 ( 11.61%)                  11.62%
10+           16.65%             17537 ( 11.39%)                  11.40%
-----------------------------------------------------------------------
bridge_coverage_estimate                                          15.55%
-----------------------------------------------------------------------
```

Example 7

The following example shows the statistical fault report when gray or black boxes are present:

report_statistics -Hierarchy -LEVel 2

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
                    Hierarchical Statistics Report
                            Stuck-at Faults
-------------------------------------------------------------------
Instance         Module            #faults     UO    AU   Test Coverage
-------------   ----------------  ---------  ----  ---  -------------
-top-           tde_top_bscan     1602        2316  768    93.16%
  bsr_il        bsr_instance_1       0           0    0        0%
  coria         tde_top           1466          36   95    90.97%
    u1(GB)      td_edt              20           0   10     0.00%
    u2          t_des             1466          36   95    90.79%
  pad_il        pad_inst1           10           0   10     0.00%
    tdbuf (BB)  tri_en_hi           10           0   10     0.00%
  tap_i         tap                  0           0    0     0.00%
```

Example 8

The following example shows that the rows for “Instances Not in netlist” and “Instances in netlist” display, because the command is set to the level of the gray box. In such a case, the report also shows the next lower level instances for the gray box:

report_statistics -hierarchy -LEVel 3

```
                    Hierarchical Statistics Report
                            Stuck-at Faults
-------------------------------------------------------------------
Instance             Module          #faults  UO    AU   Test Coverage
---------         ------------------  ------  ----  ---  -------------
-top-             tde_top_bscan         1602  2316  768    93.16%
  bsr_il          bsr_instance_1           0     0    0        0%
  coria           tde_top               1466    36   95    90.97%
    u1(GB)        td_edt                  20     0   10     0.00%
      Instances Not in netlist            15     0    5     0.00%
      Instances in netlist                 5     0    5     0.00%
        inst1     xxxx                     3     2    1     0.00%
        inst2     yyyy                     2     0   10     0.00%
    u2            t_des                 1466    36   95    90.79%
  pad_il          pad_inst1               10     0   10     0.00%
    tdbuf (BB)    tri_en_hi               10     0   10     0.00%
  tap_i           tap                      0     0    0     0.00%
```

Example 9

The following example shows the hierarchical statistics report using the -grayboxes_only switch:

report_statistics -hierarchy -grayboxes_only

```
                    Hierarchical Statistics Report (Grayboxes Only)
                          Stuck-at Faults
-----------------------------------------------------------------------
       Instance          Module   #faults    UO       AU   Test Coverage
------------------------ -------  -------  -------  -------  ------------
/elt_inst (GB)             elt     18198      8       242      98.44%
Instances not in netlist   elt     15176      8        88      99.30%
Instances in netlist       elt      3022      0       154      93.51%
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 10

The following example shows the hierarchical statistics report using the -EXClude_unlisted_faults switch:

report_statistics -Hierarchy -LEVel 3 -EXClude_unlisted_faults

```
                    Hierarchical Statistics Report
                            Stuck-at Faults
-----------------------------------------------------------------------
Instance            Module           #faults  UO    AU   Test Coverage
---------        ------------------  ------  ----  ---  -------------
-top-            tde_top_bscan         1602  2316     768    93.16%
  bsr_il        bsr_instance_1            0     0       0        0%
  coria         tde_top                1466    36      95    90.97%
    u1(GB)      td_edt                   20     0      10     0.00%
        inst1   xxxx                      3     2       1     0.00%
        inst2   yyyy                      2     0      10     0.00%
    u2          t_des                  1466    36      95    90.79%
  pad_il        pad_inst1                10     0      10     0.00%
    tdbuf (BB)  tri_en_hi                10     0      10     0.00%
  tap_i         tap                       0     0       0     0.00%
```

Example 11

This is an example of the hierarchical statistics report using the -detections option to adjust the report to reflect the values with detections greater than or equal to two.

**report_statistics -hierarchy -detections 2 -level 1 -exclude_unlisted_faults **
-format "DS %tc %rtc %fc %ae"

```
                                    Hierarchical Statistics Report
                         Stuck-at Faults (Detections >= 2)
-------------------------------------------------------------------------------------
Instance                  DS   Test Coverage  TC Loss  Relevant TC  Fault Coverage  ATPG Eff.
---------------------  -----  -------------  -------  -----------  --------------  ---------
-top-                  13324      59.76%      40.24%     56.76%        47.44%        99.28%
  CHPJ_Mg_I_CD_FLP_DCDR_I  21      79.41%       0.02%     79.41%        79.41%       100.00%
  DEF                     504      89.85%       0.49%     89.80%        89.74%       100.00%
  LVISION_JTAP_INST         0       0.00%       0.00%      0.00%         0.00%         0.00%
  LVISION_LOGICTEST_INST    0       0.00%       0.00%      0.00%         0.00%         0.00%
  LV_BURST_CLK_CTRL_I1      0       0.00%       0.00%      0.00%         0.00%         0.00%
  LV_BURST_CLK_CTRL_I2      0       0.00%       0.00%      0.00%         0.00%         0.00%
  LV_SE_CTRL_BL5_P_I1      85      77.34%       0.09%     76.74%        27.81%       100.00%
  LV_SE_CTRL_BL5_P_I2      55      76.83%       0.06%     76.83%        17.70%       100.00%
  LV_SHIFT_CLK_CTRL_INST    0       0.00%       0.00%      0.00%         0.00%         0.00%
  TLB_inst              12636      58.29%      39.15%     55.18%        46.23%        99.24%
```

Example 12

This is an example of the hierarchical statistics report using the -detections option to adjust the report to reflect the values with detections greater than or equal to five and limited with the -grayboxes_only option.

**report_statistics -hierarchy -detections 5 -grayboxes_ only **
-format "DS %tc tcl %rtc %fc %ae"

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
                          Hierarchical Statistics Report (Grayboxes Only)
                          Stuck-at Faults (Detections >= 5)
-------------------------------------------------------------------------------------
 Instance                DS    Test Coverage  TC Loss  Relevant TC  Fault Coverage  ATPG Eff.
-------------------  ------  -------------  -------  -----------  --------------  ---------
/T_inst/E_WPR_IST (GB)    685    25.19%      15.78%     20.29%        13.77%        94.86%
Instances not in netlist    0    27.77%       3.87%     27.60%        18.69%        88.28%
Instances in netlist      685    24.31%      11.91%     18.39%        12.49%        96.57%
```

Example 13

The following example shows AU fault subclasses:

```
                          Statistics Report
                            Stuck-at Faults
-------------------------------------------------------------------------
Fault Classes                              #faults          #faults
                                           (total)      (total relevant)
-----------------------------------  --------------- -------------------
  FU (full)                                 40550            29673
  ---------------------------------  --------------- -------------------
  UO (unobserved)                            3 ( 0.01%)       same ( 0.01%)
  DS (det_simulation)                    14108 (34.79%)       same (47.54%)
  DI (det_implication)                    5111 (12.60%)       same (17.22%)
  PU (posdet_untestable)                    11 ( 0.03%)       same ( 0.04%)
  UU (unused)                             2416 ( 5.96%)       same ( 8.14%)
  TI (tied)                                 24 ( 0.06%)       same ( 0.08%)
  BL (blocked)                               7 ( 0.02%)       same ( 0.02%)
  RE (redundant)                           284 ( 0.70%)       same ( 0.96%)
  AU (atpg_untestable)                   18586 (45.83%)       7709 (25.98%)
-------------------------------------------------------------------------
Fault Sub-classes
  --------------------------------
  AU (atpg_untestable)
    EDT   (edt_blocks)                     603 ( 1.49%)         deleted
    PC*  (pin_constraints)                  15 ( 0.04%)       same ( 0.05%)
    TC*  (tied_cells)                     7642 (18.85%)       same (25.75%)
    OCC   (on_chip_clock_control)          454 ( 1.12%)         deleted
    IJTAG (ijtag)                          873 ( 2.15%)         deleted
    LPCT  (low_pin_count_test)            8947 (22.06%)         deleted
    Unclassified                            52 ( 0.13%)       same ( 0.18%)
  UC+UO
    UNS  (unsuccess)                         3 ( 0.01%)       same ( 0.01%)
  *Use "report_statistics -detailed_analysis" for details.
-------------------------------------------------------------------------
Coverage  --------------------------------
  test_coverage                                50.83%              71.36%
  fault_coverage                               47.41%              64.79%
```

Example 14

The following example shows the reported coverage from a cell-aware static (CA1) ATPG run. The first report considers all UDFM faults from the Tessent CellModelGen models, and the second report considers only the port faults from the UDFM.

report_statistics

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  command: report_statistics
           Statistics Report
           UDFM Static Faults
-------------------------------------
Fault Classes               #faults
                            (total)
----------------------  --------------
  FU (full)                  15679
  --------------------  --------------
  DS (det_simulation)   15301 (97.59%)
  DI (det_implication)    262 ( 1.67%)
  UU (unused)              42 ( 0.27%)
  TI (tied)                39 ( 0.25%)
  RE (redundant)           35 ( 0.22%)
-------------------------------------
Coverage
  --------------------
  test_coverage               100.00%
  fault_coverage               99.26%
  atpg_effectiveness          100.00%
-------------------------------------
#test_patterns                     67
#simulated_patterns                83
CPU_time (secs)                   4.7
-------------------------------------
```

report_statistics -port_equivalent_faults

```
//  command: report_statistics -port_equivalent_faults
           Statistics Report
           UDFM Static Faults
-------------------------------------
Fault Classes               #faults
                            (total)
----------------------  --------------
  FU (full)                   2028
  --------------------  --------------
  DS (det_simulation)    1798 (88.66%)
  DI (det_implication)    206 (10.16%)
  UU (unused)              18 ( 0.89%)
  TI (tied)                 4 ( 0.20%)
  RE (redundant)            2 ( 0.10%)
-------------------------------------
Coverage
  --------------------
  test_coverage               100.00%
  fault_coverage               98.82%
  atpg_effectiveness          100.00%
-------------------------------------
#test_patterns                     67
#simulated_patterns                83
CPU_time (secs)                   4.7
-------------------------------------
```

Related Topics

set_atpg_timing

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_relevant_coverage

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_synchronous_clock_groups

Context: dft, patterns

Mode: setup, analysis

Displays a list of all user-defined synchronous clock groups.

Usage

report_synchronous_clock_groups [-scope {capture | shift | all}]

Note

Synchronous clock groups have no impact on scan chain analysis, test point identification, and EDT IP insertion (in the dft -scan, dft -test_points and dft -edt sub-contexts).

Description

Displays a list of all user-defined synchronous clock groups.

The report_synchronous_clock_groups command displays a list of all clock groups that you defined with the add_synchronous_clock_group command. For more information about synchronous clock groups, refer to the add_synchronous_clock_group command description.

Arguments

- -scope {capture | shift | all}

  An optional argument and literal pair that specifies whether to display only capture or shift clock groups or all clock groups. By default, all clock groups are displayed.

Examples

The following example creates two clock groups for capture and one for shift and lists them out.

```
SETUP> add_synchronous_clock_group {I1 I2}
SETUP> report_synchronous_clock_groups

User-defined Synchronous Clock Group
====================================
//  No.  Clocks (Labels)   Scope
//  ---  ---------------  -------
//   1   I1               capture
//       I2

SETUP> add_synchronous_clock_group -scope all {I3 I4}
SETUP> report_synchronous_clock_groups
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
User-defined Synchronous Clock Groups
=====================================
//  No.  Clocks (Labels)   Scope
//  ---  ---------------  -------
//   1   I1               capture
//       I2
//   2   I3               capture
//       I4
//  ---  ---------------  -------
//   1   I3                shift
//       I4
```

SETUP> report_synchronous_clock_groups -scope shift

```
User-defined Synchronous Clock Groups
=====================================
//  No.  Clocks (Labels)  Scope
//  ---  ---------------  -----
//   1   I3               shift
//       I4
```

Related Topics

add_clocks

add_synchronous_clock_group

analyze_control_signals

delete_clocks

delete_synchronous_clock_group

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_tcd_ldb_validation

Context: patterns -scan_diagnosis

Mode: setup

Verifies that the core instance pathnames and hierarchy in the TCD matches the core instance pathnames and hierarchy in the LDB.

Usage

report_tcd_ldb_validation **-tcd** ***tcdfile*** ... **-ldb** ***ldb_dir*** ...

Arguments

- **-tcd** ***tcdfile ...***

  Required switch and string pair that specifies one or more TCD files.

- **-ldb** ***ldb_dir*** ...

  Required switch and string pair that specifies one or more LDB directories.

Description

The command extracts and compares the set of chip designs and core instance paths from the TCD files and the LDBs. The report lists the coverage of each instance path in the TCD files and the LDBs. The tool issues an error when the names do not match.

The typical flow runs ATPG, LDB creation, and diagnosis in separate phases. Run report_tcd_ldb_validation after ATPG and LDB creation to ensure that the correct data is available at diagnosis. Refer to "Diagnosis for Hierarchical Design" in the *Tessent Diagnosis User's Manual* for details.

Examples

The following example validates the chip.tcd against the core data within the core.ldb and chip.ldb directories.

```
report_tcd_ldb_validation -tcd chip.tcd -ldblist core.ldb
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Core validation report:

TCD Files:

  Index      Filename          Design
  1>         chip.tcd          chip_top_edt

LDB Directories:

  Index      Directory         Design
  1>         core.ldb          uUARTs_i

Instance Coverage:

Instance        TCDs   LDBs
cpu/core_1      1      1
```

The following example shows the output when you have multiple TCDs and cores. In the transcript, the third and fourth TCD files refer to the same chip with different pattern sets. Each chip has embedded cores and a chip level LDB. The first LDB refers to core “core_D,” which is not found in the LDB directories. The second LDB contains a top-level chip, which is not found in any TCD file leading to the chip’s instance not being found.

set tcd { chip_1.tcd chip_2.tcd chip_3.tcd }

set ldb { core_A.ldb core_B.ldb core_C.ldb chip_1.ldb chip_2.ldb chip_3.ldb chip_4.ldb }

report_tcd_ldb_validation -tcd $tcdlist -ldb $ldblist

```
Core validation report:

TCD files:

  Index      Filename          Design
  1          chip_1.tcd        chip_1_edt_top
  2          chip_2.tcd        chip_2_edt_top
  3          chip_3.tcd        chip_3_edt_top
  4          chip_4.tcd        chip_3_edt_topLDB Directories:

  Index      Directory         Design
  1          core_A.ldb        core_A
  2          core_B.ldb        core_B
  3          core_C.ldb        core_CInstance Coverage:

Instance         TCDs          LDBs
cpu/core_A_1     1             1
cpu/core_A_2     1             1
cpu/core_B_1     1             2
cpu/core_D_1     1             [Not Found]
cpu/core_B_1     2             2
cpu/core_C_1     2             3
cpu/core_C_1     3 4           3
chip_4/core_B_1  [Not Found]   2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_tcl_shell_options

Context: all contexts

Mode: setup

Lists the currently specified Tcl shell options.

Usage

report_tcl_shell_options

Description

Lists the currently specified Tcl shell options as specified with the set_tcl_shell_options command.

Arguments

None

Related Topics

get_tcl_shell_option

set_tcl_shell_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_test_end_icall

Context: dft, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Modes: setup, analysis

Reports a single view of all existing iCalls specified with the set_test_end_icall command.

Usage

report_test_end_icall

Description

Reports a single view of all existing iCalls that were specified with the set_test_setup_icall command.

Arguments

None.

Related Topics

report_test_setup_icall

set_test_setup_icall

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_test_logic

Context: dft -scan, dft -test_points

Mode: insertion

Prerequisites: Test logic or test points must be added with the insert_test_logic command.

Displays the test logic that the tool added during the scan insertion process.

Usage

report_test_logic [-Instance | -Module | -Summary] [{> | >>} *file_pathname*]

Description

Displays the test logic that the tool added during the scan insertion process.

The report_test_logic command displays information about the additional library cells instantiated in the design during the scan insertion process.

Arguments

- -Instance

  An optional switch that displays the list of instance pathnames and the corresponding DFT library models inserted by the tool. This option also includes a summary of the number of each instance-based DFT library model that the tool inserted. This is the default.

- -Module

  An optional switch that displays the list of module names, the list of instance pathnames, and the corresponding DFT library models that the tool inserted as test logic and test points. This option also includes a summary of the number of each module-based DFT library model that the tool inserted.

- -Summary

  An optional switch that displays a summary that contains the number of each module and instance-based DFT library model that the tool inserted.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Related Topics

insert_test_logic

report_test_points

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_test_points

Context: dft -test_points

Mode: all modes

Displays test points that are inserted with the insert_test_logic command.

Usage

report_test_points [*object_spec*] [-columns *column_names*]
[-add_columns *column_names*]

Description

Use this command to report the test points that are inserted with the insert_test_logic command.

The report is printed in a table format where each column represents the test point attributes, such as name, location, type, or clock_domain, for each test_point. Use the -columns switch to specify which columns to show and the order in which they appear. By default, report_test_points prints the following test_point attributes:

```
name, type, control_point_type, enable_connection_name, clock_domain,
flop_location.
```

You can add additional columns after the default columns (or after the columns specified with the -columns switch) using the -add_columns switch.

The following table summarize what is new or changed in the report_test_points command and provides a mapping between the old and new command options.

Previous command option	New command option
-internal_clock_paths	-add_columns clock_connection_name
-sharing id	-add_columns shared_id
Summary	(No longer supported)
-control/-observe	report_test_points [get_test_points -type control/observe]
-tool_generated_only	report_test_points [get_test_points -origin design_analysis]

Arguments

- *object_spec*

  An optional value that specifies a Tcl list of one or more object names or a collection of one or more test_point objects.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -columns *column_names*

  An optional value that specifies a Tcl list of one or more object names that are the attributes of a test_point object. See "Test Point Data Model" for the list of Test point attributes that are valid column names.

- -add_columns *column_names*

  An optional list of columns to append after the columns specified by the -columns switch (or after the default columns). See "Test Point Data Model" for the list of Test point attributes that are valid column names.

Examples

Example 1

The following example shows the default columns printed when the command is issued with no arguments:

```
report_test_points
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
==========================================================================================
name                        type     control_point_type  enable_connection_name  clock_domain
flop_location
------------------------  -------  ------------------  ----------------------  -------------
tp#test_module2/neg1/Z    control  AND                 /lbist_en               clk2
/test_module2/ts_default_PD_u12_cp_6sffp1_i

tp#test_module2/neg2/Z    control  OR                  /lbist_en               clk2
/test_module2/ts_default_PD_u12_cp_8sffp1_i

tp#1/test_module2/neg1/Z  observe  --                  /lbist_en               clk2
/test_module2/ts_default_PD_u12_op_120sffp1_i

tp#1/test_module2/neg2/Z  observe  --                  /lbist_en               clk2
/test_module2/ts_default_PD_u12_op_122sffp1_i

tp#test_module1/neg1/Z    control  AND                 /lbist_en               clk2
/test_module1/ts_default_PD_u11_cp_0sffp1_i

tp#test_module1/neg2/Z    control  OR                  /lbist_en               clk2
/test_module1/ts_default_PD_u11_cp_4sffp1_i

tp#1/test_module1/neg1/Z  observe  --                  /lbist_en               clk2
/test_module1/ts_default_PD_u11_op_112sffp1_i

tp#1/test_module1/neg2/Z  observe  --                  /lbist_en               clk2
/test_module1/ts_default_PD_u11_op_114sffp1_i

tp#u1/U15/Z               observe  --                  /lbist_en               clk1
/u1/ts_default_PD_default1_op_128sffp1_i

tp#u1/U14/Z               observe  --                  /lbist_en               clk1
/u1/ts_default_PD_default1_op_126sffp1_i

tp#u1/U75/Z               observe  --                  /lbist_en               clk2
/u1/ts_default_PD_default1_op_212sffp1_i

tp#U64/Z                  observe  --                  /lbist_en               clk2
/ts_default_PD_default1_op_52sffp1_i

tp#u1/U71/Z               observe  --                  /lbist_en               clk2
/u1/ts_default_PD_default1_op_206sffp1_i

…
```

Example 2

The following example shows only control points below the test_module2 instance:

report_test_points [get_test_points -below_instances test_module2 -type control]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
==========================================================================================
name                    type  control_point_type  enable_connection_name  clock_domain
flop_location
------------------------------------------------------------------------------------------
tp#test_module2/neg1/Z  control  AND                 /lbist_en               clk2
/test_module2/ts_default_PD_u12_cp_6sffp1_itp#test_module2/neg2/Z  control  OR
/lbist_en               clk2
/test_module2/ts_default_PD_u12_cp_8sffp1_i
```

Related Topics

add_input_constraints

delete_test_points

get_test_points

insert_test_logic

report_test_logic

set_test_point_insertion_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_test_point_statistics

Context: dft -test_points

Mode: analysis (after analyze_test_points is run), insertion

Use this command to report statistics regarding the location of test points inside the design.

Usage

report_test_point_statistics [*object_spec*] [-top *lines*]] [-columns *column_names*] [-sort {*column* ...}] [-add_columns *column_names*]

Description

The report_test_point_statistics command provides statistical information in a tabular format regarding the dispersion of test points throughout the design. This can help identify parts of the design that can be modified for greater controllability or observability and to reduce the number of test points needed.

By default, when no arguments or switches are specified, the output includes a row for each instance in the design that includes at least one test point. Each row contains the module name (if the name is not unique it includes an asterisk followed by a unique index), columns that include the number of test points, control points and observe points in the specified instance (including test points in submodules below the module), and the full path to the instance.

The lines are sorted in descending order according to the number of test points.

Note

The command is only available after test point analysis and insertion of test points.

Arguments

- *object_spec*

  An optional value that specifies a Tcl list of one or more modules or instances in the design, that controls which instances are reported.

- -top *lines*

  An optional switch and integer pair that specifies the number of the top rows of the report to be printed, not including the header.

- -columns {*columns* ... }

  An optional switch that specifies to organize the output format according to a specified list of columns as follows:

  - module
  - test_points
  - control_points

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- observe_points
- instance

This switch also enables you to change the order of the columns and hide default columns.

- -sort *column*

  An optional switch and column name that sorts the output rows according to the value of the specified column in descending order. Default is to sort according to the test_points column.

- -add_columns *column_names*

  An optional list of columns to append after the columns specified by the -columns switch (or after the default columns).

Examples

Example 1

The following example prints the top two rows of the default report.

report_test_point_statistics -top 2

```
//  command: report_test_point_statistics -top 2
==========================================================================
module                test_points  control_points  observe_points  instance
-------------------   -----------  --------------  --------------  --------
MT_GET (top)          21 (100.0%)     14 (100.0%)      7 (100.0%)  /
MT_SET_DATA_WIDTH32   19  (90.5%)     14 (100.0%)      5  (71.4%)  /MT_BLK
```

Example 2

The following example reports the test point statistics for the specified instances.

report_test_point_statistics {MT_SET_DATA_WIDTH32 LCG_DATA_WIDTH32_1}

```
//  command: report_test_point_statistics {MT_SET_DATA_WIDTH32 LCG_DATA_WIDTH32_1}
==========================================================================
module                test_points  control_points  observe_points  instance
-------------------   -----------  --------------  --------------  ------------
MT_GET (top)          21 (100.0%)     14 (100.0%)      7 (100.0%)  /
MT_SET_DATA_WIDTH32   19  (90.5%)     14 (100.0%)      5  (71.4%)  /MT_BLK
LCG_DATA_WIDTH32_1     3  (14.3%)      3  (21.4%)      0   (0.0%)  /MT_BLK/LCG2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_test_setup_icall

Context: dft, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Modes: setup, analysis

Reports a single view of all existing iCalls specified with the set_test_setup_icall command.

Usage

report_test_setup_icall

Description

Reports a single view of all existing iCalls specified with the set_test_setup_icall command.

Each iCall is reported as follows:

```
iCall iproc_name [arguments ...]
```

When multiple iCalls are merged, the command reports these iCalls within the “iMerge -begin” and “iMerge -end” as follows:

```
iMerge -begin
    iCall iproc_name [arguments ...]
    ...
iMerge -end
```

Arguments

None.

Related Topics

report_test_end_icall

set_test_setup_icall

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_test_stimulus

Context: dft -edt, patterns -scan

Mode: analysis

Displays the stimulus necessary to satisfy the specified set, write, or read conditions.

Usage

report_test_stimulus
-Set {{***id#*** | ***pin_path_name***} {**0** | **1** | **Z**}}...
| **-Cycle_set** {[-FIrst_events | -LEading_edge_events | -LAst_events]
{{*id#* | *pin_path_name*} {0 | 1 | Z} *cycle#*}...}...
| **-Write** {{***id#*** | ***instance_name***} ***address_values*** [*data_values*]}...
| **-Read** {{***id#*** | ***instance_name***} ***address_values***}...
| **-RWx** {{***id#*** | ***instance_name***} ***address_values***}...
| {**-SENsitize** {{***id#*** | ***instance_name*** | ***pin_path_name***
[-Observe_at {*id#* | *instance_name* | *pin_path_name*} [-Expect {0 | 1 | X | Z}]]}}
| -Port *port_number*
[-Verbose | -Noverbose] [-PRevious] [-STore] [{> | >>} *file_pathname*]

Description

Displays the stimulus necessary to satisfy the specified set, write, or read conditions.

This command identifies how to sensitize scan chain blockage points. For example, if you first delete all scan groups and then go to the ATPG system mode, you can issue the report_test_stimulus command for possible conditions to satisfy sensitization. That is, if the blockage was at an AND gate, you can try to set an input of the gate to 1.

If test generation is successful, the stimulus necessary to satisfy the specified conditions is displayed. The stimulus for scan cells is identified by the gate index number, instance name, and cell ID number of the scan chain.

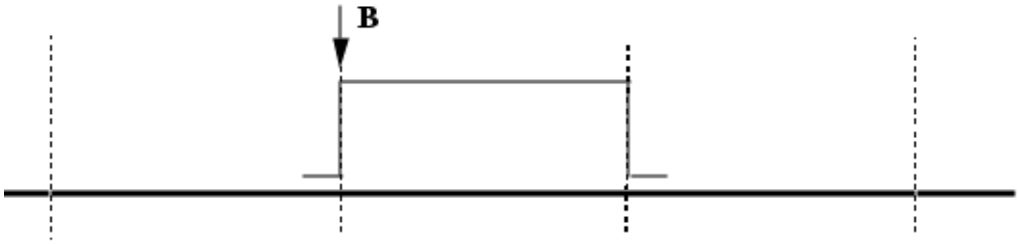
You can access the simulated values for internal gates by using the set_gate_report command with a parallel_pattern of 0.

You can also use the “report_test_stimulus -Cycle_set” command to specify the conditions at different event times or test cycles for analysis (for example, sequential controllability analysis and transition fault analysis). Each test cycle consists of at most three sets of events (clock-off, leading edge, and trailing edge) simulated in a single cycle. The following is a description of what determines the set of events within which the justified/simulated event occurs:

- When both “clock-off simulation” and “split capture” are turned off (the default unless the “set_clock_off_simulation On” and “set_split_capture_cycle On” commands are issued or set by ATPG Expert), a cycle includes one set of simulation events; that is, there are no clock-off events and the leading edge and the trailing edge are handled in one time frame. The simulation occurs at point B, as shown in Figure 7-7.

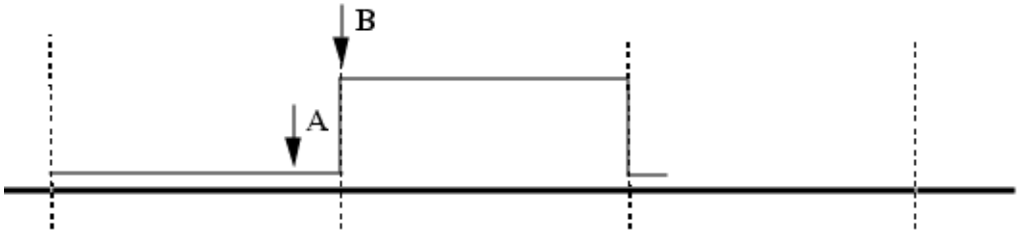
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 7-7. set_clock_off_simulation Off and set_split_capture_cycle Off



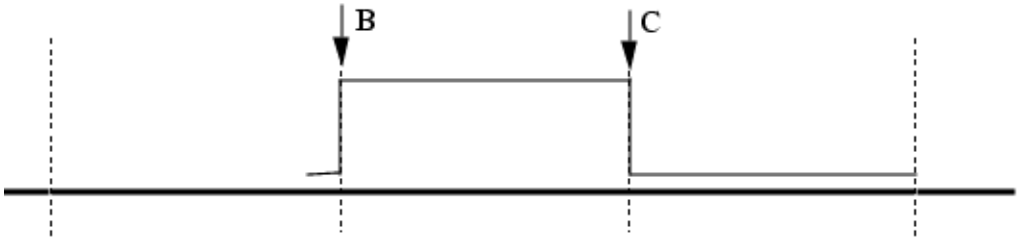
- When “clock-off simulation” is turned on and “split capture” is turned off, a cycle includes two sets of simulation events: clock-off events, and the events handling both the leading edge and the trailing edge events. The two simulation points for this case are at points A for clock-off events and B for leading and trailing edge events with “split capture” off, as shown in Figure 7-8.

Figure 7-8. set_clock_off_simulation On and set_split_capture_cycle Off



- When “clock-off simulation” is turned off and “split capture” is turned on, a cycle includes two sets of simulation events: the leading edge and trailing edge events. For this case (as shown in Figure 7-9), the simulation points are point B for the leading edge events and point C for the trailing edge events. Turning on “split capture” is necessary if a leading edge flip-flop feeds a trailing edge flip-flop in the functional path.

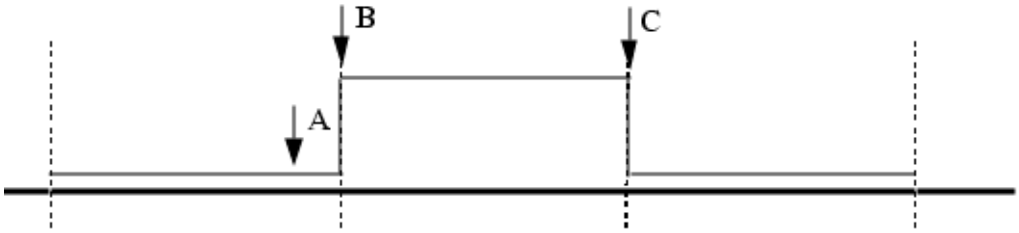
Figure 7-9. set_clock_off_simulation Off and set_split_capture_cycle On



- When both “clock-off simulation and” “split capture” are turned on, a cycle includes three sets of events: clock-off, the leading edge, and the trailing edge. Figure 7-10 includes the three simulation points: A for clock-off, B for leading edge, and C for trailing edge events.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 7-10. set_clock_off_simulation On and Set Split Capture On



Arguments

- **-Set *id#* | *pin_path_name* 0 | 1 | Z**

  A switch with a repeatable argument and literal pair that specifies the pin and its value for which you want to generate the appropriate stimulus. You may specify multiple argument pairs with a single -Set switch.

  id# — An integer that specifies the identification number of the gate for which you want to set the output pin. The gate cannot be a RAM or ROM gate.

  pin_path_name — A string that specifies the pathname of the pin you want to set.

  0 — A literal that sets the *id#* or *pin_path_name* to 0.

  1 — A literal that sets the *id#* or *pin_path_name* to 1.

  Z — A literal that sets the *id#* or *pin_path_name* to Z.

- **-Cycle_set** {[-FIrst_events | -LEading_edge_events | -LAst_events] {{*id#* | *pin_path_name*} {0 | 1 | Z} *cycle#*} …} …

  A switch and repeatable argument, literal, and number set that specifies the pin, its value, and the cycle to set the value at the pin for which you want to generate the appropriate stimulus.

  The events switches determine when the value is justified/simulated in the cycle. The events switches hold for all triples (pin, value, cycle) specified thereafter until another events switch is specified in the same command. For example, you can specify "report_test_stimulus -first <triple> <triple> -last <triple>" rather than "report_test_stimulus -first <triple> -first <triple> -last <triple>".

  -FIrst_events — A switch that sets the value of all pin triplets following this switch when the first events are simulated. If the "set_clock_off_simulation On" command has been issued or set by ATPG Expert, this event is the clock-off simulation; otherwise, it is always the present leading edge event simulation.

  -LEading_edge_events — A switch that sets the value of all pin triplets following this switch at the leading edge event simulation. ATPG always has one set of events, called the "leading edge," that it simulates by default in all cycles. Leading edge always refers to that one possibility.

  -LAst_events — A switch that sets the value of all pin triplets following this switch in the last set of simulation events that occur in every cycle. This is the default. If the "set_split_capture_cycle On" command was issued or set by ATPG expert, this event

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is the trailing edge events simulation. By default, the value of all pins is set at the trailing edge event in the cycle if an events switch is not specified in the -Cycle_set switch. Also, when there are several -Cycle_set switches specified in the same command line, each -Cycle_set switch resets its default time frame to the trailing edge frame regardless of the time frame specified in the previous -Cycle_set switch.

id# — An integer that specifies the identification number of the gate for which you want to set the output pin. The gate id# cannot be a flattened RAM or ROM gate.

pin_path_name — A string that specifies the pathname of the pin you want to set.

0 — A literal that sets the *id#* or *pin_path_name* to 0.

1 — A literal that sets the *id#* or *pin_path_name* to 1.

Z — A literal that sets the *id#* or *pin_path_name* to Z.

cycle# — An integer that specifies the post-load capture cycle in which the event is to occur.

With named capture procedures, the first cycle after the first scan load is “1”, the second cycle after first scan load is “2”, and so on.

Without named capture procedures, “1” indicates the cycle may be anywhere in the window and can “slide” within the allocated depth. The *cycle#* earlier is always “0” if it exists, before that “-1” if it exists, and so on. The *cycle#* after “1” is always “2”, and so on. For example, if the depth is set to 1 by you or ATPG Expert, then there is only cycle “1”. If the depth is set to 2, the cycles could be numbered “0” then “1” or “1” then “2”. In this case, if you specified “0” and “1” for *cycle#*, the cycles are the first and second cycles and there is no cycle “2”. However, if you specified “1” and “2” for *cycle#*, the cycles are the first and second cycles, and there is no cycle “0”.

Note

When used without named capture procedures, -Cycle_set is not constrained from satisfying control requirements in the observation cycle(s). Therefore, it is possible for it to succeed although ATPG (stuck-fault, transition fault, MacroTest read/observe pattern, etc.) would fail, because they typically have reserved at least one observation cycle. Such false successes can be prevented by temporarily reducing depth to artificially reserve observation cycle(s), or by scheduling a clock pulse or some other event in each observation cycle. Each solves the false success problem by creating -cycle_set compatibility with the cycle requirements imposed upon ATPG for it to succeed. If -Cycle_set fails, ATPG should also.

- **-Write *id#* | *instance_name address_values*** *data_values*

  A switch with a repeatable argument that specifies the RAM to which you want to write and, optionally, its address and data values. You may specify multiple argument triplets with a single -Write switch. The switch arguments are:

  id# — An integer that specifies the identification number of the RAM gate to which you want to write.

  instance_name — A string that specifies the pathname of the RAM instance to which you want to write.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

address_values — A required character string consisting of 0's and 1's that specifies the values you want to place on the RAM address lines. The least significant value must be the last character in the string. The number of characters in the string must not exceed the number of RAM address lines available.

data_values — An optional character string consisting of 0's, 1's, and X's that specifies the values you want to place on the RAM data lines. The least significant value must be the last character in the string. The number of characters in the string must not exceed the number of RAM data lines available.

If you do not specify the -Port switch, the command assumes the first port (port 0).

- **-Read *id#* | *instance_name address_values***

  A switch with a repeatable argument that specifies the RAM from which you want to read and, optionally, its address value. You may specify multiple argument pairs with a single -Read switch.

  id# — An integer that specifies the identification number of the RAM gate from which you want to read.

  instance_name — A string that specifies the pathname of the RAM instance from which you want to read.

  address_values — A required character string consisting of 0's and 1's that specifies the values you want to place on the RAM address lines. The least significant value must be the last character in the string. The number of characters in the string must not exceed the number of RAM address lines available.

  If you do not specify the -Port switch, the command assumes the first port (port 0).

- **-RWx *id#* | *instance_name address_values***

  A switch that enables a port read to be combined with a port write by combining -write and a -previous option in the commands after -rwx. Both the read and write enables of the RAM are then asserted together. The switch arguments are:

  id# — An integer that specifies the identification number of the RAM gate from which you want to read.

  instance_name — A string that specifies the pathname of the RAM instance from which you want to read.

  address_values — A required character string consisting of 0's and 1's that specifies the values you want to place on the RAM address lines. The least significant value must be the last character in the string. The number of characters in the string must not exceed the number of RAM address lines available.

- **-SENsitize *id#* | *instance_name* | *pin_path_name***

  A switch that enables any primitive pin or pins to be targeted for sensitization; RAM pin, RAM port, ROM pin, ROM port, MUX output pin, MUX input pin, and so on. The conditions necessary to observe the output pins (or pin) of that primitive are calculated and reported.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The -SENsitize argument, like macrotest, uses a functional observation technique, unlike the basic ATPG algorithm for structural faults. Consequently, there must be a single sensitizable path from the site to be observed to some observation site (scan cell or observable PO). The sensitizable past must be through combinational gates (specifically, no sequential).

If the tool can sensitize this path, then ATPG should succeed. If the tool cannot sensitize this path, then ATPG fails. This failure, however, indicates nothing about why a fault site is untestable, and you cannot make any conclusions about why ATPG is failing.

By issuing successive -sensitize options with different pin path names while also including the -previous option, you can find (and store with -store) the conditions that would permit a test to observe multiple sites simultaneously, even though they are driven by the outputs of different primitives. In this way, you can specify more than one pin for simultaneous observation. If a single primitive has multiple outputs, such as a RAM, and only the RAM instance name is given, all of the outputs of the RAM are observed simultaneously, or a report of failure to sensitize is issued. The switch arguments are:

id# — An integer that specifies the identification number of the RAM gate from which you want to read.

instance_name — A string that specifies the pathname of the RAM instance from which you want to read.

pin_path_name — A string that specifies the pathname of the pin that you want to sensitize.

-Observe_at *id# | pin_path_name | instance_name*

Specifies where the sensitized primitive output should be observed (latched for comparison). The -observe_at switch only applies to the -Sensitize switch when a single pin is specified. It is possible to specify multiple -Sensitize/-Observe_at pairings by using the -Previous option. If non-interfering paths cannot be created between all simultaneously active -Sensitize/-Observe_at points, a message is issued, and the run terminated.

-Expect 0 | 1 | X | Z

The -expect modifier also only applies if a -sensitize option is given on the same command line. It is followed by the expected output value for the primitive to be sensitized (either 0, 1, or X). The X value can also be specified using lower case x, and means unspecified output value. For example, if a RAM is being sensitized, then one output value for each output bit of the RAM must be specified if output values are given using -expect. The bits should be specified in the same order as the RAM's outputs. If a RAM named mem1 is specified with outputs Dout<7>...Dout<0>, then if the expected outputs are included, there must be 8 expected values starting with the expected value for Dout<7> as the leftmost bit. For example, 1000000x after -expect could be used to specify that the RAM's outputs are expected to be 1 for Dout<7>, and 0 for all other bits except Dout<0>, the expected output of which is Unknown or Don't Care. If this RAM's outputs were sensitized to scan latches along an inverting

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

path, 0111111X would be stored as the expected latched results to be scanned out as the test results.

- -Port *port_number*

  An optional switch and integer pair that specifies the identification number of the port to use for reading or writing RAM. The port identification number is zero-based; that is, the first port is “port 0”. The default is 0.

- -Verbose

  An optional switch that displays the pattern that the command creates to satisfy the specified settings. This is the default.

- -Noverbose

  An optional switch that specifies to not display the pattern that the command creates to satisfy the specified settings.

- -PRevious

  An optional switch that retains the settings from the previous report_test_stimulus command and adds them to the current settings. When this switch is used, the command displays all of the retained settings. The default is to not retain the settings.

- -STore (not available in Tessent TestKompress (EDT On))

  An optional switch that places the command-created pattern in the internal test pattern area. You can then write this pattern to a file, in any format, by using the write_patterns command. The default is to not place the pattern in the internal test pattern area.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

The following example command line is used for determining what stimulus is needed to write 11011 to address location 01011 for a RAM gate (gate ID number is 67):

```
set_system_mode analysis
report_test_stimulus -write 67 01011 11011
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following is an example of the display from the previous command line:

```
//  Time = 0
//  Force 1  /W1 (1)
//  Force 0  /A1[4] (2)
//  Force 1  /A1[3] (3)
//  Force 0  /A1[2] (4)
//  Force 1  /A1[1] (5)
//  Force 1  /A1[0] (6)
//  Force 0  /OE (7)
//  Force 1  /D1[0] (14)
//  Force 1  /D1[1] (15)
//  Force 0  /D1[2] (16)
//  Force 1  /D1[3] (17)
//  Force 1  /D1[4] (18)
```

Example 2

The following example command lines display the stimulus necessary to satisfy the specified conditions at different event times or test cycles. It is assumed that both “split capture” and “clock-off simulation” are turned on, and you want to determine if a slow-to-fall transition can be launched at the output pin of gate 245.

report_test_stimulus -cycle_set 245 1 0 245 0 1

In this example, logic 1 is set at the default/last event (trailing due to split capture) simulated in cycle 0, and logic 0 is set at the default/last event (trailing due to split capture) simulated in cycle 1.

report_test_stimulus -cycle_set -leading_edge 245 1 0 245 0 1

In this example, logic 1 is set at the leading edge event of cycle 0, and logic 0 is set at the leading edge event of cycle 1.

report_test_stimulus -cycle_set -first_events 245 1 0 245 0 1

In this example, logic 1 is set at the first event (clock-off) simulated in cycle 0, and logic 0 is set at the first event (clock-off in this example) of cycle 1.

report_test_stimulus -cycle_set -first_events 245 1 0 -last_events 245 0 1

In this example, logic 1 is set at the first (clock-off) event simulated in cycle 0, and logic 0 is set at the last event (trailing edge) simulated in cycle 1.

Example 3

In the following example command lines, it is assumed only split capture is turned on, and you want to determine if a slow-to-rise transition can be launched at the output pin of gate 245.

report_test_stimulus -cycle_set 245 0 1 245 1 2

In this example, logic 0 is set at the default/last (trailing due to split capture) event simulated in cycle 1, and logic 1 is set at the default/last (trailing due to split capture) event simulated in cycle 2.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_test_stimulus -cycle_set -leading_edge 245 0 1 245 1 2

In this example, logic 0 is set at the leading edge event of cycle 1, and logic 1 is set at the leading edge event of cycle 2.

report_test_stimulus -cycle_set -first_events 245 0 1 245 1 2

In this example, logic 0 is set at the first (leading-edge) event simulated in cycle 1, and logic 1 is set at the first (leading-edge in this example) event of cycle 2.

report_test_stimulus -cycle_set -first_events 245 0 1 -last_events 245 1 2

In this example, logic 0 is set at the first (leading_edge) event simulated in cycle 1, and logic 1 is set at the last (trailing edge in this example) event simulated in cycle 2.

Related Topics

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_testbench_simulation_options

Context: unspecified, all contexts

Mode: all modes

Reports the default values for many options used in the run_testbench_simulations command.

Usage

report_testbench_simulation_options [-simulator {questa | vcs}]

Description

Reports the default values for many options used in the run_testbench_simulations command.

Arguments

- -simulator questa | vcs

  A switch and literal that specifies the simulator you want to use for simulation. The default value is “questa” from Siemens EDA. The other option is vcs from Synopsys.

Examples

The following example shows the result of the report_testbench_simulation_options command before any default value was changed and after the simulation output directory default value was changed.

```
SETUP> set_context patterns
SETUP> set_testbench_simulation_options -simulation_output_directory ./my_sim_outdir
SETUP> report_testbench_simulation_options

// Testbench simulation options
// ============================
// -------------------------------  ---------------------------
// Option                           Value
// -------------------------------  ---------------------------
// default_simulator                questa
// simulation_output_directory      ./my_sim_outdir
// keep_simulation_data             on_failure
// store_simulation_waveforms       off
// simulation_timeout               unlimited
// parallel_simulations             1
// simulator_options                {}
// waveform_configuration_commands  {log -r /*; run -all; exit}
//
```

Related Topics

check_testbench_simulations

run_testbench_simulations

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_tester_options

Context: patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting, patterns -silicon_insight

Mode: All

Enables you to confirm which tester-specific options are set to control how the tool writes tester patterns when you use SSN.

Usage

report_tester_options

Description

The report_tester_options command displays a list of the tester options that are currently defined as a result of the set_tester_options command.

Note

These tester options apply only to designs with SSN.

Arguments

None

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_tester_statistics

Context: patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: analysis

Reports tester statistics, such as tester cycles and test time.

Usage

report_tester_statistics [-begin *begin*] [-end *end*] [-maxloads *maxloads*] [-external] [-pattern_sets {all | chain | scan | ssh_loopback | ssn_continuity | ssh_on_chip_compare}...] [-chain_test | -scan_test]

Description

Provides a report of pattern tester statistics before you write the pattern. This pattern requires SSN to be present and active.

Arguments

- -begin *begin*

  An optional switch and integer pair that specifies the first pattern of a range to report. This argument is inclusive. The default first pattern is pattern 0.

- -end *end*

  An optional switch and integer pair that specifies the last pattern of a range to use for reporting. This argument is inclusive. The default end pattern is the last pattern of the pattern set.

- -maxloads *maxloads*

  An optional switch and positive integer pair that specifies the maximum number of scan loads per pattern file in the tester statistics report.

- -external

  An optional switch that reports the current external pattern set. By default, this command saves internal test patterns.

- -pattern_sets {all | chain | scan | ssh_loopback | ssn_continuity | ssh_on_chip_compare}

  An optional switch and keyword pair that selects the type of patterns to include in the report. By default, the tool reports both scan and chain patterns. If you choose ssh_loopback, it reports SSH loopback patterns for the active SSH instruments. In this case, unless you choose -chain_test or -scan_test, the command uses both scan and chain patterns for throttling.

- -chain_test | -scan_test

  An optional switch that specifies whether to use chain or scan tests only as the pattern set for SSN throttling when you choose the ssh_loopback pattern set.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example uses sample output.

report_tester_statistics

```
-----------------  ---------------  --------------  -----------  -------------
  Procedure          Tester Cycles    Cycle Period    Test Time    IJTAG Clock
                                                                      Ratio
-----------------  ---------------  --------------  -----------  -------------
  init_cycle                 1            10ns             10ns
  test_setup               752            10ns           7520ns         16
  ssn_setup               1937            10ns          19370ns         16
  shift + capture        10780            10ns         107800ns
  test_end                 401            10ns           4010ns         16
-----------------  ---------------  --------------  -----------  -------------
  Total                  13871                         138710ns
                         13.9K                          138.7us
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_tied_signals

Context: all contexts

Mode: all modes

Displays a list of the tied floating signals and pins.

Usage

report_tied_signals [-Class {Full | User | System}] [{> | >>} *file_pathname*]

Description

Displays a list of the tied floating signals and pins.

The report_tied_signals command displays either the user class, system class, or full classes of tied floating signals and pins. If you do not specify a class, the command displays all the tied floating signals and pins.

Arguments

- -Class Full | User | System

  An optional switch and literal pair that specifies the source (or class) of the tied floating signals or pins which you want to display. The literal choices are as follows:

  Full — A literal that displays all the tied floating signals or pins in the user and system class. This is the default.

  User — A literal that displays only the tied floating signals or pins created using the add_tied_signals command.

  System — A literal that displays only the netlist-described tied floating signals or pins.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example displays the tied floating signals from the user class:

```
add_tied_signals 1 vcc vdd
report_tied_signals -class user
```

Related Topics

add_tied_signals

delete_tied_signals

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_black_boxes

set_tied_signals

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_timeplates

Context: dft -edt, dft -scan, dft -test_points, patterns -ijtag, patterns -scan, patterns -scan_retargeting, patterns -failure_mapping

Mode: setup, analysis

Displays the specified timeplate.

Usage

report_timeplates {***timeplate_name*** | -**All** } [-COre core_name [-MOde mode_name]] [{> | >>} *file_pathname*]

Description

Displays the specified timeplate.

The report_timeplates command displays all timeplates or the specified timeplate.

Arguments

- ***timeplate_name***

  A string that specifies which timeplate to display.

- -**All**

  A switch that specifies for the tool to display all timeplates. This is the default.

- -COre *core_name*

  An optional switch and string pair that specifies that the tool reports the timeplate for a given core. The core must exist, or an error is issued.

- -MOde *mode_name*

  An optional switch and string pair that specifies that the tool reports the timeplate for the given mode of a core defined by the -core *core_name* switch. The tool reports an error if this switch is specified without the -core switch. This switch is not necessary when there is only one mode for a core. If a core has more than one mode, an error is issued if the mode is not specified.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

Example 1

The following example displays the timeplate named *tp1*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_timeplates tp1

Example 2

The following example shows using the -EXpand_iprocs switch to work with the set of used ports and vector callbacks in:

ANALYSIS> report_timeplates test_setup -expand_iprocs

```
procedure test_setup =
  timeplate tp1 ;
  iReset;
  // Begin expanded iCall m8051_B1_edt_i.setup edt_low_power_shift_en on ;
  // cycle 1 start at time 40
  cycle =
    annotate "+ Targets: …"
     …
    end;
  // End expanded iCall
  // cycle 5 starts at time 200
  cycle =
    …
  end;
end;
```

Example 3

This is an example of using report_timeplates with the -core switch:

report_timeplates -core piccpu_maxlen16_1

```
timeplate gen_tp1 =
    force_pi 0 ;
    measure_po 100 ;
    pulse core_1_clk 200 100;
    pulse core_1_edt_clock 200 100;
    pulse core_1_ramclk 200 100;
    period 400 ;
end;
```

Related Topics

add_scan_groups

read_procfile

report_procedures

write_patterns

write_procfile

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_udfm_statistics

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: analysis

Reports detailed fault coverage based on the UDFM defect models.

Usage

report_udfm_statistics [-INstance *pathname*...] [-Groups *name* …]
[-GROUPS_Only {On | Off}] [-Percent {On | Off}] [-SET_Baseline]
[-Udfm_type *udfm_type*] [-CEll *cell_name*] [-Module *module_name*] [-Fault *fault_name*]
[(> | >>) *filename*]

Description

The report contains columns for DS, DI, PT, TI, BL, RE, AU, UC, UO, UU, PU faults. It also contains the following information as well as any additional columns you specify.

If no group is defined as a result of executing the command, by default every defect model is reported in a single row. The column headings are as follows:

- **Type** — Lists the UDFM type name that belongs to the defect model. For groups that collect defect models from multiple UDFM types, “---” is reported.
- **Block** — Identifies the object the defect model is defined on: Module, Cell, or Instance. For groups that collect different defect models, “---” is reported.
- **Name** — Lists the defect specific module, cell or instance name. For groups, the specified string is reported.
- **Instances** — Lists the number of instances of the specified module, cell, or instance identified in the design. This number is always greater zero because defect models that cannot be matched to an instance of the circuit are not reported.
- **Port-Insts** — Lists the number of instances where only port faults are active (that is, these instances have no cell internal faults). The information enables you to see if specific defects other than traditional port faults are targeted. The numbers in this column mean the following:
  - Port-Insts=0 — The UDFM definition contains fault definitions for cell internal defects. This means that not just port faults are targeted.
  - Port-Insts=Instances — When these two columns list the same number, the UDFM definition contains fault definitions for port faults only. This can be the case when fault sites have been added by using the add_fault_sites command.
  - Port-Insts > 0 and < Instances — Even if the UDFM definition contains a fault definition for cell internal defects, the cell-internal defects may have been dropped from the count because the flip-flop isn't part of a scan chain.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **Faults** — Lists the number of faults that exist due to the specified defect model or group.
- **Add-DT** — Lists the difference between the stored baseline and the current status for detected faults (DT). This column displays only if there was a previous call of this function to store the baseline.
- **GN** — Lists the coverage gain between the stored baseline and the current status with GN=(Add-DT/(DS+DI+PT+AU+UC+UO+PU)*100). This column displays only if there was a previous call of this function to store the baseline.

Arguments

- -INstance *pathname* ...

  An optional switch and repeatable string that limits the report of UDFM statistics to one or more specific instances. The pathname argument specifies the name of a circuit block.

- -Groups *name* ...

  An optional switch and repeatable string that defines a group of UDFM defect models based on the specified *name* string(s). You can specify multiple name arguments to create multiple groups. Each *name* string creates one group. The *name* string is case insensitive and may include any number of asterisk (*) and question mark (?) wildcard characters.

  For example, the following string creates two groups: one with all defect models that contain an “AND” in their name and another with all defect models that contain an “OR” in their name.

  ```
  -Groups *AND* *OR*
  ```

  All matching models are collected regardless of whether they are a module, a cell, or an instance. Note that in the case of multiple group names, the command adds individual models to only one group, which is the first group name in the list that matches a particular model.

- -GROUPS_Only On | Off

  An optional switch and literal pair that specify how UDFM defect models that do not match any of the defined groups are reported. By default, all UDFM defect models that do not match any of the defined groups are reported in a single row. If -GROUPS_Only is set to On, this row is not created.

- -Percent On | Off

  An optional switch and literal pair that specify to report individual fault classes as an integer or as a percentage. By default, the values of the individual fault classes are reported as a percentage. When this switch is set to off, fault classes are reported as an absolute number.

- -SET_Baseline

  An optional switch that specifies for the tool to store the current coverage information as a baseline so that subsequent report_udfm_statistics commands can add information about the coverage gain. Note that the baseline becomes invalid if the fault list is modified, for

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

example, by an add_faults or delete_faults command. The baseline also becomes invalid if you subsequently specify a different pathname with the -Instance switch.

- -Udfm_type *udfm_type*

  An optional switch and string pair that specifies the name of a UDFM type. Use this option to report detailed fault coverage associated with a specific UDFM type. The name string is case insensitive and can include any number of asterisk (*) and question mark (?) wildcard characters.

- -CEll *cell_name*

  An optional switch and string pair that specifies the name of a library cell. Use this option to report detailed fault coverage associated with a specific library cell. The name string is case insensitive and can include any number of asterisk (*) and question mark (?) wildcard characters.

- -Module *module_name*

  An optional switch and string pair that specifies the name of a module. Use this option to report detailed fault coverage associated with a specific module. The name string is case insensitive and can include any number of asterisk (*) and question mark (?) wildcard characters.

- -Fault *fault_name*

  An optional switch and string pair that specifies a UDFM fault name. Use this option to report detailed fault coverage associated with a specific fault. The name string is case insensitive and can include any number of asterisk (*) and question mark (?) wildcard characters.

Examples

Example 1

In this example, the command generates a detailed fault coverage report based on the UDFM defect models. The report contains columns for DS, DI, PT, TI, BL, RE, AU, UC, UO, UU, PU.

report_udfm_statistics

Note

Please note that due to space some fault class columns in the report have been removed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
UDFM Fault Statistic
Type   Block Name Instances Port-Insts Flts DS[%] DI[%] PT[%] TI[%] RE[%] AU[%]  UC[%] FC[%]
TC[%]
------ ----- ---- --------- ---------- ---- ----- ----- ----- ----- ----- ----- ------ -----
-----
intrac Cell  OR         518         0   5000  0.00  0.10  0.00  0.00  0.00  0.20  99.70  0.10
0.10
intrac Cell  NOR        139         0   1300  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00
0.00
intrac Cell  XNOR       346         0  12000  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00
0.00
intrac Cell  NAND3      387         0   3800  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00
0.00
intrac Cell  NAND4       71         0    600  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00
0.00
intrac Cell  AND2       913         0  10000  0.00  0.00  0.00  0.00  0.00  1.10  97.60  0.00
0.00
intrac Cell  AND3        27         0    300  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00
0.00
intrac Cell  AND4        19         0    400  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00
0.00
intrac Cell  AO112        4         0    100  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00
0.00
             Total     2424         0  33500  0.00  0.02  0.00  0.00  0.00  0.36  99.19  0.02
0.02
```

Example 2

In this example, the command generates a fault coverage report in which all UDFM fault models that contain the “NAN” string in their name are reported as a single row and all that contain the “AND” string in their name are reported as a single row. All others are reported one row per model.

report_udfm_statistics -groups *NAN* AND*

```
UDFM Fault Statistic
Type   Block Name  Instances Port-Insts Faults DS[%] DI[%] PT[%] TI[%] RE[%] AU[%] UC[%]
FC[%] TC[
------ ----- ----  --------- ---------- ------ ---- ----- ----- ----- ----- ----- ----- ----
- --
intrac Cell  OR          518          0   5000  0.00  0.10  0.00  0.00  0.00  0.20  99.70
0.10  0.10
intrac Cell  NOR         139          0   1300  0.00  0.00  0.00  0.00  0.00  0.00 100.00
0.00  0.00
intrac Cell  XNOR        346          0  12000  0.00  0.00  0.00  0.00  0.00  0.00 100.00
0.00  0.00
intrac Cell  AO112         4          0    100  0.00  0.00  0.00  0.00  0.00  0.00 100.00
0.00  0.00
intrac Cell  *NAN*       458          0   4400  0.00  0.00  0.00  0.00  0.00  0.00 100.00
0.00  0.00
intrac Cell  AND*        957          0  10700  0.00  0.00  0.00  0.00  0.00  1.09  97.60
0.00  0.00
             Total       2424          0  33500  0.00  0.02  0.00  0.00  0.00  0.36  99.19
0.02  0.02
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

The following example first sets a baseline for later reporting the fault coverage gain. The output of the second command displays two additional columns on the right that list the coverage gain.

report_udfm_statistics -set_baseline
...
report_udfm_statistics

```
UDFM Fault Statistics
//
Type         Block Name  Instances Port-Insts Faults  DS[%] ...  TC[%] Add-DT GN[%]
------------ ----- ----- --------- ---------- ------ ------     ------ ------ -----
intra_cell   Cell AO          1801          0 142918  99.36      99.91   5162  3.63
intra_cell   Cell MUXI4        332          0  35856 100.00     100.00   1514  4.22
intra_cell   Cell XOR3          50          0   2350  98.89     100.00     21  0.90
intra_cell   Cell AND          959          0  11660  97.64      98.96     19  0.17
intra_cell   Cell BF          1653       1653   8265  74.41      95.06     11  0.13
...
intra_cell   Cell MUXI2        204          0   4360  99.27      99.63      9  0.21
intra_cell   Cell IV          1356       1356   6780  88.79      98.02      9  0.13
intra_cell   Cell OR           515          0   7745  92.46      98.49      1  0.01
port_defects Cell DF           864         13   5220   1.02      94.61      0  0.00
                  Total      33886       3022 724779  87.71      95.69  10763  1.50
```

Example 4

The following example reports only faults with a name matching “sa*_*”. You can use this to report coverage data for only stuck port faults, which are added with a naming convention of “sa0_<*cell_port_name*>” or “sa1_<*cell_port_name*>” by the Tessent CellModelGen tool (see the section “The Cell-Aware UDFM File” in the *Tessent CellModelGen Tool Reference*) or the add_fault_sites command.

report_udfm_statistics -fault sa*_*

```
UDFM Fault Statistics

Type                  Block     Name   Instances  Port-Insts  Faults  DS[%] ... TC[%]
--------------------  --------  -----  ---------  ----------  ------  -----     -----
auto-inserted-faults  Instance  /              1           1      10   0.00     70.00
intra_cell_defects    Cell      sff            3           3      36   0.00     70.00
                                Total          4           4      46   0.00     70.00
```

Similarly, the following example reports only faults with a name matching "tr*_*". you can use this to report coverage data for only transition port faults, which are added with a naming convention of “tr0_<*cell_port_name*>” or “tr1_<*cell_port_name*>” by the Tessent CellModelGen tool or the add_faults command.

report_udfm_statistics -fault tr*_*

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
UDFM Fault Statistics

Type                  Block     Name   Instances  Port-Insts  Faults  DS[%] ... TC[%]
--------------------  --------  -----  ---------  ----------  ------  -----     -----
auto-inserted-faults  Instance  /              1           1      10   0.00     80.00
intra_cell_defects    Cell      or02           2           2      12   0.00     0.00
intra_cell_defects    Cell      sff            3           3      36   0.00     80.00
                                Total          6           6      58   0.00     61.54
```

Related Topics

write_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_version_data

Context: unspecified, all contexts

Mode: all modes

Displays the current software version information.

Usage

report_version_data [{> | >>} *file_pathname*]

Description

The report_version_data command displays information relating to the software title, version, and date.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following is an example of the output this command:

report_version_data

```
Version data: Tessent Shell 2012.3   Wed Sep 05 21:31:33 GMT 2012
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_wrapper_cells

Context: dft -scan

Mode: analysis

Reports information about the identified wrapper cells for each I/O port subjected to wrapper cell identification.

Usage

report_wrapper_cells [-Summary | -Verbose]

Description

If you specify no arguments, the tool reports the following information for each I/O:

- Primary I/O port name and its direction (input or output).
- Total number of shared wrapper cells identified for this I/O.
- Whether the I/O's wrapper cell is a dedicated wrapper cell or an existing (shared) cell.
- If shared wrapper cell identification failed, the reason for the failure. For a description of all possible reasons see Table 7-8.

If you specify -Verbose, in addition to the above information, the tool reports the following:

- Shared and dedicated wrapper cell path names for each wrapped port.
- Wrapper cell type (input or output).
- Clocks controlling the wrapper cells.

For ports with identified shared wrapper cells, the reason column can have the following values:

- Scan_element feedback — feedback of shared wrapper cells that are reached directly from the port.
- Clock gater fanout — a shared wrapper cell in fanout of a clock gater whose functional enable pin is reached from the given port. Only one such cell is reported per clock gater. It is also the cell where faults in the functional enable cone of logic of that clock gater are observed.
- Clock gater feedback — a shared wrapper cell in a functional enable pin feedback of a clock gater whose functional enable pin is reached from the given port.

  For more details on interaction between clock gater and wrapper analysis, refer to Clock Gaters and Wrapper Analysis in the *Tessent Scan and ATPG User's Manual.*

If you specify -Summary, the tool reports the following:

- Total number of primary inputs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Total number of primary outputs.
- Total number of identified shared wrapper cells.
- Total number of added dedicated wrapper cells.

Table 7-8. Reason for Failed Identification

Reason Failed Identification	Description
Max Logic Level	Exceeded the integer limit on the number of combinational logic levels between this PI/PO and the first level of sequential cells. If this happened during feedback tracing, then “Max Logic Level (feedback)” is reported.
Combinational Feed-thru	Encountered no sequential cells or combinational logic other than buffers/inverters before reaching PO/PI: port_name.
Combinational Logic Only	Encountered combinational logic only before reaching PO/PI: port_name.
Black Box Instance	Encountered a blackboxed instance: instance_name.
Interface-only Module	Encountered an instance of interface-only module: instance_name.
Pre-Existing Scan Cell	Encountered a sequential cell belonging to a pre-existing scan chain: instance_name.
Input/Output Wrapper Cell	Encountered a sequential cell that has specified_wrapper_type attribute set to input/output.
Max Identified Cells	Exceeded the integer limit on the number of sequential cells identified per PO/PI. If this happened during feedback tracing, then “Max Identified Cells (feedback)” is reported.
Scan-Related	A PI/PO that is connected to a global scan enable.
Internal Scan Only	Reached a scan segment with is_internal_scan_only attribute set to “true”.
Inferred Dedicated Wrapper Cell	While tracing fanout of a primary input, encountered an output port that was inferred with a dedicated wrapper cell.
Non-scan Depth Threshold	Exceeded the threshold of non-scan cells during tracing.
Unconnected	The PI/PO has fanouts/fan-ins that are unconnected or tied to an X value.
Blocked	The PI/PO has fanouts/fan-ins that are blocked or tied to a non-X value.
Unconnected in Library Cell	The PI/PO is connected to a library cell through a port that is unconnected inside the library cell.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 7-8. Reason for Failed Identification (cont.)

Reason Failed Identification	Description
Reached Scan Resource Instrument	The PI/PO is connected to an ICL instrument with the keep_active_during_scan attribute set to “true”.
SW7 Violations	The tool was unable to cover any of the branches of this PI/PO with dedicated wrapper cells due to SW7 violations.

Arguments

- -Summary

  An optional switch. For information about this switch, refer to the preceding description.

- -Verbose

  An optional switch. For information about this switch, refer to the preceding description.

Examples

Example 1

The following example shows the output generated when the report_wrapper_cells command is run without the -verbose or -summary switch.

report_wrapper_cells

```
> report_wrapper_cells
---------------------------------------------------------------------------
Primary I/O  # Shared Wrapper Cells      Shared/       Reason
Port         (Direct/Internal-Feedback)  Dedicated
             Identified [32/32]          Wrapper Cell
---------------------------------------------------------------------------
'po' (O)     1                           shared        --
'pi' (I)     1/0                         shared        --
---------------------------------------------------------------------------
```

Example 2

The following example shows the output generated when the report_wrapper_cells command is run with the -summary switch.

report_wrapper_cells -summary

```
> report_wrapper_cells -summary
-------------------------------------------------------
Inputs  Outputs  Shared Wrapper Cells  Inferred Dedicated
(#)     (#)      Identified (#)        Wrapper Cells (#)
-------------------------------------------------------
17      5        3                     0
-------------------------------------------------------
```

Example 3

The following example shows a snippet of the output generated when the report_wrapper_cells command is run with the -verbose switch.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_wrapper_cells -verbose

```
> report_wrapper_cells -verbose-----------------------------------------------------
Primary I/O  # Shared Wrapper Cells     Wrapper  Wrapper  Clock   Shared/      Reason
Port         (Direct/Internal-Feedback) Cell     Cell             Dedicated
             Identified [32/32]         Name     Type             Wrapper Cell
-------------------------------------------------------------------------------------
'div_by_zero'  (O)  1     'fpu1/div_by_zero_reg'  output  'clk'     shared         --
'zero'         (O)  1     'fpu1/zero_reg'         output  'clk'     shared         --
'underflow'    (O)  1     'fpu1/underflow_reg'    output  'clk'     shared         --
'overflow'     (O)  1     'fpu1/overflow_reg'     output  'clk'     shared         --
'ine'          (O)  1     'fpu1/ine_reg'          output  'clk'     shared         --
'qnan'         (O)  1     'fpu1/qnan_reg'         output  'clk'     shared         --
-------------------------------------------------------------------------------------

The wrapper cells identification has failed for the following PIs and POs.
No dedicated wrapper cells will be added to these PIs and POs unless explicitly
specified by the user.
-----------------------------------------------------------------------------------------
'bidi1'                  output                        Bidi port
'bidi2'                  output                        Bidi port
'clk'                    input                         Clock port
'empty_out1'             output                        Blocked PO
'my_edt_clock'           input                         Unconnected PI
'scan_en                 input                         Blocked PI
'test_en'                input                         Constrained port
-----------------------------------------------------------------------------------------
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_write_controls

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Displays the currently defined write control lines and their off-states.

Usage

report_write_controls [{> | >>} *file_pathname*]

Description

The report_write_controls command displays the write control lines, with corresponding off-states, added using the add_write_controls command.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Examples

The following example adds four write control lines and then displays a list of the control line definitions:

```
add_write_controls 0 w1 w3
add_write_controls 1 w2 w4
report_write_controls
```

Related Topics

add_write_controls

analyze_control_signals

delete_write_controls

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_write_patterns_options

Context: patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: setup, analysis

Displays the current set of ports or current set of vector callbacks to use when writing patterns.

Usage

Usage for Port Lists

report_write_patterns_options [-existing_used_ports | -additional_port_list | -all_used_port_list] [-continuous_expect_during_payload]

Usage for Vector Callbacks

report_write_patterns_options [-vector_callbacks]

Description

Use the report_write_patterns_options command to display the current set of ports or vector callbacks.

Port List Specifics

This command and options display the current set of ports to use when writing patterns. The report is a tabular format that lists the port names, port direction if the port is an additional port, and whether the port is a clock and, if so, its off-state. If no subset exists, the tool reports that there is no subset, but it then provides the full list of default ports. The report includes any expect ports in a table at the end. The -continuous_expect_during_payload option displays only this table, or a message if none exist.

If you run the report_write_pattern_options command without any options, it reports all write_pattern_options settings (the used ports and the vector callbacks).

Vector Callback Specifics

This command reports the current vector callback procs listing each callback proc name followed by a textual representation of any activation trigger. If the report_write_pattern_options command is issued without any options, all write_pattern_options settings are reported (the used ports and the vector callbacks).

Arguments

- -existing_used_ports

  An optional switch that reports only the current subset of the existing design ports.

- -additional_port_list

  An optional switch that reports only the additional ports that were not part of the original design ports.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -all_used_port_list

  An optional switch that reports the current subset of existing design ports and any extra ports added with -additional_ports.

- -continuous_expect_during_payload

  An optional switch that reports the current set of continuous expect ports the tool is using when writing patterns. The report appears as a table listing the port names and expect values. If no continuous expect ports exist, the report provides an explanatory message instead of a table.

- -vector_callbacks

  An optional switch that reports the current vector callback procs.

Related Topics

get_write_patterns_options

set_write_patterns_options

write_patterns

Vector Creation and Modification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_xbounding

Context: dft -scan, dft -test_points

Mode: analysis

Reports the X-sources and optionally the scan cells used in bounding. You must run this command to report X-bounding.

Usage

report_xbounding {[-verbose [all | non_mcp_bounding_points | mcp_bounding_points]]} [-ignored_x_sources { on | off }] [> | >> *file_pathname*]

Description

By default, this command reports a summary that includes the number of identified X-sources and the number of control points that are required to block these X-sources from reaching any scan cells (or potential scan cells). You can report the location of the X-bounding logic (typically muxes) and the control signals for the bounding logic.

Arguments

- -verbose all | non_mcp_bounding_points | mcp_bounding_points

  An optional switch and optional literal that reports the location of the scan cells used in the bounding.

  Choose one from the following:

  all — Reports all scan cells used in the bounding including non-MCP and MCP bounding points. This is the default.

  non_mcp_bounding_points — Reports only non-MCP bounding points.

  mcp_bounding_points — Reports only MCP bounding points.

- -ignored_x_sources { on | off }

  An optional switch and literal that specifies to report X-sources that do not need to be bounded, and the reasons why they are not bounded. The default is off.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example shows the tool's verbose output:

```
report_xbounding -verbose
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Analyzed 52 static X-sources and 1 flop that captures false/
//  multi-cycle paths.
//  Added 53 control points for X-bounding.
//     52 control points bounding static X-sources.
//     1 control point bounding false or multi-cycle paths.
MUX gate at /NEA(3) bounding mux is driven by existing flop at : /u12/
L_MSIZ_reg_7_/Q (326)
MUX gate at /NESFR(4) bounding mux is driven by existing flop at : /u12/
u1/L_SP_reg_0_/Q (323)
```

The following example reports new scan cells that the tool inserted for X-bounding:

set_xbounding_options -connect_to new_scan_cell
set_xbounding_options

```
Xbounding enable name           : lbist_en
Connect to                      : new scan cell
New scan cell model name        : default
Scan drive limit                : 1
Exclude sdc cross domain paths  : OFF
MCP bounding enable name        : lbist_en
Use test clock                  : OFF
Exclude pin or port list : empty
```

analyze_xbounding

```
//  Warning: DRC handling for E11 rule has been set to ignore.
//           Potential X-sources due to
//           tristate bus float will not be detected.
//  Scan enables:  scan_en discovered.
//  ---------------------------------------------------------------
//  Begin X bounding.
//  ---------------------------------------------------------------
//  43 potentially observable X sources found.
//  4 TIEXs.
//  Analyzed 43 static X-sources.
//  Added 41 control points for static X-bounding.
//  Marked 9 previously observable gate pins with no_observe_point
//  attribute to avoid inserting observe points that would capture
//  an unblocked X-source.
```

report_xbounding -verbose

```
//  Analyzed 43 static X-sources.
//  Added 41 control points for static X-bounding.
//  Marked 9 previously observable gate pins with no_observe_point
//  attribute to avoid inserting observe points that would capture
//  an unblocked X-source.
// MUX gate at /CO(1)
//      Driven by new scan element /ts_0_default_PD_default1_xb_172sffp1_i
//      Bounds static X-source '/CO' (1)
// MUX gate at /ACO(2)
//      Driven by new scan element /ts_0_default_PD_default1_xb_175sffp1_i
//      Bounds static X-source '/ACO' (2)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

analyze_xbounding

insert_test_logic

read_sdc

set_xbounding_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_xy_instance

Context: patterns -scan_diagnosis

Mode: analysis

Returns a list of cell instances located at the specified location in the LDB.

Usage

report_xy_instance ***x y*** [-ldb_units]

Description

This command returns a list of instance names that overlap the specified x and y coordinates, as stored in the LDB. You must provide chip-level coordinates, which the tool automatically converts to core-level coordinates if a chip-mapped core-level LDB is currently open. When using a chip-mapped core-level LDB, you must set the layout core instance before invoking this command.

By default, the command assumes the specified coordinates are expressed in microns units. You can override the default by specifying the -ldb_units switch.

Arguments

- ***x y***

  Required x and y chip-level coordinates within the LDB. The default is microns.

- -ldb_units

  An optional switch that enables you to specify the coordinates in LDB units rather than the default micron units.

Related Topics

get_instance_polygons

get_xy_instance

report_instance_polygons

report_net_polygons

set_layout_core_instance

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_xy_net

Context: patterns -scan_diagnosis

Mode: analysis

Returns a list of nets located at the specified location in the LDB.

Usage

report_xy_net ***x y*** [-ldb_units] [-report_polygons]

Description

This command returns a list of net names that overlap the specified x and y coordinates, as stored in the LDB. You must provide chip-level coordinates, which the tool automatically converts to core-level coordinates if a chip-mapped core-level LDB is currently open. When using a chip-mapped core-level LDB, you must set the layout core instance before invoking this command.

Note

This command only returns results for verbose LDBs (non-compact).

By default, the command assumes the units of the specified coordinates are expressed in microns. You can override the default by specifying the -ldb_units switch.

Arguments

- ***x y***

  Required x and y chip-level coordinates within the LDB. The default is microns.

- -ldb_units

  An optional switch that enables you to specify the coordinates in LDB units rather than microns which is the default.

- -report_polygons

  An optional switch that prints the polygons of the nets which overlap the specified x/y coordinates. (Not all polygons for the entire net are reported).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reset_attribute_value

Context: unspecified, all contexts

Mode: all modes

Resets an attribute to its default value for the design objects specified in *obj_spec*.

Usage

reset_attribute_value { ***obj_spec*** | **-object_types *type_list***} **-name *attribute_name*** [-silent]

Description

Resetting an attribute differs from setting it to its default value in the fact that the reset_attribute_value also makes the attribute unspecified so it does not appear when you run the report_attributes and get_attribute_list commands using Usage 1. The attribute's default value is specified during registration of the attribute.

This command returns the default value set for the attribute as a string.

If the attribute_name does not exist, or if a design object or object type does not exist, an error message displays unless -silent is specified.

Arguments

- ***obj_spec***

  A switch that specifies a Tcl list of one or more object names or a collection of one or more objects.

- **-object_types *type_list***

  A switch that specifies a list of object types for which the given attribute name is to be reset. As an example, using “reset_attribute_value -name XXX -object_type instance” is equivalent to specifying “reset_attribute_value -name XXX [get_instances]” but is more efficient and requires less memory and time when the number of objects for the given object type is large.

- **-name *attribute_name***

  A required switch and value pair that specify the name of the attribute.

- -silent

  An optional switch that specifies to suppress error messages if the attribute_name does not exist, or if an object in *obj_spec* does not exist.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example shows how the command resets the added_by values for all nets to its default (“John”):

```
reset_attribute_value {u1/net1 u1/net2 u2/net3 u2/net4} -name added_by
get_attribute_value_list {u1/net1 u1/net2 u2/net3 u2/net4} -name added_by

{John John John John}
```

Example 2

The following example resets the my_constraint attribute on all pins and nets.

reset_attribute_value -name my_constraint -object_types {pin net}

Related Topics

get_attribute_list

get_attribute_option

get_attribute_value_list

report_attributes

set_attribute_options

set_attribute_value

unregister_attribute

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reset_au_faults

Context: dft -edt, patterns -scan

Mode: analysis

Resets the faults in certain untestable categories by changing their classifications.

Usage

reset_au_faults [-REClassify]

Description

The following table shows the untestable categories for which this command changes the fault classifications. Refer to "Fault Classes" in the *Tessent Scan and ATPG User's Manual* for further information about fault classification.

Table 7-9. Untestable Faults that are Reclassified by reset_au_faults

Untestable Fault	Fault Classification After Reset
Possibly detected untestable (PU)	Possibly detected testable (PT)
ATPG untestable (AU) except for AU.LBIST (not reclassified) **Note:** You can have certain AU faults reclassified back to AU after this command resets their classifications. Use the -reclassify switch to do this. Refer to the argument description for more details.	Uncontrolled (UC)

Deterministic fault simulators classify some untestable faults differently depending on the algorithm. You can use the reset_au_faults command to align those (potentially) misclassified faults.

When this command changes the fault classification, the tool then has the ability to analyze and further reclassify each previously untestable fault. Enabling the tool's ability to analyze and reclassify those particular untestable faults increases the tool's efficiency.

Arguments

- -REClassify

  An optional switch that analyzes the AU faults to identify those that the tool cannot test with the current settings. After resetting all AU faults (except for AU.LBIST, which is not initially reclassified to UC), the tool reclassifies faults that are still untestable in the current mode back to AU. This switch is useful when reading faults from a previous ATPG run that has different settings or a different design configuration. Reclassifying the faults to AU before pattern generation improves performance. If you omit this switch, the tool does not reclassify the faults and resets all AU faults. See Example 2.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

See Fault Sub-Classes in the *Tessent Scan and ATPG User's Manual* for more information.

Examples

Example 1

The following example sets up the tool to run the simulation with an external pattern file and resets the ATPG untestable faults so that the tool can determine their appropriate fault category:

```
read_patterns testpatterns
read_faults /user/design/fault_file -retain
reset_au_faults
simulate_patterns
```

Example 2

The following example incrementally runs ATPG in two different modes, combining the results of each session, to target additional undetected faults and increase coverage.

Dofile 1

```
// set up design to run ATPG in mode 1
add_scan_groups grp1 testproc
add_scan_chains chain1 grp1 scan_in1 x_out[0]
add_scan_chains chain2 grp1 scan_in2 scan_out1
add_scan_chains chain3 grp1 scan_in3 scan_out2
add_clocks 0 rst
add_clocks 0 clk
add_input_constraints /G -C1

// generate patterns from mode 1 and write out faults
set_system_mode analysis
create_patterns
write_faults mode1.fault -replace
```

Dofile 2

```
// set up design to run ATPG in mode 2
add_scan_groups grp1 testproc
add_scan_chains chain4 grp1 scan_in4 y_out[2]
add_scan_chains chain5 grp1 scan_in5 x_out[5]
add_clocks 0 rst
add_clocks 0 clk
add_input_constraints /G -C1

// generate patterns from mode 2 and write out faults
set_system_mode analysis
read_faults mode1.fault -retain
reset_au_faults -reclassify
create_patterns
write_faults modes_1_and_2.fault -replace
```

Related Topics

read_faults

reset_di_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reset_bypass_chains

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Removes user-specified EDT bypass chain connections.

Usage

reset_bypass_chains

Description

The reset_bypass_chains command removes all EDT bypass chain connections that you previously defined using the set_bypass_chains command, and reverts determination of these connections to the tool.

Arguments

None

Examples

The following example displays information on all user-defined EDT bypass chains in a design with eight scan chains and two scan channels. The example then removes the definitions, and confirms that configuration of any EDT bypass chains are now done automatically by the tool.

report_bypass_chains

```
//    Reporting user-defined bypass chains:
//
//    Bypass#  EDT chains
//    -------  ----------
//    1        chain1, chain5, chain6
//    2        chain2, chain3, chain4, chain7, chain8
```

reset_bypass_chains
report_bypass_chains

```
//  No user-defined bypass chains to report.
```

Related Topics

report_bypass_chains

set_bypass_chains

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reset_capture_timing_options

Context: all contexts

Modes: all modes

Resets all, or a specified SSN scan operation capture procedure timing option to its default value and clears the associated is_specified flag.

Usage

reset_capture_timing_options {[*resetOption*] | [-all]}

Description

Enables you to reset all, or a specific timing option of the capture procedure timing during SSN scan operations to the default value. The command also clears the associated is_specified flag set by a previous set_capture_timing_options command. If you do not specify a specific timing option, the tool resets all capture timing options to their default values and clears the associated is_specified flags.

The is_specified flag is set by a previous set_capture_timing_options command on that option. You can examine the is_specified flag for a specific capture timing option with the "get_capture_timing_option *timing_option* -is_specified" command.

Arguments

- *resetOption*

  An optional switch that specifies a timing option of the capture procedure to reset to default and clear its associated is_specified flag. This switch is mutually exclusive with the -all switch. Choose a timing option from the arguments listed for the set_capture_timing_options command.

- -all

  An optional default switch that resets all timing options for the capture procedures of SSN scan operations to their default value and clears the associated is_specified flag. This switch is mutually exclusive with the *resetOption* switch.

Related Topics

get_capture_timing_option

report_capture_timing_options

set_capture_timing_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reset_compactor_connections

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Removes user-specified EDT compactor connections.

Usage

reset_compactor_connections

Description

The reset_compactor_connections command removes all EDT compactor connections that you previously defined using the set_compactor_connections command, and reverts determination of these connections to the tool.

Arguments

None

Examples

The following example displays information on all user-defined compactor connections in a design with eight scan chains and two scan channels. The example then removes the user-specified definitions. Configuration of compactor connections are now handled automatically by the tool.

report_compactor_connections

```
//    Reporting user-defined compactor connections:
//
//    Channel  Chains
//    -------  ------
//    1        chain1, chain2, chain6, chain7, chain8
//    2        chain3 - chain5
```

reset_compactor_connections
report_compactor_connections

```
//  No user-defined compactor connections to report.
```

Related Topics

report_compactor_connections

set_compactor_connections

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reset_default_ssn_datapath_configuration

Context: all contexts

Mode: setup

Deletes all override settings of an ICL network's default SSN datapath configuration.

Usage

reset_default_ssn_datapath_configuration

Description

Deletes all the custom override settings of the default values of an ICL network.

Refer to the command "set_default_ssn_datapath_configuration" on page 2441 to learn more about the purpose and the use model of the default SSN datapath configuration and how set_default_ssn_datapath_configuration assigns settings to the PDL values of ICL targets to override their default state.

Arguments

- None.

Examples

The following example sets two PDL values of two ICL targets and then resets them to their default value.

```
SETUP> set_default_ssn_datapath_configuration -target cfginst.din -value 0b1101xx10
SETUP> set_default_ssn_datapath_configuration -target ssnMuxCtrl -value Upper
SETUP> report_default_ssn_datapath_configuration

//  target       value
//  ===========  ==========
//  cfginst.din  0b1101xx10
//  ssnMuxCtrl   Upper
SETUP> reset_default_ssn_datapath_configuration
SETUP> report_default_ssn_datapath_configuration
SETUP>
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reset_design

Context: all contexts

Mode: setup

Clears most design setup data, including data from previous runs, but retains the design and library in memory.

Usage

reset_design

Description

TCD file-related data, scan path related data, all core instances, and pattern data are cleared. DFT signals, black box settings, and the design elaboration are deleted. Also, constraint settings and clocks are removed.

The command retains the setting from the most recent set_current_design command execution.

Using the reset_design command has the same effect as running the following commands:

- delete_dft_clock_muxes -all
- delete_dft_clock_enables -all
- delete_dft_modal_connections -all
- delete_dft_control_points -all
- delete_dft_signals -all_user_added -silent

Arguments

None

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example shows that clock and DFT signals are cleared with the reset_design command when there are no design- or IJTAG-defined DFT signals:

```
set_current_design your_design
set_design_level physical_block

set_dft_specification_requirements -memory_test on
add_clocks clock -period 20ns

add_dft_signals memory_bypass_en

reset_design

report_clocks
// No clocks have been defined.
report_dft_signals
// There are no existing nor to be created DFT signals. You can use the
//    'report_dft_signal_names' command to see the registered names.
```

Example 2

The following example shows that design- or IJTAG-defined DFT signals are not cleared with the reset_design command:

```
set_current_design your_design
set_design_level physical_block

set_dft_specification_requirements -memory_test on
add_clocks clock -period 20ns

add_dft_signals memory_bypass_en

reset_design

report_clocks
// No clocks have been defined.
report_dft_signals
// Existing and to be created DFT signals
// ======================================
// ------------------------------- -------- ------------------------ --
// Name Status Origin Type Location
// ------------------------------- -------- ------------------------ --
// edt_clock Existing Imported from DesignInfo Pin
//   top_rtl_insertion_tessent_lbist_inst/edt_lbist_clock
// edt_update Existing Imported from DesignInfo Port edt_update
// ext_mode ExistingICL attributes IJTAG register
//   top_rtl_insertion_tessent_tdr_sri_ctrl_inst/ext_mode
```

Related Topics

delete_design

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_current_design

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reset_di_faults

Context: dft -edt, patterns -scan

Mode: analysis

Reclassifies DI faults to the UC category.

Usage

reset_di_faults {**-ALL** | **-END_of_chains** | **-CHains** [***chain_name...***] | ***pin_pathname...***}
[-Stuck_at {01 | 0 | 1}] [-Both | -Rise | -Fall] [-CELL_Output_only]

Description

The reset_di_faults command reclassifies det_implication (DI) faults to the uncontrolled (UC) category. When the fault type is stuck-at or transition, this command by default resets both stuck-at-0 (or slow-to-rise for transition faults) and stuck-at-1 (or slow-to-fall for transition faults) faults at each DI fault site. The command does not reclassify DI faults identified by other Tessent plugins, such as DI.MBIST and DI.MBISR faults identified by Tessent MemoryBIST. You can use one of the optional switches to reset just one, rather than both, faults at each DI fault site.

Use this command when you want faults that were previously classified as DI (and thus no longer targeted for detection) retargeted in the next ATPG or fault simulation run. For example, the tools currently declare scan path faults DI because they are tested by the chain test. This is not optimal in some cases; the end of each scan chain (after the last scan cell) is often used as a system path and it is sometimes desirable to create an at-speed transition fault test to exercise this path. The DI fault category prevents ATPG from targeting or simulating such faults. When the fault classification is changed to UC, the tool then has the ability to target the faults during ATPG and fault simulation.

Arguments

- **-ALL**

  A switch that specifies for the tool to reset DI faults to UC. This is the invocation default.

- **-END_of_chains**

  A switch that specifies to reset, for each scan chain, the scan path DI faults from the output of the last scan cell in the chain to the scan output pin.

- **-CHains** [***chain_name...***]

  A switch and optional repeating string pair that resets DI faults, including scan input and output pin faults. If you do not specify a chain_name, the tool resets faults along all scan chains. You can optionally restrict the switch operation to one or more chain_names.

- ***pin_pathname***...

  A repeating string that resets the DI faults on one more pins. The ***pin_pathname*** must be an instance/pin name that exists in the flat model. These are typically model boundary pins, PIs, and POs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Stuck_at 01 | 0 | 1

  An optional switch and literal pair that specifies which DI faults to reset to UC. You should use this switch only if the fault type is stuck, although the tool accepts the switch regardless of fault type (see set_fault_type).

  01 — A literal specifying that for stuck-at faults the tool reset both the "stuck-at-0" and "stuck-at-1" faults; or for transition faults the tool reset both "slow-to-rise" and "slow-to-fall" faults. This is the default.

  0 — A literal specifying that for stuck-at faults the tool reset only the "stuck-at-0" faults; or for transition faults the tool reset only "slow-to-rise" faults.

  1 — A literal specifying that for stuck-at faults the tool reset only the "stuck-at-1" faults; or for transition faults the tool reset only "slow-to-fall" faults.

- -Both| -Rise | -Fall

  Optional switches that specify which transition faults to reset to UC. These switches apply to transition faults only:

  -Both - An optional switch that specifies to reset both slow-to-rise and slow-to-fall transition faults. This is the default.

  -Rise - An optional switch that specifies to reset only the slow-to-rise transition faults.

  -Fall - An optional switch that specifies to reset only the slow-to-fall transition faults.

- -CELL_Output_only

  Optional switch that reclassifies DI faults only on the scan output ports of scan cells.

Examples

Example 1

The following example resets all stuck-at-0 and stuck-at-1 DI faults (including clock faults) to UC.

```
reset_di_faults
```

Example 2

The following example resets all stuck-at-0 faults declared DI anywhere in the netlist.

```
reset_di_faults -stuck_at 0
```

Example 3

The following example resets stuck-at-0 and stuck-at-1 DI faults at the output of the last cell of each chain.

```
reset_di_faults -end_of_chain
```

Example 4

The following example resets only the slow-to-rise transition fault at the output of the last cell of each chain.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reset_di_faults -end_of_chain -rise

Example 5

The following example resets any DI fault anywhere along chain1.

reset_di_faults -chain chain1

Example 6

The following example resets both DI faults at pin /top/mid/bottom/Q (any type fault).

reset_di_faults /top/mid/bottom/Q -stuck_at 01

Example 7

The following example is equivalent to the preceding example, except that it applies to transition faults only.

reset_di_faults /top/mid/bottom/Q -both

Example 8

The following example uses the defaults to reset_di_faults at the specified location (any fault type).

reset_di_faults /top/mid/bottom/Q

Example 9

The following example resets only the slow-to-rise transition fault at the specified location.

reset_di_faults /top/mid/bottom/Q -rise

Example 10

The following example resets only the stuck-at-1 fault (or slow-to-rise if transition fault) at the specified location.

reset_di_faults /top/mid/bottom/Q -stuck_at 1

Related Topics

read_faults

reset_au_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reset_ijtag_retargeting_options

Context: all contexts

Mode: all modes

Resets all, or a specified IJTAG retargeting option to its default value and clears the associated is_specified flag.

Usage

reset_ijtag_retargeting_options {[*resetOption*] | [-all]}

Description

Enables you to reset a specific IJTAG retargeting option to its default value and clear the associated is_specified flag set by a previous set_ijtag_retargeting_options command. If you do not specify a specific retargeting option, the tool resets all retargeting options to their default values and clears the associated is_specified flags.

The is_specified flag is set by a previous set_ijtag_retargeting_options command on that option. You can examine the is_specified flag for a specific retargeting option with the get_ijtag_retargeting_options -is_specified switch.

Arguments

- *resetOption*

  An optional switch that specifies an IJTAG retargeting option to reset to default and clear its associated is_specified flag. This switch is mutually exclusive with the default -all switch. Choose one of the following IJTAG retargeting options:

  -test_setup_network_end_state — A switch that resets the value of the -test_setup_network_end_state option (keep or reset) specified by a previous set_ijtag_retargeting_options command to the default value (reset).

  -test_setup_ireset — A switch that resets the value of the -test_setup_ireset option (off, first_icall or when_icl_present) specified by a previous set_ijtag_retargeting_options command to the default value (when_icl_present).

  -iapply_target_annotations — A switch that resets the value of the -iapply_target_annotations option (off, dense, or full) specified by a previous set_ijtag_retargeting_options command to the default value (dense).

  -max_operations_per_iapply — A switch that resets the value of the -max_operations_per_iapply option specified by a previous set_ijtag_retargeting_options command to the default value (64).

  -compare_constant_capture_values — A switch that resets the value of the -compare_constant_capture_values option (on or off) specified by a previous set_ijtag_retargeting_options command to the default value (on).

  -compare_constant_data_out_ports — A switch that resets the value of the -compare_constant_data_out_ports option for the primary DataOut ports (on or off)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

specified by a previous set_ijtag_retargeting_options command to the default value (off).

-ignore_non_reset_end_state_implicit_compares — A switch that resets the value of the -ignore_non_reset_end_state_implicit_compares option (off, auto or on) specified by a previous set_ijtag_retargeting_options command to the default value (auto).

-constrain_update_en_only — A switch that resets the value of the -constrain_capture_en_only option (auto or on) specified by a previous set_ijtag_retargeting_options command to the default value (auto).

-default_scan_out_strobe_point — A switch that resets the value of the -default_scan_out_strobe_point option (before_rising_edge_of_tck, before_falling_edge_of_tck, or auto) specified by the previous set_ijtag_retargeting_options command to the default value (auto).

-interface_type — A switch that resets the value of the -interface_type option (auto, ijtag or tpsp) specified by a previous set_ijtag_retargeting_options command to the default value (auto).

-merge_irunloop_only — A switch that resets the value of the -merge_irunloop_only option (on or off) specified by a previous set_ijtag_retargeting_options command to the default value (off).

-network_end_state_warnings — A switch that resets the value of the -network_end_state_warnings option (on or off) specified by a previous set_ijtag_retargeting_options command to the default value (off).

-disable_pio_operations — A switch that resets the value of the -disable_pio_operations option (on or off) specified by a previous set_ijtag_retargeting_options command to the default value (off).

-ignore_iwrite_when_value_already_present — A switch that resets the value of the -ignore_iwrite_when_value_already_present option (on or off) specified by a previous set_ijtag_retargeting_options command to the default value (off).

-allow_skip_rti — A switch that resets the value of the -allow_skip_rti option (on or off) specified by a previous set_ijtag_retargeting_options command to the default value (on).

-inject_cycles — A switch that resets the value of the -inject_cycles option (Tcl list with an even number of arguments, each comprising a TAP state and the related number of injected cycles in alternating order) specified by a previous set_ijtag_retargeting_options command to the default of no injected cycles.

-tck_ratio — A switch that resets the value of the -tck_ratio option (auto or a power of 2) specified by a previous set_ijtag_retargeting_options command to the default value (auto).

-tck_period — A switch that resets the value of the -tck_period option specified by a previous set_ijtag_retargeting_options command to the default value (100ns).

-annotation_parameter_values — A switch that resets the value of the -annotation_parameter_values option (Tcl list with an even number of arguments, each comprising a key and a value in alternating order, describing the configuration

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

of notes, comments and annotations in Tessent IJTAG) specified by a previous set_ijtag_retargeting_options command to the default value.

- -all

  An optional default switch that resets all IJTAG retargeting options to their default value and clears the associated is_specified flag.

Related Topics

get_ijtag_retargeting_options

set_ijtag_retargeting_options

report_ijtag_retargeting_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reset_load_unload_timing_options

Context: all contexts

Modes: all modes

Resets all, or a specified SSN scan operation load/unload procedure timing option to its default value and clears the associated is_specified flag.

Usage

reset_load_unload_timing_options {[*resetOption*] | [-all]}

Description

Enables you to reset all, or a specific timing option of the load/unload procedure during SSN scan operations to the default value. The command also clears the associated is_specified flag set by a previous set_load_unload_timing_options command. If you do not specify a specific timing option, the tool resets all load/unload timing options to their default values and clears the associated is_specified flags.

The is_specified flag is set by a previous set_load_unload_timing_options command on that option. You can examine the is_specified flag for a specific load/unload timing option with the "get_load_unload_timing_option *timing_option* -is_specified" command.

Arguments

- *resetOption*

  An optional switch that specifies a timing option of the load/unload procedure to reset to default and clear its associated is_specified flag. This switch is mutually exclusive with the -all switch. Choose a timing option from the arguments listed for the set_load_unload_timing_options command.

- -all

  An optional default switch that resets all timing options for the load/unload procedures of SSN scan operations to their default value and clears the associated is_specified flag. This switch is mutually exclusive with the *resetOption* switch.

Related Topics

get_load_unload_timing_option

report_load_unload_timing_options

set_load_unload_timing_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reset_open_pattern_set

Context: patterns -ijtag

Mode: analysis

Clears the content of the currently open pattern set.

Usage

reset_open_pattern_set

Description

After this command executes, the pattern set remains opened. The state of the ICL network is restored to the state it was in at the beginning of the pattern set which is either the end state of the previously closed pattern set when the open_pattern_set -no_initial_ireset option is used or the reset state.

Arguments

None

Return Values

None

Examples

The following example clears the content of the currently open pattern set.

```
reset_open_pattern_set
```

Related Topics

close_pattern_set

get_open_pattern_set

get_pattern_set_list

open_pattern_set

report_pattern_sets

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reset_physical_block_settings

Context: dft -edt, dft -scan, dft -test_points, patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: setup, analysis

Clears the settings specified with the set_current_physical_block command.

Usage

reset_physical_block_settings
[-icl_modules *icl_module_names* | -modules *module_names* | -current_physical_block | -all]

Description

The reset_physical_block_settings command clears the settings specified with the set_current_physical_block command. It also affects the settings specified with the set_load_unload_timing_options, set_capture_timing_options, and set_ijtag_retargeting_options commands.

Arguments

- -icl_modules *icl_module_names*

  An optional switch and string pair that specifies the name of a valid ICL module that the tool will reset. You can specify more than one ICL module name.

- -modules *module_names*

  An optional switch and string pair that specifies the name of a module that the tool will reset. You can specify more than one module name.

- -current_physical_block

  An optional switch that specifies the current physical block that the tool will reset. This is the default.

- -all

  An optional switch that resets all physical blocks set with the set_current_physical_block command.

Examples

The following example resets the specified ICL module:

```
reset_physical_block_settings -icl_module xtea
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reset_state

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_diagnosis

Mode: analysis

Resets the circuit status.

Usage

reset_state

Description

Resets all faults, except AU, TI, BL, and RE faults, to be undetected in order to run a new simulation; and deletes the internal patterns.

dft -scan only:

Removes all instances from both the scan identification and test point identification lists that the tool identified.

Arguments

None

Examples

The following example first performs an ATPG run, then resets the circuit status, resets the faults to be undetected, deletes the internal patterns, and then performs a new ATPG run:

```
set_system_mode analysis
create_patterns
reset_state
create_patterns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reset_static_dft_signal_values

Context: dft patterns

Mode: setup, analysis

Resets the static DFT signal values specified with the set_static_dft_signal_values command.

Usage

reset_static_dft_signal_values
[-all | -current_design | -icl_instances *icl_instances* | -instances *instances*]
[{*dft_signal_name* ...}]

Description

Resets the static DFT signals that were previously set using the set_static_dft_signal_values command. All static DFT signals implemented with IJTAG can be set using this command. See the add_dft_signals command for information on how to add the DFT signals. The DFT signals are grouped by each module in which you added DFT signals and ran the process_dft_specification command. These modules have the tessent_design_level attribute equal to “chip”, “physical_block”, and “sub_block” in their extracted ICL module view.

When setting a static DFT signal within an instance, the other DFT signals without an explicit setting are assigned an implicit value following the logic documented in the set_static_dft_signal_values command description. The reset_static_dft_signal_values command is used to remove an explicit setting. If applicable, the implied settings are also removed.

Arguments

- -all

  An optional switch that removes the settings on the specified DFT signals found within all ICL instances that have the tessent_design_level attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition. The DFT signals found within the top module are also reset.

  The -all, -current_design, -icl_instances, and -instances switches are mutually exclusive. When unspecified, the -current_design switch is the default.

- -current_design

  An optional switch that removes the settings on the specified DFT signals found within top module. The top-level module must have the tessent_design_level attribute set to “chip”, “physical_block”, and “sub_block” within its ICL module definition.

  The -all, -current_design, -icl_instances, and -instances switches are mutually exclusive. When unspecified, the -current_design switch is the default.

- -icl_instances *icl_instances*

  An optional switch that removes the settings on the specified DFT signals found within the specified ICL instances. The specified ICL instances must have the tessent_design_level

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition.

The -all, -current_design, -icl_instances, and -instances switches are mutually exclusive. When unspecified, the -current_design switch is the default.

- -instances *instances*

  An optional switch that removes the settings on the specified DFT signals found within the specified design instances. The specified design instances must match the value of the “hierarchical_design_instance” attribute found on an ICL instance with the tessent_design_level attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition.

  The -all, -current_design, -icl_instances, and -instances switches are mutually exclusive. When unspecified, the -current_design switch is the default.

- *dft_signal_name*

  A repeatable set of strings used to specify DFT signal names to reset. The string can also be a single well-formatted Tcl list containing DFT signal names. If you specify no *dft_signal_name*, then the tool resets all DFT signals.

Examples

The following example shows the use of the report_static_dft_signal_settings command after a signal was set using the set_static_dft_signal_values and after the same setting was reset using the reset_static_dft_signal_values command.

```
set_context patterns -ijtag
read_icl ../sub_block_l3.icl
set_current_design sub_block_l3

set_system_mode analysis
set_static_dft_signal_values int_edt 1
report_static_dft_signal settings

// ICL Module      : sub_block_l3

// ---------------  -------------------  ---------  ----------
// DFT Signal Name  Usage                Set value  Set source
// ---------------  -------------------  ---------  ----------
// all_test         global_dft_control       1      Inferred
// ltest_en         logic_test_control       1      Inferred
// int_ltest_en     logic_test_control       1      Inferred
// ext_ltest_en     logic_test_control       -      -
// int_edt          scan_mode(internal)      1      Explicit
// ext_multi        scan_mode(external)      -      -
//

reset_static_dft_signal_values int_edt
report_static_dft_signal settings
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// ICL Module      : sub_block_l3

// ---------------  -------------------  ---------  ----------
// DFT Signal Name  Usage                Set value  Set source
// ---------------  -------------------  ---------  ----------
// all_test         global_dft_control       -      -
// ltest_en         logic_test_control       -      -
// int_ltest_en     logic_test_control       -      -
// ext_ltest_en     logic_test_control       -      -
// int_edt          scan_mode(internal)      -      -
// ext_multi        scan_mode(external)      -      -
```

Related Topics

add_dft_signals

get_static_dft_signal_icall

report_static_dft_signal_settings

set_static_dft_signal_values

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

restore_design

Context: dft

Mode: insertion

Discards all edits performed in insertion mode and restores the design to its state prior to entering insertion mode.

Usage

restore_design

Description

Use the restore_design command to discard all edits performed in insertion mode and to restore the design to its state prior to entering insertion mode.

Arguments

None

Examples

The following example creates a minor edit to the design, and then discards the edit:

```
SETUP> set_context dft -rtl
SETUP> read_verilog design1
SETUP> get_ports

{I1 I2}

SETUP> set_system_mode insertion
INSERTION> create_port I3

{I3}

INSERTION> get_ports

{I1 I2 I3}

INSERTION> restore_design
INSERTION> get_ports

{I1 I2}
```

Related Topics

write_design

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

run_synthesis

Mode: all contexts

Context: all modes

Runs a synthesis manager to perform synthesis of the test logic RTL.

Usage

run_synthesis [-design_name *design_name*] [-design_id *design_id*] [-generate_scripts_only]

[-synthesis_tool {dc_shell | genus}] [-compilation_options *option_list*]
[-pre_compilation_commands *command_list*]
[-post_compilation_commands *command_list*]
[-synthesis_output_directory *directory*] [-startup_file *filename*]
[-no_wait]

Description

The check_synthesis command monitors the synthesis results. Additionally, the set_run_synthesis_options command specifies the default values for options that can be set for run_synthesis command. The options that are defined with set_run_synthesis_options are overridden if they are explicitly defined with the run_synthesis command.

The run_synthesis command only synthesizes test logic RTL contained within the TSDB. During creation and insertion of memory BIST, boundary scan, or IJTAG logic, the generated RTL is automatically written to the TSDB during process_dft_specification. With EDT logic RTL, the write_edt_files with the -tsdb option is required to create a TSDB that can be used with run_synthesis.

When the run_synthesis command is used with a gate level design, a concatenated netlist of the design which contains the synthesized test logic and modified design modules is automatically created and placed in the dft_inserted_designs directory of the TSDB.

This command is typically run with no arguments. It automatically uses the design name and design id from the previously processed DftSpecification(*design_name*, *design_id*) wrapper.

The command dispatches and connects to a synthesis manager using a socket. If a manager is already running, it simply connects to it. The synthesis manager is an unlicensed Tcl shell executable that uses the content found in the *design_name_design_id.synthesis_dc_shell* directory to synthesize the test logic for the given *design_name* and *design_id*.

Figure 7-11 shows the subdirectories and files created with run_synthesis. Descriptions of their content and usage are also described. One synthesis manager is created per user/host combination. It remains alive even if Tessent Shell is terminated or suspended. The manager terminates itself when it has finished dispatching all of the simulations and Tessent Shell is not attached to it through a socket.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 7-11. Typical Contents of the synthesis_outdir Directory

```
synthesis_outdir
|-- design_name_design_id.synthesis_dc_shell
|-- design_name_design_id_cells.instrument
| |-- design_name_design_id_tessent_*.vg
|-- design_name_design_id_ijtag.instrument
| |-- design_name_design_id_tessent_*.vg
|-- design_name_design_id_mbisr.instrument
| |-- design_name_design_id_tessent_mbisr_register_*.vg
|-- design_name_design_id_mbist.instrument
| |-- design_name_design_id_tessent_mbist_bap.vg
| |-- design_name_design_id_tessent_mbist_*_controller.vg
| |-- design_name_design_id_tessent_mbist_*_retiming_*.vg
|-- command.log
|-- synthesis_tool.synthesis_log
|-- synthesis_tool.synthesis_tcl
|-- file_copy.script
|-- file_list
|-- synthesis.script
```

- *design_name_design_id.synthesis_dc_shell* — A directory that holds the synthesis data associated with a given *design_name* and *design_id*.
- *design_name_design_id_cells.instrument* — A directory that holds the synthesized rtl cells associated with a given *design_name* and *design_id*.
- *design_name_design_id_ijtag.instrument* — A directory that holds the synthesized IJTAG RTL instruments associated with a given *design_name* and *design_id*.
- *design_name_design_id_mbisr.instrument* — A directory that holds the synthesized BISR RTL instruments associated with a given *design_name* and *design_id*.
- *design_name_design_id_mbist.instrument* — A directory that holds the synthesized memory BIST RTL instruments associated with a given *design_name* and *design_id*.
- *command.log* — A file that contains a transcript of the commands executed by the synthesis tool.
- *synthesis_tool.synthesis_log* — A file that contains the complete transcript that is generated when running the synthesis.script. The synthesis manager simply redirects the output of the script into this file. The check_synthesis command parses this file and reports the status.
- *synthesis_tool.synthesis_tcl* — A file that contains the Tcl script that this used to perform the synthesis by the *synthesis_tool* invoked by the synthesis.script.
- *file_copy.script* — A file that contains the script to manually copy the synthesized modules back to the TSDB.
- *file_list* — A file that contains the list of files used by the *file_copy.script*.
- *synthesis.script* — A file that contains the commands to invoke *synthesis_tool* with the *synthesis_tool.synthesis_tcl* that synthesizes the test logic RTL.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Environment Variables

Synthesis Script Variables

For the run_synthesis command, a list of environment variables is exported to a side shell script that is used by the synthesis scripts when they are executed. This way, the execution should be consistent even when executed standalone, without the simulation/synthesis managers.

This is particularly important because the environment variables can be changed in Tessent Shell. When changed this way, the modified environment variables were automatically inherited when sub-invoking a shell command with the 'exec' or 'system' command. However, since those environment variable were not exported to the simulation/synthesis scripts, the behavior could be different than when sub-invoking the scripts directly with 'exec' or 'system' from Tessent Shell if some key environment variables were modified inside Tessent Shell.

By default, the following list of environment variables are exported:

- VCS_ARCH_OVERRIDE
- PATH
- LD_LIBRARY_PATH
- *_HOME
- *_VERSION
- *_LICENSE_FILE

In addition to the variables in that list, more environment variables can be exported by specifying a list of names ('*' accepted) in an environment variable.

For synthesis, environment variable 'TESSENT_SYNTH_ENVAR_LIST' is used.

For example, in Tessent Shell, using:

```
setenv TESSENT_SYNTH_ENVVAR_LIST [list MY_VARS_* ANOTHER_VAR]
```

picks-up and exports any environment variable with names that start with 'MY_VARS_' and the variable name 'ANOTHER_VAR'. When a variable name in those lists is optionally prefixed by a '-' sign, it won't be exported. The optional '+' sign is accepted and also, it the default behavior.

A side file is generated that includes the export of all the selected environment variables and that file is sources by the main synthesis scripts. The file is named *setup.script* and exported to the following location:

```
synthesis_outdir/design_name_design_id.synthesis_tool
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

VCS_ARCH_OVERRIDE Variable

The VCS_ARCH_OVERRIDE environment variable is automatically exported in the generated *synthesis.script* with the 'linux' value only when the variable is not set in the setup script and the operating system is detected as 'linux'.

In order to change this default behavior, issue the following command prior to the calling the run_synthesis command:

```
setenv VCS_ARCH_OVERRIDE ""
```

Synthesis Logfile Variable

When the environment variable TESSENT_SYNTH_MANAGER_ENABLE_LOG is set to the value 1, a log file is produced that contains the information exchanged between Tessent Shell and the synthesis manager. By default, the name of the file is './tessent_synth_manager.log' and the path and name of this file can be specified by the environment variable TESSENT_SYNTH_MANAGER_LOG_FILE_NAME.

By default, the file is overwritten for every Tessent Shell session. It is possible to append to an existing file by changing the filename and specifying the letter 'a' after the filename. The optional letter 'w' reproduces the default behavior.

For example, the following enables the log append to the specified file, if it already exists:

```
setenv TESSENT_SYNTH_MANAGER_ENABLE_LOG 1
setenv TESSENT_SYNTH_MANAGER_LOG_FILE_NAME './my_tessent_synth_manager.log a'
```

Synthesis Instance Manager

The TESSENT_SYNTH_MANAGER_INSTANCES environment variable can be used to control the number of managers. By default, when this environment variable is not set, the tool creates one manager per user per host. This is the same as setting the value to 1.When you use a value greater than 1, one manager per user per host per Tessent Shell session is started with the manager name being the following:

```
tessent_synth_manager_pid
```

Using this name, it is possible to find the manager in the PID list reported by the OS using the command:

```
ps -F -C tclsh8.5.exe | cat
```

Synthesis Socket Manager

The TESSENT_SYNTH_MANAGER_SOCKET_SELECTION environment variable is used to manage sockets. The default value is “server” which causes the manager to use any socket channel returned by the operating system.

The value “auto” means that the tool automatically selects a socket as returned by the operating system and requests that the started manager use it.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Specifying an integer value forces the started manager to use the specified socket, if available.

Synthesis Timeout Manager

The TESSENT_SYNTH_MANAGER_TIMEOUT_SECS environment variable controls how long the tool waits for the socket manager. By default, the run_synthesis command waits for the manager to start and open a socket for 300 sec (5 minutes). You can use this environment variable to change this value.

Synthesis Debug

As an alternative to the enable log environment variable, you can use the TESSENT_DEBUG_CONTROL environment variable to generate a logfile. You set the TESSENT_DEBUG_CONTROL environment variable to the following value:

```
+tessent_synth_manager/log_file
```

or use the following in Tessent Shell:

```
set_debug_control +tessent_synth_manager/log_file
```

It is possible to generate debug information while the managers are starting/running with the following debug controls:

```
+tessent_synth_manager/show_debug_info
```

This information can be of value to your Siemens representative in the case where a manager refuses to start or does not perform as expected.

Arguments

- -design_name *design_name*

  An optional switch and value pair that specifies the design name to which run_synthesis associates with. By default, the name of the current design is used when it is set. If you run the command immediately after running the process_dft_specification command, you do not need to specify this option.

- -design_id *design_id*

  An optional switch and value pair that specifies the design id to which run_synthesis is associated with. By default, the name of the current design id is used when it is set. If you run the command immediately after running the process_dft_specification command, you do not need to specify this option.

- -generate_scripts_only

  An optional switch that fully populates the synthesis output directory but does not run the synthesis tool. You can use this option to create the synthesis scripts, and then invoke synthesis using the generated scripts stand alone rather than using the synthesis manager.

- -synthesis_tool dc_shell | genus

  An optional switch and value pair that specifies the synthesis tool you wish to use for synthesis. The default is “dc_shell”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -compilation_options *option_list*

  An optional switch and option list that specifies additional options to be added to the synthesis compile command used in the *<synthesis_tool>.synthesis_tcl*. An example would be the -boundary_optimization option for dc_shell.

- -pre_compilation_commands *command_list*

  An optional switch and Tcl command list that adds the provided commands to *<synthesis_tool>.synthesis_tcl* so that they are executed before the compilation command is executed by the synthesis tool. Each element of the list becomes a command line in the *<synthesis_tool>.synthesis_tcl* script.

- -post_compilation_commands *command_list*

  An optional switch and Tcl command list that adds the provided commands to *<synthesis_tool>.synthesis_tcl* so that they are executed after the compilation command is executed by the synthesis tool. Each element of the list becomes a command line in the *<synthesis_tool>.synthesis_tcl* script.

- -synthesis_output_directory *directory*

  An optional switch and value pair that specifies the synthesis output directory. The default directory is *synthesis_outdir* and is located in the Tessent Shell invocation directory.

- -startup_file *filename*

  An optional switch and file path that explicitly specifies the startup file that is used by the synthesis tool. The file specified is copied into the actual synthesis directory.

  Note

  If a startup file is not specified and a *.synopsys_dc.setup* file exists in the Tessent run directory, it is automatically copied into the synthesis run directory. If relative paths are utilized within this setup file, it should be noted the synthesis run directory is two levels below where Tessent Shell is launched and these paths should account for this execution point.

- -no_wait

  An optional switch that starts the synthesis but does not wait for the command to complete. To inquire about the status of a synthesis task started in this manner, use the check_synthesis command.

  When the -no_wait switch is not specified, the run_synthesis command waits for the synthesis task to finish before returning with a completion message. When running in interactive mode, the command reports a status line that updates every second showing at

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

what stage synthesis is currently operating. This status can be can be unscheduled, queued, running, pass, or fail, as shown below:

```
...
// Synthesis script creation for 'blockB_rtl' completed.
//
// Submitting
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/synthesis.script'
to synthesis manager
// Waiting for the synthesis to complete
unscheduled 0 queued 0 running 0 pass 1 fail 0
// The 17 synthesized files listed in
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/file_list' were
copied to the current tsdb output directory.
```

Examples

The following are examples of run_synthesis using various options.

Example 1

This example runs synthesis without any command options, after completing process_dft_specification:

SETUP> run_synthesis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Synthesis script creation for 'blockB_rtl'
// Begin processing of instrument synthesis dictionaries
// Processing containers for instrument type 'rtl_cells'
// Synthesis dictionary
./tsdb_outdir/instruments/blockB_rtl_cells.instrument/
blockB_rtl_cells.synthesis_dictionary
// Processing containers for instrument type 'ijtag'
// Synthesis dictionary
./tsdb_outdir/instruments/blockB_rtl_ijtag.instrument/
blockB_rtl_ijtag.synthesis_dictionary
// Processing containers for instrument type 'mbisr'
// Synthesis dictionary
./tsdb_outdir/instruments/blockB_rtl_mbisr.instrument/
blockB_rtl_mbisr.synthesis_dictionary
// Processing containers for instrument type 'mbist'
// Synthesis dictionary
./tsdb_outdir/instruments/blockB_rtl_mbist.instrument/
blockB_rtl_mbist.synthesis_dictionary
// Done processing of instrument synthesis dictionaries
//
// Writing Tcl synthesis script
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/dc_shell.synthesis_tcl'
// Writing the synthesis shell script
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/synthesis.script'
// Writing the file copy shell script
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/file_copy.script'
// Writing the synthesized file list
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/file_list'
// Synthesis script creation for 'blockB_rtl' completed.
//
// Submitting
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/synthesis.script' to
synthesis manager
```

Example 2

This example runs synthesis with -generate_scripts_only options so that you may run synthesis outside Tessent Shell:

SETUP> run_synthesis -generate_scripts_only

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Synthesis script creation for 'blockB_rtl'
// Begin processing of instrument synthesis dictionaries
// Processing containers for instrument type 'rtl_cells'
// Synthesis dictionary
./tsdb_outdir/instruments/blockB_rtl_cells.instrument/
blockB_rtl_cells.synthesis_dictionary
// Processing containers for instrument type 'ijtag'
// Synthesis dictionary
./tsdb_outdir/instruments/blockB_rtl_ijtag.instrument/
blockB_rtl_ijtag.synthesis_dictionary
// Processing containers for instrument type 'mbisr'
// Synthesis dictionary
./tsdb_outdir/instruments/blockB_rtl_mbisr.instrument/
blockB_rtl_mbisr.synthesis_dictionary
// Processing containers for instrument type 'mbist'
// Synthesis dictionary
./tsdb_outdir/instruments/blockB_rtl_mbist.instrument/
blockB_rtl_mbist.synthesis_dictionary
// Done processing of instrument synthesis dictionaries
//
// Writing Tcl synthesis script
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/dc_shell.synthesis_tcl'
// Writing the synthesis shell script
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/synthesis.script'
// Writing the file copy shell script
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/file_copy.script'
// Writing the synthesized file list
'./synthesis_outdir/blockB_rtl.synthesis_dc_shell/file_list'
// Synthesis script creation for 'blockB_rtl' completed.
```

Related Topics

check_synthesis

get_run_synthesis_options

report_run_synthesis_options

set_run_synthesis_options

write_design_import_script

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

run_testbench_simulations

Context: all contexts

Mode: all modes

Runs a simulation manager to run a set of simulation testbenches.

Usage

run_testbench_simulations
[-design_name *design_name*]
[-design_id *design_id*]
[-pattern_id *pattern_id*]
[-report_list]
[-select *pattern_name_glob_list*]
[-exclude *pattern_name_glob_list*]
[-generate_scripts_only | -run_only]
[-parallel_simulations {1 | 2...*MAX_INT* | maxcpu}]
[-expected_miscompare_count *int*]
[-simulation_id *string*]
[-simulator questa | vcs | incisive]
[-simulation_timeout time | unlimited]
[-keep_simulation_data on | off | on_failure]
[-store_simulation_waveforms on | off]
[-simulation_run_commands *simulator_commands*]
[-waveform_configuration_commands *simulator_command_list*]
[-compilation_macro_definitions {*macro*[=*value*] ... }]
[-simulation_macro_definitions {*macro*[=*value*] ... }]
[-compilation_options *option_list*]
[-simulator_options *option_list*]
[-extra_verilog_files *list_of_files*]
[-extra_top_modules *module_name_list*]
[-simulation_output_directory *simulation_output_directory*]
[-simdut_output_directory *simdut_output_directory*]
[-simdut_server] [-generate_simdut_files_only]
[-use_design_view_per_simulation on | off]
[-no_wait]

Description

The run_testbench_simulations command is typically run with no arguments. It automatically uses the design name, design id, and pattern id found in the previously processed PatternsSpecification(*design_name*, *design_id*, *pattern_id*) wrapper.

The check_testbench_simulations command monitors the simulation results.

The information on how to load the design view into the simulator is contained in the <*design_name*>.*design_source_dictionary* files found inside the dft_inserted_designs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

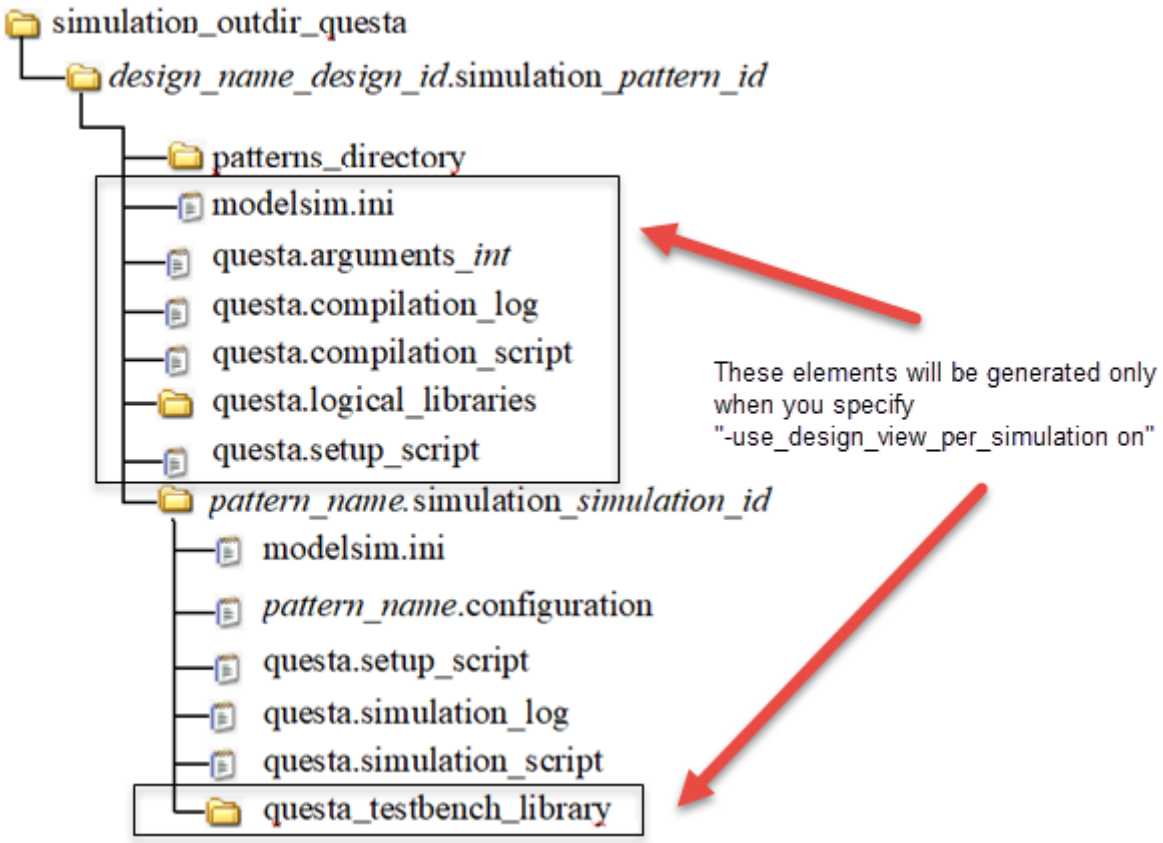
directories. The only missing information is where to find the simulation library models because those are not loaded inside Tessent Shell. You specify the location of the simulation library models using the set_simulation_library_sources command.

Refer to the "check_testbench_simulations -report_status" switch for details on obtaining status for a given simulation run.

The command dispatches and connects to a simulation manager using a socket. If a manager is already running, it simply connects to it. The simulation manager is an unlicensed Tcl shell executable that uses the content found in the simulation.data_dictionary file inside the *design_name_design_id.patterns_pattern_id* directory to compile the design and the testbenches into the specified -simulation_output_directory, and to simulate them using the specified simulator. The simulations are run in series or in parallel based on the specified -parallel_simulations value. Figure 7-12 shows the subdirectories and files created below the specified "-simulation_output_directory *simulation_outdir_questa*" command.

One simulation manager is created per user/host combination. It remains alive even if Tessent Shell is terminated. The manager terminates itself when it has finished dispatching all of the simulations and Tessent Shell is not attached to it through a socket.

Figure 7-12. Example Output Directory Structure for Questa



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The -simulation_output_directory switch is set to *simulation_outdir_questa* for this example.

- *design_name_design_id.simulation_pattern_id*

  A directory that holds the simulation data associated with a given *design_name*, *design_id*, and *pattern_id* combination.

- *patterns_directory*

  A soft link that points back to the directory in which the Verilog testbench files exists. The *simulator.simulation_script* file uses that link to access the testbench file.

- *simulator.arguments_int*

  A file created for all each unique set of "-y ... -v ... +libext+..." in the compilation/simulation shell scripts generated. This file is added to the Verilog compilation command as "-f *simulator*.arguments_*int*".

- *simulator.compilation_script*

  A file that contains simulator-specific commands to map logical libraries to physical directories and compile the complete design into the various logical libraries. An extra logical library called "tsdb_interface" is created and the *design_name.format_interface* files of the first level of physical block modules are compiled into it. The *pattern_name.configuration files* use this library to bind the interface view for different instances of physical blocks based on what was specified in the LowerPhysicalBlockInstances wrapper inside the SimulationOptions wrapper.

- *simulator.logical_libraries*

  A directory that contains one subdirectory per logical library, and in which the compilation data is stored. This directory is retained or deleted at the end of a simulation based on the value of the -keep_simulation_data {on | off | on_failure} option.

- *simulator.compilation_log*

  A file that contains the complete transcript generated when running the *simulator.compilation_script*. The simulation manager simply redirects the output of the script into this file.

- *pattern_name.simulation[_simulation_id]*

  A directory that is created to simulate each pattern. The *_simulation_id* extension suffix is only present when the "run_testbench_simulations -simulation_id *string*" option is used.

- *simulator_testbench_library*

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This directory stores several files and directories associated with the simulation of a pattern, including:

- Configuration file
- Simulation script
- Log file
- Simulation waveforms
- Directory containing the compilation data for the testbench and the configuration file

The use of a unique directory per pattern enables you to retain or delete it at the end of a simulation based on the value of the -keep_simulation_data {on | off | on_failure} option. The -store_simulation_waveforms on | off option also determines if the directory is retained or deleted at the beginning of a simulation.

- *simulator.simulation_script*

  A file that contains simulator-specific commands to map logical libraries to physical directories, compile the testbench as well as the configuration file, and run the simulation.

- *patterns.configuration*

  A file that contains a Verilog configuration that is used to control whether a *physical_block* instance uses the full view or the interface view when simulating the given patterns testbench.

- *simulator.simulation_log*

  A file that contains the complete transcript that is generated when running the *simulator.simulation_script*. The simulation manager simply redirects the output of the script into this file. The check_testbench_simulations command parses those files to look for miscompares.

- *simulator.simulation_log_reference*

  A file that contains a copy of a previously-generated *simulator.simulation_log* file. This file is manually created by copying the *simulator.simulation_log file* when you are doing simulations with fault insertion.

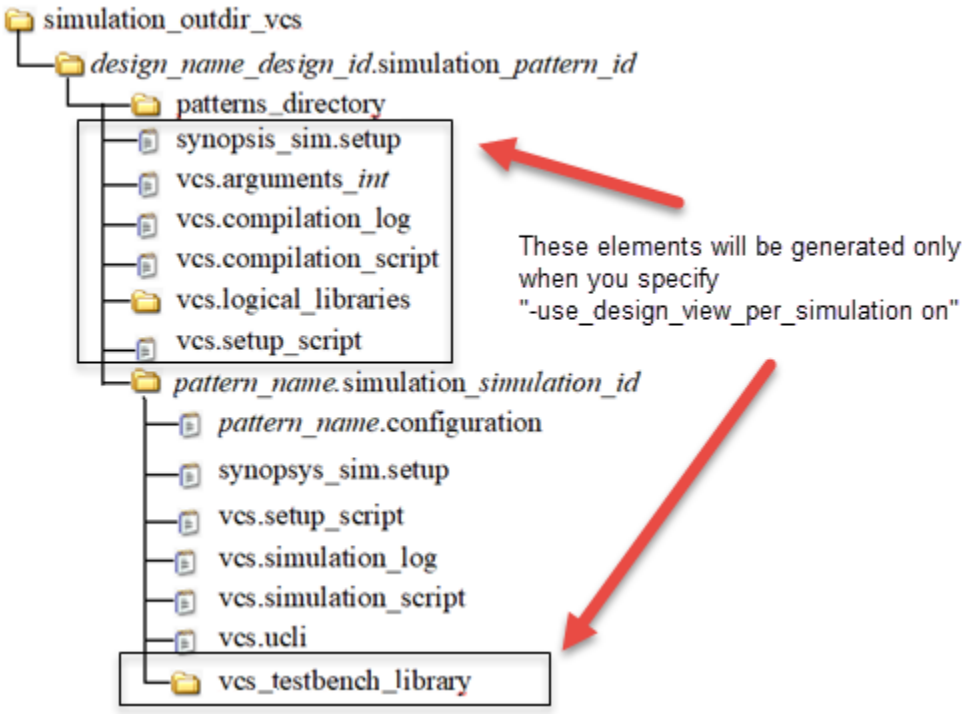
  This file instructs the check_testbench_simulations command to expect those exact failures in the newly generated *simulator.simulation_log* file. You can also use the -expected_miscompare_count option but this method does not check on which object and in which test step the miscompares actually happened. For example, if you are doing fault insertion to test the memory repair solution, you want to make sure the miscompares are in the memory BIST step prior to repair and not in the one after repair.

  The use of the *simulator.simulation_log_reference* file enables you to precisely specify the location of the miscompares. Those files are typically stored in

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

pattern_name.simulation[_simulation_id] directories where the *simulation_id* is not null so as not to lose the fault-free simulation directories.

Figure 7-13. Example Output Directory Structure for VCS

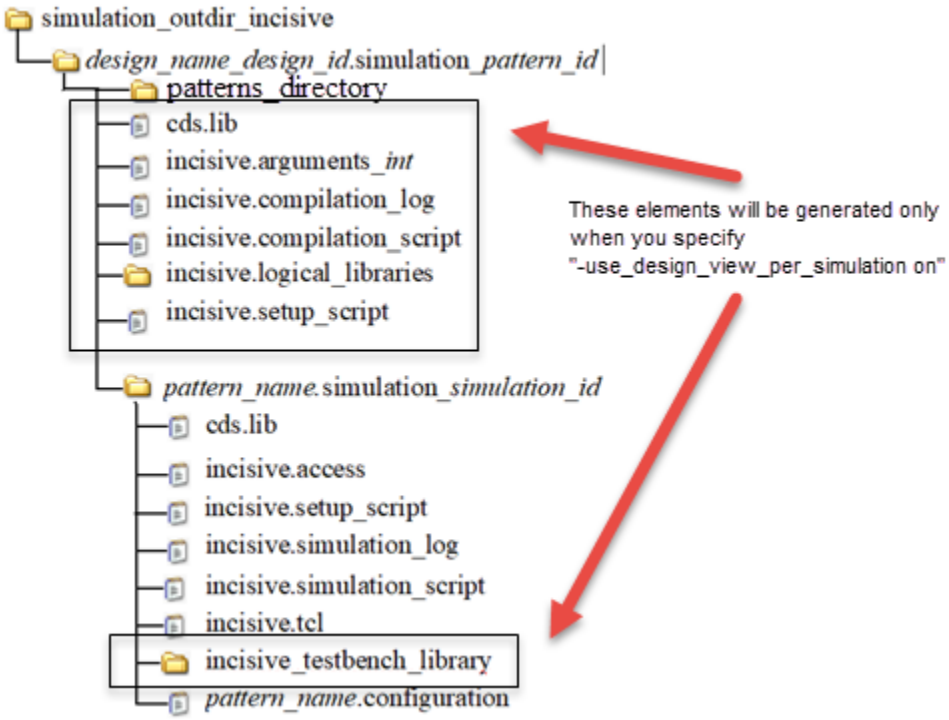


Note

The -simulation_output_directory switch is set to *simulation_outdir_vcs* and the -simulator switch is set to “vcs” for this example.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 7-14. Example Output Directory Structure for Incisive



Note

The -simulation_output_directory switch is set to *simulation_outdir_incisive* and the -simulator switch is set to “incisive” for this example.

Environment Variables

Simulation Script Variables

For the run_testbench_simulations command, a list of environment variables is exported to a side shell script that is used by the synthesis scripts when they are executed. This way, the execution should be consistent even when executed standalone, without the simulation/synthesis managers.

This is particularly important because the environment variables can be changed in Tessent Shell. When changed this way, the modified environment variables were automatically inherited when sub-invoking a shell command with the 'exec' or 'system' command. However, since those environment variable were not exported to the simulation/synthesis scripts, the behavior could be different than when sub-invoking the scripts directly with 'exec' or 'system' from Tessent Shell if some key environment variables were modified inside Tessent Shell.

By default, the following list of environment variables are exported:

- PATH
- LD_LIBRARY_PATH

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- *_HOME
- *_VERSION
- *_LICENSE_FILE
- MTI* (set to “64” when the MTI_VCO_MODE variable is *not* set)

In addition to the variables in that list, more environment variables can be exported by specifying a list of names ('*' accepted) in an environment variable.

For simulations, the 'TESSENT_SIM_ENVVAR_LIST' environment variable is used.

For example, in Tessent Shell, using:

```
setenv TESSENT_SIM_ENVVAR_LIST [list MY_VARS_* ANOTHER_VAR]
```

picks-up and exports any environment variable with names that start with 'MY_VARS_' and the variable name 'ANOTHER_VAR'. When a variable name in those lists is optionally prefixed by a '-' sign, it won't be exported. The optional '+' sign is accepted and also, it the default behavior.

A side file is generated that includes the export of all the selected environment variables and that file is sources by the main simulation scripts. The file is named *simulator.setup_script* and is exported to the following locations:

```
simulation_outdir/design_name_design_id.simulation_simulation_id

simulation_outdir/design_name_design_id.simulation_simulation_id/
pattern_name.simulation[_pattern_id]
```

Simulator HOME Environment Variables

When setting the following simulator HOME environment variables to the simulator software’s *install_home*:

- QUESTA_HOME
- VCS_HOME
- INCISIVE_HOME

the run_testbench_simulations command automatically looks in the following locations:

- *install_path*/bin
- *install_path*/tools/bin

to find the required commands. When found, the generated *simulator*.setup_script has the PATH environment variable automatically appended with the path to the *bin* directory. Setting this environment variable is not required if the PATH environment variable in Tessent Shell already includes the path to the *bin* directory of the required commands.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The TESSENT_HOME Environment Variable

The TESSENT_HOME environment variable is automatically set inside Tessent Shell, and it refers to the full path to the Tessent installation directory.

This variable is automatically exported to the <*simulator*>.*setup_script* generated by the run_testbench_simulations command as follows:

```
if [ "$TESSENT_HOME" = "" ] ; then export \
TESSENT_HOME='<full_path_to_tessent_installation_directory>'; fi
```

When needed, the path to the quick synthesis library *dft_sim.v*, which resides in a subdirectory of the Tessent installation directory, is also reported in the simulation scripts as a command line argument to simulator commands. For example, for the Questa 'vlog' command, the path is passed as follows:

```
vlog ... -v $TESSENT_HOME/lnx-x86/lib64/hdleng/lib/dft_sim.v ...
```

Given the way the TESSENT_HOME environment variable is conditionally set in the <*simulator*>.*setup_script*, it is possible to override the value of TESSENT_HOME when the <*simulator*>.*compilation_script* and <*simulator*>.*simulation_script* are called directly from the operating system. Here is an example for Questa:

In Tessent Shell:

```
run_testbench_simulations -generate_scripts_only ...
```

From the operating system shell:

```
setenv TESSENT_HOME /path/to/alternate/tessent/installation/directory
cd simulation_outdir/chip_rtl.simulation_signoff
./questa.compilation_script
cd ICLNetwork.simulation
./questa.simulation_script
```

Simulation Logfile Variable

When the environment variable TESSENT_SIM_MANAGER_ENABLE_LOG is set to the value 1, a log file is produced that contains the information exchanged between Tessent Shell and the simulation manager. By default, the name of the file is './tessent_sim_manager.log' and the path and name of this file can be specified by the environment variable TESSENT_SIM_MANAGER_LOG_FILE_NAME.

By default, the file is overwritten for every Tessent Shell session. It is possible to append to an existing file by changing the filename and specifying the letter 'a' after the filename. The optional letter 'w' reproduces the default behavior.

For example, the following enables the log append to the specified file, if it already exists:

```
setenv TESSENT_SIM_MANAGER_ENABLE_LOG 1
setenv TESSENT_SIM_MANAGER_LOG_FILE_NAME './my_tessent_sim_manager.log a'
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Simulation Instance Manager

The TESSENT_SIM_MANAGER_INSTANCES environment variable can be used to control the number of managers. By default, when this environment variable is not set, the tool creates one manager per user per host. This is the same as setting the value to 1.

When you use a value greater than 1, one manager per user per host per Tessent Shell session is started with the manager name being the following:

```
tessent_sim_manager_pid
```

Using this name, it is possible to find the manager in the PID list reported by the OS using the command:

```
ps -F -C tclsh8.5.exe | cat
```

Simulation Socket Manager

The TESSENT_SIM_MANAGER_SOCKET_SELECTION environment variable is used to manage sockets. The default value is “server” which causes the manager to use any socket channel returned by operating system.

The value “auto” means that the tool automatically selects a socket as returned by the operating system and requests that the started manager use it.

Specifying an integer value forces the started manager to use the specified socket, if available.

Simulation Timeout Manager

The TESSENT_SIM_MANAGER_TIMEOUT_SECS environment variable controls how long the tool waits for the socket manager. By default, the command waits for the manager to start and open a socket for 300 sec (5 minutes). You can use this environment variable to change this value.

Simulation Debug

As an alternative to the enable log environment variable, you can use the TESSENT_DEBUG_CONTROL environment variable to generate a logfile.

You set the TESSENT_DEBUG_CONTROL environment variable to the following value:

```
+tessent_sim_manager/log_file
```

or use the following in Tessent Shell:

```
set_debug_control +tessent_sim_manager/log_file
```

It is possible to generate debug information while the managers are starting/running with the following debug controls:

```
+tessent_sim_manager/show_debug_info
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This information can be of value to your Siemens representative in the case where a manager refuses to start or does not perform as expected.

Clock Monitoring During Simulation

By default clock monitoring is enabled during simulation.

Note

The clock monitoring relies on the design hierarchy being present. If the design has been ungrouped then it may be necessary to turn off the monitoring using the monitor_internal_clock_pins property in the SimulationOptions wrapper or use one of the macros shown in the example below when running the simulation.

This monitoring is done by a Verilog side file generated in the same directory as the simulation testbenches as follows:

```
tsdb_outdir/patterns/design_name_design_id.patterns_pattern_id/
patterns_name_CLOCK_MONITOR.v
```

The same file can be found with the *patterns_directory* softlink created by the run_testbench_simulation command under the following:

```
<simulation_outdir>/<design_name>_<design_id>.patterns_<pattern_id>/patterns_directory/
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For example, for a Patterns(MemoryBist_P1) entry in the PatternsSpecification, the clock monitoring side-file contains the following:

```
//
==========================================================================
//
//  File        :  MemoryBist_P1_CLOCK_MONITOR.v
//  Description :  Clock Monitor module for pattern MemoryBist_P1
//
//
==========================================================================
//
// By default the clock monitoring is enabled.
// To disable clock monitoring define one of the following macros:
//   TESSENT_DISABLE_CLOCK_MONITOR
//   TESSENT_DISABLE_CLOCK_MONITOR_MemoryBist_P1
//
`define TESSENT_ENABLE_CLOCK_MONITOR
`ifdef TESSENT_DISABLE_CLOCK_MONITOR
    `undef TESSENT_ENABLE_CLOCK_MONITOR
`endif
`ifdef TESSENT_DISABLE_CLOCK_MONITOR_MemoryBist_P1
    `undef TESSENT_ENABLE_CLOCK_MONITOR
`endif
`ifdef TESSENT_ENABLE_CLOCK_MONITOR
  `timescale 100fs/10fs
`endif

To disable all clock monitoring, use:
  -simulation_macro_definitions TESSENT_DISABLE_CLOCK_MONITOR

To disable clock monitoring only for the Patterns(MemoryBist_P1), use:
  -simulation_macro_definitions
TESSENT_DISABLE_CLOCK_MONITOR_MemoryBist_P1
```

Arguments

- -design_name *design_name*

  An optional switch and value pair that specifies the design name to which the patterns to simulate are associated with. By default, the name of the *current_design* is used when it is set. If you run the command immediately after running the process_patterns_specification command, you do not need to specify this option.

- -design_id *design_id*

  An optional switch and value pair that specifies the *design_id* to which the patterns to simulate are associated to. By default, the *patttern_id* used by the previously run process_patterns_specification command is reused automatically; otherwise, the *design_id* value set using the set_context command is used. If you run the command immediately after running the process_patterns_specification command, you do not need to specify this option.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -pattern_id *pattern_id*

  An optional switch and value pair that specifies the pattern id to which the patterns to simulate are associated to. By default, the *pattern_id* used by the previously run process_patterns_specification command is reused automatically. If you run the command immediately after running the process_patterns_specification command, you do not need to specify this option.

- -report_list

  An optional switch that tells the command to simply echo the list of patterns available in the *design_name_design_id.patterns_pattern_id/simulation.data_dictionary* file after it has been filtered using the -select or -exclude switch values.

- -select *pattern_name_glob_list*

  An optional switch and value pair that specifies a list of pattern names or pattern name glob patterns to include in the simulation run. The full list of pattern names can be reported using the -report_list option, and is stored in the simulation.data_dictionary file found inside the *design_name_design_id.patterns_pattern_id* directory.

- -exclude *pattern_name_glob_list*

  An optional switch and value pair that specifies a list of pattern names or pattern name glob patterns to exclude from the simulation run. The full list of pattern names can be reported using the -report_list option that is stored in the simulation.data_dictionary file found inside the *design_name_design_id.patterns_pattern_id* directory.

- -generate_scripts_only | -run_only

  An optional switch that tells the run_testbench_simulations command to create the required simulation scripts but not run them, or to run with the scripts already located in the simulation subdirectory.

  -generate_scripts_only — This option creates or replaces the subdirectory contents with the files needed to run the simulation, as described in Figure 7-12, but does not actually start the simulation. You can use this option to invoke the simulation script yourself or, later, start the simulation with the "-run_only" option.

  -run_only — This option uses the scripts already located in the simulation subdirectory, and runs them. The content of the *simulation* subdirectory is checked to make sure all the necessary files are still present, but nothing is deleted before starting the simulation. Use this option to avoid unnecessary processing, including recompilation of the design, when only modifying and simulating a pattern on a design that has not changed. This option is ignored if you also use the "-use_design_view_per_simulation on" option.

- -parallel_simulations {1 | 2...*MAX_INT* | maxcpu}

  An optional switch and value pair that specifies to the manager the number of simulations that are permitted to run in parallel. By default, only one simulation is launched at a time. *MAX_INT* is the maximum integer value that the host machine permits. The maxcpu literal specifies using all available CPUs on the host machine. You can increase the parallelism

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

based on the number of simulator licenses you have and the number of CPUs and memory available on your machine.

- -expected_miscompare_count *int*

  An optional switch and value pair that specifies that the set of patterns selected by the -select option have an expected mismatch count. This option is typically used in combination with the -simulation_id and the -simulation_macro_definition switches when doing fault insertion simulations. The fault insertion logic is triggered by the specified "define" macros. They can also be done with explicit forces inside a side module which you load and compile using the -extra_verilog_files option. Using the -simulation_id option enables you to keep a fault free simulation directory while generating new ones for the fault insertion variations.

  A more precise mechanism to document expected failures is to copy the *simulator*.simulation_log file to *simulator*.simulation_log_reference once you have inspected and verified it contains the right mismatches. You can also create a directory beside the specified *simulation_outdir* having the same name with an extra *.reference* extension.

  You copy the *simulator*.simulation_log file in the same subdirectory relative to *simulation_outdir*.reference as it exists in the *simulation_outdir* directory. If the *simulator*.simulation_log_reference is not found beside the log file but *simulator*.simulation_log exists inside *simulation_outdir*.reference/*same_sub_path*, then this file is used as the reference. The check_testbench_simulations command not only verifies that the right number of mismatches exists, but that they actually exist on the right objects and in the right test step. For future reruns of the flow, simply pre-populate the *simulator*.simulation_log_reference files inside the directories in which you are doing fault insertion and the check_testbench_simulations command automatically uses them.

- -simulation_id *string*

  An optional switch and value pair that specifies a simulation id string to name the *pattern_name*.simulation[*_simulation_id*] directory. This is typically used in conjunction with the -simulation_macro_definitions or the -extra_verilog_files options to trigger failures in the simulations. For more information, see the -expected_miscompare_count switch description.

- -simulator questa | vcs | incisive

  An optional switch and value pair that specifies the simulator you want to use for simulation. The default value is "questa"; the Questa Advanced Simulator from Siemens EDA.

- -simulation_timeout {time | unlimited}

  An optional switch and value pair that specifies the time out value you want to use when waiting for a simulation to complete. The default value is "unlimited" which means the simulation is permitted to run for as long as it takes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -keep_simulation_data {on | off | on_failure}

  An optional switch and value pair that specifies the conditions under which to keep the simulation data after the simulation has been completed. The deletion of the *simulator_testbench_library* directory, including the *simulator.logical_libraries* subdirectory, and the debug output generated when using the -store_simulation_waveforms option is affected by this option.

  on — All the files and subdirectories generated by the simulation process are kept. This is the default.

  off — All the files and subdirectories generated by the simulation process are removed, except the *simulator.simulation_log* file, irrespective of whether the simulation passes or fails. Use this option to preserve disk space.

  on_failure — All the files and subdirectories generated by the simulation process are kept but only when the simulation fails. When the simulation passes, all those files are removed except the *simulator.simulation_log* file. This also saves disk space.

- -store_simulation_waveforms on | off

  An optional switch and value pair that specifies to store simulation waveform data so that they can be analyzed with the simulator tools. This is especially useful to find the root cause of a simulation failure.

  on — Waveform data is generated and retained at the end of a simulation if the -keep_simulation_data option is also set to “on” (default) or set to “on_failure” and the simulation fails.

  off — Waveform data is not generated. This is the default.

  Note

  The “-store_simulation_waveforms” and “-keep_simulation_data” options influence the behavior of the “run_testbench_simulation -run_only” command. When both options are “on,” all the files in the *simulation* subdirectory are kept prior to starting the simulation, but all subdirectories below it are removed. This means that the simulation waveform files generated by the simulator are not erased, enabling them to be updated by the simulator. This enables the simulation viewer tool to only refresh the waveforms with the updated data once the simulation completes.

- -simulation_run_commands *simulator_commands*

  An optional switch and value pair that specifies the *simulator_commands* to use when the -store_simulation_waveform switch is off. If the -store_simulation_waveforms is set to on, the content from the -waveform_configuration_commands switch is prepended to the -simulation_run_commands switch.

  The *simulator_commands* value should be consistent with your choice of simulator. The default varies by simulator, but is an equivalent of the following Questa commands:

  ```
  "run -all; exit"
  ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -waveform_configuration_commands *simulator_command_list*

  An optional switch and value pair that specifies a list of simulator-specific commands to control the resolution of the stored waveform data. When you specify "-store_simulation_waveforms on" without the -waveform_configuration_commands option, all waveforms are stored which may dramatically slow down the simulation on a large design.

- -compilation_macro_definitions {*macro*[=*value*] ...}

  An optional switch and value pair that specifies a list of macro definitions to pass to the Verilog compilation command inside the *simulator*.compilation_script file which is where your design is compiled. This is typically used to configure your simulation library.

- -simulation_macro_definitions {*macro*[=*value*] ...}

  An optional switch and value pair that specifies a list of macro definitions to pass to the Verilog compilation command inside the *simulator*.simulation_script file which is where your testbench and side modules are compiled. You can use this option to trigger special fault insertion code inside your side modules to perform fault insertion simulation. See the -expected_miscompare_count switch description.

- -compilation_options *option_list*

  An optional switch and value pair that specifies a list of compilation options (for example, +initreg) that modify the compilation script during runtime. The *option_list* uses the following syntax:

  -compilation_options {[*global_options*] [verilog *verilog_options*}] [vhdl {*vhdl_options*}]}

  - The verilog options are passed only to the command that compiles Verilog source files.
  - The vhdl options are be passed only to the command that compiles the VHDL source files.
  - The *global_options* can also be expressed as "global {*global_options*}".

  The braces are mandatory in order to identify the options preceded by the language or "global".

  See Example 4.

- -simulator_options *option_list*

  An optional switch and value pair that specifies a list of options to pass to the simulation command. These are simulator-specific commands and must be consistently specified with the -simulator option.

  If you are using Questa SIM as your simulator, you can use this switch to pass in the -debugDB option to the tool so the *vsim.dbg* file is created and can be accessed using the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

schematic viewer when debugging your waveforms. You must specify this option in conjunction with the -store_simulation_waveform switch as follows:

```
run_testbench_simulation -store_simulation_waveforms on \
  -simulator_options "-debugDB"
```

Note

Some versions of Questa SIM may require -debugdb instead of -debugDB.

Example 6 shows how to create an *option_list* for the run_testbench_simulations or set_testbench_simulation_options command '-simulator_options' argument. This is useful when you need to pass escaped identifiers as options.

Example 7 shows how to create a file of options that the simulator can load indirectly instead of being passed directly to the simulator command line.

- -extra_verilog_files *list_of_files*

  An optional switch and value pair that specifies a list of individual Verilog files to compile in parallel to the testbench file. The *list_of_files* are *actual* Verilog files in contrast to a file containing references to Verilog source files. For example:

```
# A single Verilog file:
run_testbench_simulations -extra_verilog_files a.v

# Multiple Verilog files:
run_testbench_simulations -extra_verilog_files { a.v b.v c.v }
```

  The files are typically used to contain codes that perform monitoring or fault insertion in the design. You must list the module names compiled within those files that you want to be active in parallel with the testbench using the -extra_top_modules option.

Tip

When using the Two-Pin Serial Port for IJTAG, refer to the topic "How to Avoid Simulation Issues When Using the Two-Pin Serial Port Controller" in the *Tessent Shell User's Manual*.

- -extra_top_modules *module_name_list*

  An optional switch and value pair that specifies a list of modules to run in parallel with the testbench module. These modules are typically used to contain codes that perform monitoring or fault insertion in the design. You must load these modules in files specified with the -extra_verilog_files option.

- -simulation_output_directory *simulation_output_directory*

  An optional switch and value pair that specifies the name and path of a directory in which to place the simulation directories. See Figure 7-12 for more information about this directory. The default is *simulation_outdir*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -simdut_output_directory *simdut_output_directory*

  An optional switch and value pair that is specified with the -simdut_server option to reference the path to the directory that contains the SimDUT setup files that are required for simulation. Refer to "Simulating Diagnosis of ATPG Patterns (Non-TSDB Flow)" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

- -simdut_server

  An optional argument that specifies to run the testbenches in Tessent SiliconInsight SimDUT mode. Refer to "Simulating Desktop, ATE, and ATPG Behavior" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

- -generate_simdut_files_only

  An optional argument that is specified with the -simdut_server option to indicate that Tessent SiliconInsight should generate the SimDUT setup files. By default, this directory is named SIMDUT.simulation and is located in the simdut_outdir directory. Refer to "Generating the Setup Specification for ATPG (Non-TSDB Flow) Desktop Mode" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

- -use_design_view_per_simulation on | off

  An optional switch and value pair that forces the tool to perform the simulation with the full view of the design when this switch is on. The default value is off.

  When this switch is on, the highlighted files in 7-12, 7-13 and 7-14 are generated. The *simulator.compilation_script* includes all the instructions for the simulator to compile the full design.

  The run_testbench_simulation command automatically runs this script and when it succeeds, it proceeds to run every *simulator.simulation_script* in the sub-directories *pattern_name*.simulation_*simulation_id* and generates the files under each subdirectory.

  Use this switch to overcome simulator limitations. For example, when a mixed-language (Verilog+VHDL) design uses direct instantiation for some VHDL instances, this is the only way for the simulator to handle those simulations.

- -no_wait

  By default, the run_testbench_simulations command automatically waits for the simulations to complete and periodically updating a status line giving the progress of the ongoing simulations.

  The optional -no_wait switch instructs the command to dispatch the simulation to the manager and not wait for them to complete. It is then possible to follow the progress of the simulations by invoking the check_testbench_simulations command. In fact, calling the run_testbench_simulations command without the '-no_wait' option is equivalent to doing the following:

```
run_testbench_simulations    -no_wait ...
check_testbench_simulations -wait
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example runs the run_testbench_simulations command twice—once with no options and once to select a few patterns for which fault insertion is performed. Notice the use of the -simulation_id in the second invocation which enables the fault-free and fault-injected simulations to co-exist even when the same pattern name is reused. The example assumes that the *simulator.simulation_log* files were copied to *simulator.simulation_log_reference* in the *MemoryBist_*.simulation_FI* directories using the result of a previous run.

```
run_testbench_simulations
run_testbench_simulations -select MemoryBist_* \
    -simulation_macro_definitions {mem32x56_fi_add=4 mem32x56_fi_bit=2} \
    -simulation_id FI
check_testbench_simulations
```

Example 2

The following example shows how to use the "write_design -tsdb -softlink_netlist" command and switches to create a TSDB dft_inserted_designs container for a netlist generated from synthesis or layout, and on which you want to use the run_testbench_simulations command to simulate the patterns described in your PatternsSpecification.

As shown in Example 3 of the read_design command, you read in the post-synthesis or post-layout netlist using read_verilog, and then you read the ICL and TCD description using the read_design -no_hdl command. You then write out a TSDB dft_inserted_design container using the "write_design-tsdb -softlink_netlist" command and switches. If you are using this command to create a TSDB dft_inserted_design container for the post-layout view, you would specify -design_id with a name other than "gate," because "-design_id gate" was likely used for the post-synthesis view.

```
set_context dft -design_id gate -no_rtl
read_cell_library cell_library
read_verilog path/my_design.vg
read_design my_design -design_id rtl -no_hdl -verbose
set_current_design my_design
write_design -tsdb -softlink_netlist -verbose
```

Now that you have a dft_inserted_design container for your netlist, you can either create a new PatternsSpecification using create_patterns_specification or re-use a pre-existing PatternsSpecification that was potentially modified to signoff the pre-synthesis design view.

```
set_context dft -no_rtl -design_id gate
# This loads the ICL and the interface view of the verilog design to get the full port list of the
# design. No need to read the full netlist to run process_patterns_specification
read_design my_design -view interface
set_current_design my_design
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# Option 1: Recreate a brand new PatternsSpecification
create_patterns_specification

# Option 2: re-use pre-existing PatternsSpecification
read_config_data \
      ./tsdb_outdir/patterns/my_design_rtl.patterns_spec_signoff
set_config_value PatternsSpecification(my_design,rtl,signoff) -id <1>  gate
process_patterns_specification
run_testbench_simulations

process_patterns_specification
run_testbench_simulations
```

Example 3

The following example shows the tool command sequence to setup the libraries, run the simulation, and check the simulation:

```
set_simulation_library_sources -v adk.v -v ram.v
run_testbench_simulations
check_testbench_simulations
```

Example 4

The following example demonstrates using the -compilation_options switch:

```
run_testbench_simulations -generate_scripts_only -simulator vcs \
      -simulator_options {-kdb -lca -debug_all} \
      -compilation_options \
            { global { -kdb } vhdl { -skip_translate_body } verilog { -timescale=1ns/1ps } }
```

Example 5

The following example creates the simulation files for ATPG patterns using the "create_patterns_specification -testbench_files" command and then simulates the patterns using the run_testbench_simulations command:

```
set_context patterns -scan
read_cell_library ../library/standard_cells/tessent/adk.tcelllib
read_cell_library ../library/memories/picdram.tcelllib

# Point to the TSDB directory
set_tsdb_output_directory ../tsdb_outdir
read_design cpu_top -design_id gate -verbose
set_current_design cpu_top

# Create and process a pattern specification based on ATPG patterns saved with
#   the write_patterns command
set spec [create_patterns_specification -replace \
   -testbench_files {patterns/cpu_top_stuck_parallel.v patterns/cpu_top_stuck_serial.v  \
   patterns/cpu_top_transition_parallel.v patterns/cpu_top_transition_serial.v}]
process_pattern_specification
set_simulation_library_sources -v ../library/standard_cells/verilog/adk.v \
   -v ../library/memories/picdram.v
# Run the simulations
run_testbench_simulations
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 6

The following example builds a list of simulator options that include escaped identifiers. You must supply the arguments as a list that can be passed to the "run_testbench_simulations -simulator_options <*option_list*>" or "set_testbench_simulation_options <*simulator*> -simulator_options <*option_list*>" commands.

The following dofile code builds a list for the Questa SIM simulator "-G" option to override the "data_from_file" parameter of a RAM simulation model that loads data into specified instances of the RAM. The parameter value is the path to a RAM data file compatible with the $readmh Verilog command.

```
set_context patterns -ijtag -rtl

# Global settings
set design_name chip
set_simulation_library_sources -y mem -extension v
set_testbench_simulation_options \
    -simulation_output_directory simulation_outdir_options_cmd

# Create an empty list
set simOptionsList [list]
```

The following code uses a list with an even number of elements to specify the instance paths of two memories in the current design (*instPathInDut*) and their respective data files (*dataFilePath*). The data files contain the data to load into the RAMs, specified as a relative path.

```
foreach {instPathInDut dataFilePath} {
  {\My_m@#m_container_inst /mem1} data1.hex
  {\My_m@#m_container_inst /mem2} data2.hex} {

  # Make the path to the dataFilePath data file
  # absolute because the simulation script exists in a subdirectory:
  set dataFilePathFull [file normalize $dataFilePath]
  # Prepend each instance path in the design with the default path in the testbench:
  set fullInstPathText [string cat "/TB/DUT_inst/" $instPathInDut]
  # Create the full Questa SIM "-G" option to specify the "data_from_file" parameter for
  # each design instance of the memory:
  set gLine [string cat "-G '" $fullInstPathText \
    "/data_from_file'=" $dataFilePathFull ]
  # Append the full "-G" option to the list:
  lappend simOptionsList $gLine
}
```

Show the "-G" options list, one line at a time:

```
puts [join $simOptionsList \n]
```

$simOptionsList should look like the following:

```
-G '/TB/DUT_inst/\My_m@#m_container_inst /mem1/data_from_file'=/
<absolute_path>/data1.hex
-G '/TB/DUT_inst/\My_m@#m_container_inst /mem2/data_from_file'=/
<absolute_path>/data2.hex
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Pass this list directly to the simulator command line using -simulator_options:

```
run_testbench_simulations -design_name ${design_name} \
    -simulator_options $simOptionsList
```

Alternatively, you can set those simulator options globally using the set_testbench_simulation_options command:

```
set_testbench_simulation_options questa \
    -simulator_options $simOptionsList
run_testbench_simulations -design_name $design_name
```

Example 7

This example is an alternative method to the one Example 6 uses. Instead of passing the simulator options on the command line, you can save them to a file and load it. The following dofile shows how to perform this for Questa SIM. First, write the "-G" simulator options into a file, then pass this file to the simulator using the "-f" option.

Tip

Use this method when you have a large number of options to pass to the simulator. There is a character limit to the length of the simulator command line in a simulation script.

```
set_context patterns -ijtag -rtl

# Global settings
set design_name chip
set_simulation_library_sources -y mem -extension v

# Ensure the options file does not already exist
file delete -force simOptionsFile
```

The following code uses a list with an even number of elements to specify the instance paths of two memories in the current design (*instPathInDut*) and their respective data files (*dataFilePath*). The data files contain the data to load into the RAMs, specified as a relative path.

```
foreach {instPathInDut dataFilePath} {
  {\My_m@#m_container_inst /mem1} data1.hex
  {\My_m@#m_container_inst /mem2} data2.hex} {

  # Make the path to the dataFilePath data file
  # absolute because the simulation script exists in a subdirectory:
  set dataFilePathFull [file normalize $dataFilePath]
  # Prepend each instance path in the design with the default path in the testbench:
  set fullInstPathText [string cat "/TB/DUT_inst/" $instPathInDut]
  # Create the full Questa SIM "-G" option to specify the "data_from_file" parameter for
  # each design instance of the memory:
  set gLine [string cat "-G '" $fullInstPathText \
     "/data_from_file'=" $dataFilePathFull ]
  # Append the full "-G" option list to the options file:
  echo $gLine >> simOptionsFile
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The last line above is the difference between supplying the options list to the simulator command line, as in Example 6, and loading the options via a file.

View the *simOptionsList* file:

cat simOptionsFile

```
-G '/TB/DUT_inst/\My_m@#m_container_inst /mem1/data_from_file'=/
<absolute_path>/data1.hex
-G '/TB/DUT_inst/\My_m@#m_container_inst /mem2/data_from_file'=/
<absolute_path>/data2.hex
```

Pass this file directly to the simulator with the "-f" option using the -simulator_options argument of the set_testbench_simulation_options command:

**set_testbench_simulation_options questa **
-simulator_options "-f [pwd]/simOptionsFile"
run_testbench_simulations -design_name ${design_name} -design_id rtl

Related Topics

check_testbench_simulations

report_testbench_simulation_options

set_simulation_library_sources

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 8
Command Dictionary (S)

The Command Dictionary describes the application commands for Tessent Shell. For quick reference, the commands appear alphabetically with each beginning on a separate page.

The command usage line syntax conventions are common to all products documented in this manual. See "Conventions for Command Line Syntax" on page 97.

Command Descriptions 2272
set_abort_limit 2284
set_atpg_fill 2286
set_atpg_limits 2287
set_atpg_timing 2290
set_attribute_options 2294
set_attribute_value 2299
set_au_analysis 2305
set_bidi_gating 2306
set_bist_chain_test 2309
set_bist_debug 2310
set_bist_trace 2314
set_boundary_scan_port_options 2317
set_bsdl_extraction_options 2319
set_burnin_options 2321
set_bus_handling 2323
set_bus_simulation 2326
set_bypass_chains 2328
set_capture_clock 2331
set_capture_procedures 2334
set_capture_timing_options 2336
set_cdp_test_data 2340
set_cdp_test_diagnosis_option 2342
set_cell_library_options 2344
set_chain_test 2351
set_checkpointing_options 2355
set_clock_controller_pins 2357
set_clock_controls 2361
set_clock_gating 2364
set_clock_gating_enable 2367
set_clock_gating_options 2369
set_clock_off_simulation 2371
set_clock_options 2372

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_clock_restriction 2375
set_compactor_connections 2385
set_config_value 2387
set_contention_check 2391
set_context 2396
set_core_instance_parameters 2402
set_critical_area_options 2410
set_current_design 2415
set_current_edt_block 2424
set_current_edt_configuration 2427
set_current_mode 2429
set_current_physical_block 2431
set_current_silicon_insight_setup 2432
set_current_simulation_context 2434
set_dedicated_wrapper_cell_options 2436
set_default_ssn_datapath_configuration 2441
set_defaults_value 2443
set_design_include_directories 2447
set_design_level 2449
set_design_macros 2455
set_design_sources 2457
set_dft_cell_selection_mapping 2461
set_dft_enable_options 2462
set_dft_signals_options 2464
set_dft_specification_requirements 2466
set_diagnosis_options 2473
set_display 2487
set_drc_handling 2489
set_driver_restriction 2498
set_edt_abort_analysis_options 2499
set_edt_finder 2500
set_edt_instances 2502
set_edt_mapping 2505
set_edt_options 2508
set_edt_pins 2521
set_edt_power_controller 2533
set_external_capture_options 2537
set_external_simulator 2540
set_fails_report 2542
set_failure_mapping_options 2543
set_fault_mode 2545
set_fault_sampling 2547
set_fault_selection_options 2549
set_fault_subclass_analysis 2550
set_fault_type 2553

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_flat_model_options . . . 2562
set_flattener_rule_handling . . . 2567
set_gate_level . . . 2570
set_gate_report . . . 2571
set_gzip_options . . . 2606
set_icl_extraction_options . . . 2608
set_icl_network . . . 2610
set_icl_scan_interface_ports . . . 2612
set_icl_scan_mux . . . 2616
set_icl_ssn_datapath_ports . . . 2617
set_iddq_checks . . . 2621
set_ijtag_instance_options . . . 2627
set_ijtag_retargeting_options . . . 2629
set_insertion_options . . . 2642
set_insert_test_logic_options . . . 2651
set_internal_fault . . . 2656
set_internal_name . . . 2657
set_io_mask . . . 2658
set_latch_handling . . . 2659
set_layout_core_instance . . . 2660
set_lbist_controller_options . . . 2661
set_lbist_instances . . . 2666
set_lbist_pins . . . 2668
set_lbist_power_controller_options . . . 2674
set_learn_report . . . 2676
set_lfsr_seed . . . 2678
set_lfsrs . . . 2679
set_license_queue_timeout . . . 2681
set_list_file . . . 2683
set_load_unload_timing_options . . . 2684
set_logfile_handling . . . 2693
set_logical_design_libraries . . . 2695
set_loop_handling . . . 2697
set_lpct_condition_bits . . . 2699
set_lpct_controller . . . 2701
set_lpct_instances . . . 2705
set_lpct_pins . . . 2706
set_lvx_options . . . 2714
set_macrotest_options . . . 2716
set_marker_file_options . . . 2718
set_memory_cluster_library_generation_options . . . 2721
set_memory_cluster_validation_options . . . 2724
set_memory_identification_options . . . 2727
set_memory_instance_options . . . 2729
set_misr_connections . . . 2733

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_module_matching_options 2735
set_multiple_detection 2739
set_multiprocessing_options. 2743
set_net_dominance 2752
set_net_resolution 2754
set_number_shifts 2755
set_observation_point 2757
set_output_masks 2758
set_parallel_load_subchains 2761
set_pathdelay_holdpi 2763
set_pattern_buffer 2764
set_pattern_classification 2765
set_pattern_filtering 2766
set_pattern_source 2778
set_pattern_type 2787
set_physical_translation 2791
set_possible_credit 2793
set_power_control. 2794
set_power_metrics 2803
set_procedure_cycle_checking 2807
set_procedure_retargeting_options 2808
set_procfile_name 2811
set_quick_synthesis_options. 2813
set_ram_initialization 2817
set_random_clocks 2818
set_random_patterns 2820
set_read_verilog_options 2821
set_register_value 2823
set_relevant_coverage. 2825
set_run_synthesis_options 2829
set_scan_chain_options 2832
set_scan_enable 2834
set_scan_insertion_options 2838
set_scan_signals 2845
set_screen_display 2850
set_shadow_check 2851
set_shift_register_identification 2853
set_silicon_insight_option 2855
set_simulation_library_sources 2857
set_simulation_options 2860
set_skewed_load 2865
set_split_capture_cycle 2866
set_ssn_datapath_options 2867
set_ssn_options 2869
set_stability_check 2873

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_static_dft_signal_values 2877
set_static_learning 2881
set_system_mode 2883
set_tcl_shell_options. 2885
set_test_end_icall 2888
set_test_logic 2891
set_test_point_analysis_options 2893
set_test_point_insertion_options 2906
set_test_point_sharing_restrictions. 2909
set_test_point_types 2911
set_test_setup_icall 2913
set_testbench_simulation_options 2918
set_tester_options 2921
set_tied_signals. 2923
set_timing_exceptions_handling 2925
set_tla_loop_handling. 2932
set_tool_options 2934
set_trace_flat_model_options 2936
set_trace_report. 2939
set_transcript_style 2940
set_transient_detection 2943
set_transition_holdpi. 2945
set_tristate_gating 2946
set_tsdb_options 2948
set_tsdb_output_directory. 2950
set_visualizer_preferences 2952
set_wrapper_analysis_options 2953
set_write_patterns_options 2959
set_xbounding_options 2964
set_xclock_handling 2968
set_z_handling 2970
set_zhold_behavior 2972
setenv 2973
shmoo_cdp_test 2974
shutdown_sid_tester 2978
simulate_clock_pulses 2979
simulate_forces 2981
simulate_patterns 2983
sizeof_collection 2986
sort_collection. 2988
start_silicon_insight 2990
stop_silicon_insight 2991
system 2992

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Command Descriptions

This section describes the Tessent Shell commands alphabetically for the letter S. The beginning of each command description shows the contexts and modes that support the command and provides the following information:

Context: Lists each context and sub-context that supports the command. “All contexts” means that the command is supported in all available contexts. “Unspecified” means that you do not have to set a context to use the command.

Mode: Lists each mode that supports the command. “All modes” means that the command is supported in all available modes. In some cases, a particular mode supports the command only within certain contexts, and this is noted in parentheses after the mode type.

Table 8-1. Commands S

Command	Description
set_abort_limit	Specifies the abort limit for the test pattern generator.
set_atpg_fill	Specifies how the tool should fill unspecified bits in ATPG patterns.
set_atpg_limits	Specifies the ATPG process limits at which the tool terminates the ATPG process.
set_atpg_timing	Enables and defines parameters for timing-aware ATPG and fault simulation.
set_attribute_options	Modifies options of any registered user-defined attribute.
set_attribute_value	Sets an attribute's value for the objects specified in ***object_spec***.
set_au_analysis	Specifies whether the ATPG uses the ATPG untestable information to place ATPG untestable faults directly in the AU fault class.
set_bidi_gating	Specifies how bidirectional (bidi) pins are controlled during scan chain shifting to prevent potential bus contention or to ensure an on/off state during testing.
set_bist_chain_test	Disables all capture clock activity for the specified number of patterns in the fault simulation while running the chain test.
set_bist_debug	Sets up a trace of LFSR values during a pattern’s shift cycles.
set_bist_trace	Enables the tracing of PRPG and MISR values associated with each BIST pattern to facilitate signature mismatch debugging and other diagnostics.
set_boundary_scan_port_options	Sets options relative to boundary scan on ports of the design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-1. Commands S (cont.)

Command	Description
set_bsdl_extraction_options	Sets values for various BSDL attributes in the BSDL file extracted using the BSDL extraction flow.
set_burnin_options	Configures the burn-in pattern output.
set_bus_handling	Specifies the bus contention results that you desire for the identified buses.
set_bus_simulation	Specifies whether the tool uses global or local bus simulation analysis.
set_bypass_chains	Specifies a list of scan chains to connect in series to form a particular EDT bypass chain.
set_capture_clock	Specifies the capture clock for random pattern simulation or, optionally, for all ATPG pattern simulation.
set_capture_procedures	Enables or disables the tool's use of a specified subset of named capture procedures when generating test patterns.
set_capture_timing_options	Sets the capture_clock period and other timing aspects of the capture procedure during SSN scan operations for the current physical block.
set_cdp_test_data	Sets or changes data values in a specified characterization and debug package (CDP) test.
set_cdp_test_diagnosis_option	Enables including or excluding MemoryBist instruments from the bitmap diagnosis report for a specific TestStep wrapper or all TestStep wrappers of the specified CDP test, after the CDP diagnostic patterns are generated.
set_cell_library_options	Sets the specified options for library parsing and design elaboration.
set_chain_test	Specifies binary sequences for chain test.
set_checkpointing_options	Enables checkpointing and specifies the associated options.
set_clock_controller_pins	Specifies the connection information for the clock controller pins.
set_clock_controls	Enables or disables the clock control definitions in the test procedure file.
set_clock_gating	Specifies clock gating cells whose unconnected test-enable ports need to be connected to either the scan enable signal or a specified signal (pin).
set_clock_gating_enable	Associates clocks that control clock gating devices with enable signal drivers.
set_clock_gating_options	Specifies clock gater handling options for scan insertion.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-1. Commands S (cont.)

Command	Description
set_clock_off_simulation	Enables or disables the simulation where all clock primary inputs are at their "off" value, other primary inputs have been forced to values, and state elements are at the values scanned in or resulting from capture in the previous cycle.
set_clock_options	Enables you to specify multiple options for a single clock.
set_clock_restriction	Specifies whether ATPG can create patterns with more than one active capture clock.
set_compactor_connections	Specifies the scan chains for the tool to connect to a particular scan channel output.
set_config_value	Sets the value of an element in the configuration data.
set_contention_check	Specifies whether or not contention checking is on and the conditions under which checks are performed. Contention checking is set to On when the tool is invoked.
set_context	Specifies the current usage context of Tessent Shell.
set_core_instance_parameters	Changes the parameters of instruments previously added with the add_core_instances command.
set_critical_area_options	Enables you to specify options for the critical area handling.
set_current_design	Specifies the top level of the design for all subsequent commands until reset by another run of this command.
set_current_edt_block	Directs the tool to apply certain subsequent commands to a particular EDT block only.
set_current_edt_configuration	Sets a particular compression configuration for the tool to use within the current design block.
set_current_mode	Specifies the current test mode in pattern retargeting and core mapping flows.
set_current_physical_block	Set the current physical block where all subsequent load/unload and capture timing settings are applied.
set_current_silicon_insight_setup	Sets the currently selected setup in the Tessent SiliconInsight Setup Specification.
set_current_simulation_context	Sets the current simulation context.
set_dedicated_wrapper_cell_options	Controls how the tool infers the dedicated wrapper cells (DWCs) for primary IOs.
set_default_ssn_datapath_configuration	Adds an override setting to ICL ports, ICL pins, or ICL ScanRegisters, to override the default SSN datapath configuration.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-1. Commands S (cont.)

Command	Description
set_defaults_value	Sets the default value of a property in one of the DefaultsSpecification (policy) wrappers.
set_design_include_directories	Specifies one or more directory paths that contain include files for use by subsequent read_verilog commands.
set_design_level	Specifies the level of the current design.
set_design_macros	Specifies one or more defined macros for use by subsequent commands.
set_design_sources	Specifies where to search for module definitions during module matching.
set_dft_cell_selection_mapping	Maps a library cell to one or more power domains.
set_dft_enable_options	Enables or disables static control signals implemented using an IJTAG TDR inside the LogicBIST controller.
set_dft_signals_options	Specifies certain implementation details for DFT signals.
set_dft_specification_requirements	Specifies design rule requirements and instrument types.
set_diagnosis_options	Specifies the type of diagnosis performed by the diagnose_failures command.
set_display	Sets the DISPLAY environment variable from the tool's command line.
set_drc_handling	Specifies how the tool handles design rule violations.
set_driver_restriction	Specifies whether multiple drivers on buses and multiple active ports on gates can be used.
set_edt_abort_analysis_options	Specifies the number of EDT Aborted (EAB) test cubes on which to perform EAB fault analysis.
set_edt_finder	Controls whether the tool automatically identifies EDT logic and updates scan chain pins during test pattern generation.
set_edt_instances	Specifies the location in which the tool places the EDT logic.
set_edt_mapping	Enables or disables automatic mapping of block-level dofile commands so block-level dofiles can be reused without modification for top-level pattern creation.
set_edt_options	Determines whether compression is on and specifies parameters for the EDT logic, including scan channels, bypass circuitry, lockup cells, pipeline stages, and compactor type.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-1. Commands S (cont.)

Command	Description
set_edt_pins	By default, all EDT pins are created as dedicated pins with default names and no inversions. The same set_edt_pins command must be used for both the pattern generation phase and the EDT logic creation phase to generate patterns correctly for modified pins.
set_edt_power_controller	Generates and inserts a power controller in the compression logic to enable low-power shift.
set_external_capture_options	Specifies the number of tester cycles to be used in the capture window of all patterns.
set_external_simulator	Specifies the shell command to use to invoke an external simulator for verifying tool-produced patterns.
set_fails_report	Specifies whether the design rules checker displays clock rule failures.
set_failure_mapping_options	Specifies filters you can apply when performing reverse mapping of top-level failures to the core in hierarchical designs flows.
set_fault_mode	Specifies whether the fault mode is collapsed or uncollapsed.
set_fault_sampling	Specifies the fault sampling percentage used for circuit evaluation or scan identification.
set_fault_selection_options	Controls fault selection options for the write_faults, report_faults, and delete_faults commands.
set_fault_subclass_analysis	Toggles the analysis for the specified fault class(es).
set_fault_type	Specifies the fault model type for ATPG.
set_flat_model_options	Controls the removal of unused library cell input ports in the flat model. Enables loadboard loopback emulation.
set_flattener_rule_handling	Specifies how the tool globally handles flattening violations.
set_gate_level	Specifies the hierarchical level of gate reporting and displaying.
set_gate_report	Specifies the information displayed by the report_gates command and in the Tessent Visualizer tabs.
set_gzip_options	Specifies GNU gzip options to use with the GNU gzip command.
set_icl_extraction_options	Customizes the behavior of ICL extraction.
set_icl_network	Context: patterns -silicon_insight -ijtag
set_icl_scan_interface_ports	Sets the names of the ports in the specified scan interface.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-1. Commands S (cont.)

Command	Description
set_icl_scan_mux	Context: patterns -silicon_insight -ijtag
set_icl_ssn_datapath_ports	Specifies the ports that belong to a given SSN datapath and defines the order of the SSN bus data ports.
set_iddq_checks	Specifies the restrictions and conditions to use when creating or selecting patterns for detecting IDDQ faults.
set_ijtag_instance_options	Sets options on instances for which there is a matched ICL module.
set_ijtag_retargeting_options	Enables you to configure different aspects of IJTAG retargeting.
set_insertion_options	Specifies insertion option values that affect the behavior of the insertion commands.
set_insert_test_logic_options	Sets new prefix or infix values to use during test logic insertion.
set_internal_fault	Specifies whether the tool reports faults within library models.
set_internal_name	Specifies whether to delete or keep pin names of library internal pins containing no-fault attributes.
set_io_mask	Modifies the behavior of bidirectional pins so that their expected values are always X during test cycles in which the primary input portion of the bidirectional pin is being forced.
set_latch_handling	Specifies whether the tool considers non-transparent latches for scan insertion while test logic is turned on.
set_layout_core_instance	For hierarchial layout-aware diagnosis, sets the current layout core instance for inter-scan cell logic reporting.
set_lbist_controller_options	Specifies global options to configure the LogicBIST controller.
set_lbist_instances	Specifies the instance in which the LogicBIST controller or single chain mode logic is placed.
set_lbist_pins	Specifies the connection information for LogicBIST controller pins and enables renaming LogicBIST pins.
set_lbist_power_controller_options	Specifies creation of the low-power shift controller for LogicBIST.
set_learn_report	Specifies whether the report_gates command can display the learned behavior for a specific gate.
set_lfsr_seed	Sets the initial value of a PRPG or a MISR.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-1. Commands S (cont.)

Command	Description
set_lfsrs	Changes the *shift_type* and *tap_type* default settings for the add_lfsrs and add_lfsr_taps commands.
set_license_queue_timeout	Specifies the amount of time that the tool queues for one or more required licenses when not immediately available.
set_list_file	Specifies the name of the list file into which the tool places the pins' logic values during simulation.
set_load_unload_timing_options	Sets the scan shift, SSN bus clock period, and other timing aspects of the load/unload procedure during SSN or LogicBIST scan operations for the current physical block.
set_logfile_handling	Specifies for the tool to direct the transcript information to a file.
set_logical_design_libraries	Defines the set of available logical libraries that can be used by the read_verilog and read_vhdl -in_library option.
set_loop_handling	Specifies how the tool handles feedback networks.
set_lpct_condition_bits	Specifies the condition scan cells (condition bits) in the design core to connect to the Type 3 low pin count test (LPCT) controller.
set_lpct_controller	Enables the creation of and specifies the configurations for a low pin count test (LPCT) controller for use with EDT logic.
set_lpct_instances	Inserts the low pin count test (LPCT) controller and clock gater in a specified hierarchical instance within the design core.
set_lpct_pins	Specifies control pins and connections between the low pin count test (LPCT) controller and the core design.
set_lvx_options	Configures Tessent LVX (Laser Voltage Imaging/Probing) settings.
set_macrotest_options	Prepares a black box macro to enable MacroTest conversion of patterns at the boundaries of that instance.
set_marker_file_options	Specifies settings for the marker file generated by the extract_fault_sites and extract_inter_cell_data commands.
set_memory_cluster_library_generation_options	Configures the automation of Shared Bus physical to logical (P2L) mapping and logical memory TCD file generation.
set_memory_cluster_validation_options	Enables Shared Bus memory cluster library validation.
set_memory_identification_options	Sets options and pattern expressions to validate memory TCD are associated with memory modules to be tested by MemoryBIST.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-1. Commands S (cont.)

Command	Description
set_memory_instance_options	Set memory instance options for design rules and design specifications.
set_misr_connections	Describes the connections between the scan chain outputs and the MISR.
set_module_matching_options	Defines the acceptable prefixes and suffixes or regular expressions to use when matching an ICL or a TCD module to a design module within ICL extraction.
set_multiple_detection	Enables the multiple detection of faults.
set_multiprocessing_options	Sets the values of several variables that affect multiprocessing commands for ATPG or simulation.
set_net_dominance	Specifies the fault effect of bus contention on tri-state nets.
set_net_resolution	Specifies the behavior of multi-driver nets.
set_number_shifts	Sets the number of shifts for loading or unloading the scan chains.
set_observation_point	Specifies the observation point for random pattern fault simulation.
set_output_masks	Ignores any fault effects that propagate to the primary output pins you select.
set_parallel_load_subchains	Specifies whether the internal parallel access functionality is enabled or disabled.
set_pathdelay_holdpi	Specifies whether the ATPG keeps non-clock primary inputs at a constant state after the first force.
set_pattern_buffer	Stores run-time pattern data in temporary files in the directory specified.
set_pattern_classification	Specifies the order in which to store test patterns.
set_pattern_filtering	Creates a temporary set of sampled scan patterns.
set_pattern_source	Specifies the source of the test patterns.
set_pattern_type	Specifies the type of pattern(s) used by ATPG simulation.
set_physical_translation	Translates special Verilog characters to SPICE-readable characters that Calibre applications can read.
set_possible_credit	Specifies the percentage of credit that the tool assigns possible-detected faults.
set_power_control	Enables low-power ATPG and sets up a shift or capture threshold.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-1. Commands S (cont.)

Command	Description
set_power_metrics	Calculates shift and capture power during test pattern generation and simulation.
set_procedure_cycle_checking	Enables test procedure cycle timing checking to be done immediately following scan chain tracing during design rules checking.
set_procedure_retargeting_options	Controls the behavior of the automatic merging and mapping (retargeting) of core-level load_unload and shift procedures to top-level test procedures and mapping of core-level test_setup and test_end IJTAG procedures.
set_procfile_name	Specifies a new procedure file for the tool to process at a later time.
set_quick_synthesis_options	Sets options for quick synthesis.
set_ram_initialization	Specifies whether to initialize RAM and ROM gates that do not have initialization files.
set_random_clocks	Specifies whether Tessent FastScan uses combinational or clock_sequential patterns for random pattern simulation.
set_random_patterns	Specifies the number of random patterns that the tool simulates.
set_read_verilog_options	Sets certain options globally for subsequent read_verilog commands.
set_register_value	Enables you to specify the register value variable outside of Setup mode.
set_relevant_coverage	Determines which sub-class(es) of untestable faults are included in or excluded from fault coverage reports.
set_run_synthesis_options	Specifies the default values for options that can be set for the run_synthesis command.
set_scan_chain_options	Specifies options for managing scan chains.
set_scan_enable	Assigns scan_enable signals to specific scan chains.
set_scan_insertion_options	Specifies the high-level options for the scan insertion process.
set_scan_signals	Sets the input control signals' drivers used by the insert_test_logic command.
set_screen_display	Specifies whether the tool writes the transcript to the session window.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-1. Commands S (cont.)

Command	Description
set_shadow_check	Specifies whether the tool identifies sequential elements as "shadow" elements during scan chain tracing.
set_shift_register_identification	Enables or disables shift register identification. The default is ON.
set_silicon_insight_option	Specifies optional settings for running Tessent SiliconInsight.
set_simulation_library_sources	Defines the directories and files in which to search for the simulation Verilog library cells that are to be used by the run_testbench_simulations command.
set_simulation_options	Controls the behavior of the tool during pattern generation and simulation.
set_skewed_load	Specifies whether the tool includes a skewed load in the patterns.
set_split_capture_cycle	Controls whether the tool updates simulation data between clock edges.
set_ssn_datapath_options	Use this command to specify options controlling SSN datapaths.
set_ssn_options	Enables or disables SSN on the current physical block or design.
set_stability_check	Specifies whether the tool checks the effect of applying the main shift procedure on non-scan cells.
set_static_dft_signal_values	Sets the static DFT signal that exists in the design such that the signal can be used as part of a test_setup.
set_static_learning	Specifies whether the tool performs the learning analysis to make the ATPG process more efficient.
set_system_mode	Specifies the operational state you want the tool to enter.
set_tcl_shell_options	Specifies options for the Tcl shell.
set_test_end_icall	Adds an iCall to the test_end procedure.
set_test_logic	Inserts test logic to control the set, reset, clock, enable, or write control signals to make them scannable when scan chains are inserted.
set_test_point_analysis_options	Sets the maximum number of test points, the breakdown in control and observe points, the target fault coverage and the number of pseudo random patterns to be applied and some other parameters that are taken into account during test point analysis.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-1. Commands S (cont.)

Command	Description
set_test_point_insertion_options	Sets parameters related to test point insertion.
set_test_point_sharing_restrictions	Specify sharing restrictions during test point insertion.
set_test_point_types	Specifies the type of test points to insert in the design.
set_test_setup_icall	Adds an iCall to the end of the test_setup procedure.
set_testbench_simulation_options	Specifies the default values for a set of options that exist in the run_testbench_simulations command.
set_tester_options	Enables you to define tester-specific options that control how the tool writes tester patterns when you use SSN.
set_tied_signals	Changes the default value for floating pins and floating nets which do not have assigned values.
set_timing_exceptions_handling	Changes the default handling of timing exception paths (false paths and multicycle paths) read by the read_sdc command.
set_tla_loop_handling	Specifies how to simulate feedback loops that contain transparent latches (TLAs).
set_tool_options	Sets certain options for Tessent Shell.
set_trace_flat_model_options	Modifies the default behavior of the trace_flat_model command.
set_trace_report	Specifies whether the tool displays gates in the scan chain trace.
set_transcript_style	Specifies the style in which the tool transcribes commands.
set_transient_detection	Specifies whether the tool detects all zero width events on the clock lines of state elements.
set_transition_holdpi	Specifies for the tool to freeze all primary input values other than clocks and RAM controls during multiple cycles of pattern generation.
set_tristate_gating	Specifies how tri-state devices are controlled during scan chain shifting.
set_tsdb_options	Modifies options for how a file path located within the design source dictionary file is interpreted.
set_tsdb_output_directory	Sets the TSDB output directory used by the process_dft_specification command.
set_visualizer_preferences	Controls the Tessent Visualizer display preferences for the Flat Schematic and Hierarchical Schematic windows.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-1. Commands S (cont.)

Command	Description
set_wrapper_analysis_options	This command sets up parameters for the wrapper cells analysis.
set_write_patterns_options	Creates the subset of ports or vector callbacks to use with the write_patterns command.
set_xbounding_options	Sets up the parameters for X bounding insertion. The analysis is triggered by the analyze_xbounding command.
set_xclock_handling	Specifies whether the sequential element model outputs X when any of its clock inputs become X.
set_z_handling	Specifies the simulation handling for high impedance signals on internal and external tri-state nets.
set_zhold_behavior	Specifies whether ZHOLD gates retain their state values.
setenv	Sets a shell environment variable within the tool environment.
shmoo_cdp_test	Generates a two-dimensional shmoo plot for a characterization and debug package (CDP) test.
shutdown_sid_tester	Terminates the currently active SID tester process that was created by executing the launch_sid_tester command.
simulate_clock_pulses	Pulses one or more clocks within the current simulation context.
simulate_forces	Simulates the queued forces in the current simulation context.
simulate_patterns	Performs simulation by applying the specified pattern source.
sizeof_collection	Returns the number of objects in a collection.
sort_collection	Sorts a collection based on the value of one or more attributes, resulting in a new sorted collection.
start_silicon_insight	Starts Tessent SiliconInsight and initializes the data structures.
stop_silicon_insight	Stops Tessent SiliconInsight.
system	Passes the specified command to the operating system for execution.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_abort_limit

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the abort limit for the test pattern generator.

Usage

set_abort_limit ***comb_abort_limit*** [*seq_abort_limit*]

Description

The set_abort_limit command performs slightly differently depending on which tool you are using. Use this command when there are some remaining undetected faults and the test coverage is still too low. By increasing the abort limit, you can enable the tool to detect those remaining undetected faults, and thereby raise the coverage. The following paragraphs describe how the command operates for each tool.

Tip

If you use the create_patterns command, you do not need to issue this command. This is because create_patterns automatically determines an appropriate abort limit.

The set_abort_limit command specifies, for combinational and clock sequential test generation, the maximum number of attempts the test pattern generator permits before aborting a fault. If the limit is too low, the test pattern generator may abort too many faults and fault coverage could be too low. However, if the limit is too high, it may take too much time to complete the test generation process. The default combinational abort limit value is 30.

During the test generation process for a given fault, the test pattern generator attempts combinational or ram sequential test generation first. If this fails to create a test or prove redundancy, and you specified a non-zero sequential depth by using the set_pattern_type command, the test pattern generator performs clock sequential test generation. The default abort limit for clock sequential test generation is the same as that for the combinational test generation (30).

Arguments

- ***comb_abort_limit***

  A required integer that specifies the maximum number of conflicts for each target fault that the test pattern generator permits during the combinational test generation process. The default is 30.

  If you set the combinational abort limit to 0 and the test pattern generator can perform clock sequential test generation, the generator does not perform combinational test generation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- *seq_abort_limit*

  An optional integer that specifies the maximum number of conflicts for each target fault that the test pattern generator permits during the clock sequential test generation process. The default is the current *comb_abort_limit* default.

  If you set the sequential abort limit to 0, the test pattern generator does not perform clock sequential test generation.

Examples

The following example performs an ATPG run, then continues the run with a higher abort limit for the maximum number of permitted conflicts:

```
set_system_mode analysis
create_patterns
set_abort_limit 100
create_patterns
```

Related Topics

create_patterns

report_aborted_faults

report_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_atpg_fill

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies how the tool should fill unspecified bits in ATPG patterns.

Usage

set_atpg_fill {**Random** | **X** | **0** | **1**}

Description

Every pattern generated by ATPG has deterministic bits and unspecified bits. This command enables you to specify that the tool fills those unspecified bits with random values (the default) or X's, 0's, or 1's. This command is useful for getting the test cube that the ATPG generated to test a given fault by setting the fill to X.

Note, this command is only valid in Tessent TestKompress when EDT is set to off with the "set_edt_options -off" command.

Arguments

- **Random** | **X** | **0** | **1**

  A required literal that instructs the tool how to fill unspecified bits in ATPG patterns.

Examples

The following example fills unspecified bits in ATPG patterns with X's:

```
set_atpg_fill X
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_atpg_limits

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the ATPG process limits at which the tool terminates the ATPG process.

Usage

set_atpg_limits {OFf | {[-Cpu_seconds [OFf | *integer*]] [-Pattern_count [OFf | *integer*]] [-Test_coverage [OFf | *real* | {*real integer*} ...]]}}

Description

The set_atpg_limits command determines the limitations under which the ATPG process operates. Upon invocation of the tool, all the command option limitations are off. If you set any of the limitations, and during an ATPG run the tool reaches one of those limits, the tool terminates the ATPG process. You can use any combination of the three arguments (limitation types). When you specify more than one limitation type, the tool terminates the ATPG process when any of the specified limitation types reaches its limit.

You can check the current settings of the set_atpg_limits command by using the report_environment command.

Arguments

- OFf

  An optional literal that turns off all ATPG process limits previously set with this command.

- -Cpu_seconds OFf | *integer*

  An optional switch and argument pair that specifies the maximum number of CPU seconds that any future ATPG process can consume before the tool terminates the process. The argument choices are as follows:

  OFf — A literal specifying that there is no limit to the amount of CPU time the ATPG process consumes during an ATPG process. This is the invocation default.

  integer — A positive integer that specifies the maximum number of CPU seconds that the tool can consume during an ATPG process. When the tool reaches the maximum, it terminates the ATPG process.

- -Pattern_count [OFf | *integer*]

  An optional switch and argument pair that specifies the maximum number of test patterns that any future ATPG process can generate before the tool terminates the process. The argument choices are as follows:

  OFf — A literal specifying that there is no limit to the number of test patterns the ATPG process generates during an ATPG process. This is the invocation default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

integer — A positive integer that specifies the maximum number of test patterns that the tool can generate during an ATPG process. When the tool reaches the maximum, it terminates the ATPG process.

- -Test_coverage [OFf | *real* | {*real integer*} ...]

  An optional switch and argument pair that specifies the maximum percentage of test coverage that any future ATPG process needs to reach before the tool terminates the process. The argument choices are as follows:

  OFf — A literal specifying the 100 percent test coverage limit during an ATPG process. The tool terminates the ATPG process when either 100 percent coverage is attained or when the ATPG process has completed. This is the invocation default.

  real — A positive real number that specifies the maximum percentage of test coverage that the tool should achieve during an ATPG process (value range is 0 to 100.0). When the tool reaches the maximum, it terminates the ATPG process.

  {real integer} — One or more pairs of real and integer numbers that specifies the multiple detect coverage. For example, "-test_coverage 90.5 2" specifies the multiple detect test coverage as 90.5 for multiple detection of 2. When you specify multiple list pairs, such as "-test_coverage 95 1 85 5," ATPG stops at 95 percent coverage in the first pass targeting one detect. Then in the second pass, ATPG stops at 95 percent or if it completes before 95 percent. This continues for passes 3 and 4. Pass 5 (targeting 5-detect) stops when ATPG reaches 85 percent or if it completes before 85 percent. When you specify multiple list pairs, you cannot associate a higher detect number with a higher test coverage limit. When you specify multiple test coverage limits, the tool terminates the ATPG process only when all limits are met.

Examples

Example 1

The following example sets two of the three limits on the ATPG process. It then shows the relevant setup data from the report_environment command:

set_atpg_limits -cpu_sec 500 -test_coverage 99.5 -pattern_count 100000
report_environment

```
...
atpg limits =     500.00 sec 95.50% coverage 100000 patterns
...
```

If the ATPG process reaches either of these two limits, the process terminates. Notice that the information from the report_environment command only shows the settings that are different from the invocation defaults of Off.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example shows ATPG to target 5-detect with 99 percent test coverage as first detect limit and 95 percent multiple detect test coverage limit for the 2-detect to 5-detect:

```
set_multiple_detection -guaranteed_atpg_detections 5
set_atpg_limits -test_coverage 99 1 95 2
report_environment
create_patterns
```

Related Topics

report_environment

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_atpg_timing

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables and defines parameters for timing-aware ATPG and fault simulation.

Usage

set_atpg_timing
[OFf | ON]
[[-CLock_waveform {DEFAULT | *clock_pin_pathname* | *clock_gate_id*} {*clock_waveform*}]...]
[[-DELEte_clock_waveform {ALL | *clock_pin_pathname* | *clock_gate_id*}]...]
[-SLack_margin_for_fault_dropping {OFf | *slack_margin* | *slack_margin_percentage*%}]
[-slack_margin_for_timing_critical_faults {OFf | *slack_margin* | *slack_margin_percentage*%}]
[-TIMING_CRITICAL {OFf | *timing_margin%*}]
[-EXCLUDE_INTER_CLOCK_PATHS {ON | OFf}]

Description

For complete information, see "Timing-Aware ATPG" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- OFf | ON

  An optional literal that enables or disables timing-aware ATPG.

- -CLock_waveform

  An optional switch that defines a clock waveform for evaluating slack and test metrics.

  With the exception of SDQL, timing-aware ATPG uses the test clock frequency for the clock waveform you specify with this switch. The test clock frequency is the ATE testing clock frequency, which may not be the same as the normal operation frequency (System Clock).

  When you issue the report_atpg_timing command, the tool reports both the test clock and system clock.

  Note

  The timing information from the test procedure file is not used. You must define the clock information for all clocks in the design, even for those not used for ATPG (not used in a named capture procedure).

  There are two parameters associated with the -CLock_waveform switch:

  DEFAULT | *clock_pin_pathname* | *clock_gate_id*

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

A string that identifies the clock pin either by pin pathname or gate id. Use the DEFAULT keyword to define a waveform for all clocks that are not explicitly defined. If you define multiple waveforms for a pin or gate id, the last specified waveform is used.

clock_waveform

Defines the waveform of the clock. The clock waveform is used for the test clock domain.

The *clock_waveform* includes three floating numbers in the following order, separated by commas:

clock_period, *offset*, *pulse_width*

The units are the same as what is specified in the SDF file, which defaults to 1 ns.

- -DELEte_clock_waveform {ALL | *clock_pin_pathname* | *clock_gate_id*}

  An optional switch and string or literal pair that deletes a clock waveform that was previously defined by the -clock_waveform switch. Use the ALL keyword to delete all explicitly defined clock waveforms. The DEFAULT clock waveform is not deleted by the ALL keyword.

- -SLack_margin_for_fault_dropping [OFF | *slack_margin* | *slack_margin_percentage*%]

  An optional switch and literal or numerical pair that specifies the fault dropping criteria in addition to the fault detection.

  The default setting for this switch is 50 percent. This means that for faults detected by fault simulation, if the testable minimum slack (Tms) is more than 50 percent of the actual test slack (Ta), then the fault is dropped. When ATPG targets a fault, that fault is effectively dropped no matter what fault dropping criteria is in effect.

  OFF

  Disables the additional dropping criteria, so the detection by the fault simulation is the same as the regular transition faults.

  slack_margin

  When a real number “r” is specified without a percentage sign (%), it is treated as the slack margin for the fault dropping condition. A fault is dropped from the target list when the fault is detected and the following condition holds:

  (Ta-Tms) < r

  Where Ta is the slack used by ATPG, and Tms is the testable minimum slack with respect to the test clock.

  slack_margin_percentage%

  If a real number “r”, ranged in [0, 100.0], is followed by a percentage sign (%), it is treated as the slack margin percentage for the fault dropping condition. A fault is dropped from the target list when the fault is detected and the following condition holds:

  (Ta-Tms)*100/ Ta < r

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Where Ta is the slack used by ATPG, and Tms is the testable minimum slack with respect to the test clock.

If set to 100 percent, faults are dropped regardless of slack.

- -slack_margin_for_timing_critical_faults {OFf | *slack_margin* | *slack_margin_percentage%*}

  An optional switch and literal or numerical pair that specifies the timing-critical fault margin. To specify a margin value, use either an exact, non-negative margin value (*slack_margin*) or a percentage value followed by a percent sign (%), which selects the timing-critical faults as the faults where the value of (*static_slack*/*clock_period*) × 100 is less than the supplied *slack_margin_percentage* value. The "off" value (default) disables the slack margin for timing-critical faults, which means that the tool treats all faults as timing-critical.

- -TIMING_CRITICAL {OFf | *timing_margin%*}

  An optional switch and literal or numerical pair that selects timing-critical faults or disables specifically selecting them. Before running ATPG, the static longest delay path is calculated for each fault. The *timing_margin* is calculated using the following equation:

  Timing_margin% = (static_longest_delay_path) / (static_longest_delay_path + slack) * 100%

  If the *timing_margin* for a fault is greater than the specified value, the fault is selected. For example, assume there are two faults (F1 and F2) with static longest delay paths of 8.5 ns and 9.5 ns, respectively. The clock period is 10 ns, which creates a *timing_margin* of 85 percent for F1 and 95 percent for F2. If you specify "-timing_critical 90%", F2 is selected and F1 is not.

  ATPG still runs with the complete fault list, but the *slack_margin_for_fault_dropping* is applied only to the selected timing-critical faults. Selecting a large *timing_margin* (90 percent) is recommended to speed up ATPG. If you want to target only selected timing-critical faults, follow the command sequence described in "Run Time Reduction for Timing-Aware ATPG" in the *Tessent Scan and ATPG User's Manual*.

- -EXCLUDE_INTER_CLOCK_PATHS {ON | OFf}

  An optional switch and literal pair that instructs the tool to exclude inter-clock paths during static timing analysis for calculating the path delay through every gate pin.

Examples

The following example enables timing-aware ATPG and defines the clock information:

```
set_atpg_timing -clock test_clk 16000 8000 8000
set_atpg_timing -clock default 36000 18000 18000
set_atpg_timing on
```

Related Topics

report_atpg_timing

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_statistics

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_attribute_options

Context: unspecified, all contexts

Mode: all modes

Modifies options of any registered user-defined attribute.

Usage

set_attribute_options **-name *attribute_name_list*** [-object_types *type_list*]
[-export_during_write {auto | off | on}]
[-preserve_boundary_in_flat_model {off | on}]
[-applies_to_child_instances {off | on}]
[-display_in_gui {off | on}]
[-gui_marking_index *marking_index*] [-silent]

Description

The set_attribute_options command modifies the -applies_to_child_instances, -preserve_boundary_in_flat_model, -display_in_gui, and -gui_marking_index options of any registered user-defined attribute.

Arguments

- **-name *attribute_name_list***

  A required switch and value pair that specifies an attribute name or a Tcl list of attribute names whose options are to be modified.

- -object_types *type_list*

  *A*n optional switch and value pair that specifies the object type or a Tcl list of design object types whose attribute configuration settings are to be modified. The attributes of object types not included in this list remain unaffected. If this argument is omitted, by default, the attribute options of all design object types for which the attributes are registered on are modified.

- -export_during_write auto | off | on

  An optional switch and literal pair that specifies which module and port ICL attributes are written into the ICL output when a module is written out using the write_icl command. Any attribute that does not have this switch set is written out in the same form as it was read in; any attribute value set with the set_attribute_value command since being read in is ignored.

  You should use this default behavior when you want to write out the attribute as it was read in. You can modify the default behavior using the following literals:

  auto — Use auto when you want to update the value of an attribute that was read in with the ICL source. In this case, the value read in from ICL is written out unless an alternative value has been set using the set_attribute_value command, which is written out instead. If the ICL source did not have an attribute definition, the value set using the set_attribute_value command is ignored and not written out when the ICL is written.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

on — Use on when you want to update an attribute that was read in with the ICL source, or to cause a new ICL attribute definition to be created for that attribute when the ICL is written. The attribute value set using the set_attribute_value command is written out when the ICL is written out. However, the attribute definitions that were not read in from the ICL source are also written out.

off — Use off when you want an attribute to be ignored and omitted as the ICL is being written out. The value read in from ICL and any subsequent attribute value set using the set_attribute_value is ignored when the ICL is written out. The output ICL file does not have an attribute definition for this attribute.

The -export_during_write switch cannot be changed for predefined attributes that reflect the state of internal data such as the direction of a pin. For predefined storage attributes which hold additional data for some objects, this switch can be promoted from off to auto and from off to on, or from auto to on, but not in the other direction.

- -preserve_boundary_in_flat_model off | on

  An optional switch that specifies whether to preserve certain hierarchical pins or nets and their attributes during design flattening based on an attribute of a module, instance, port, pin, or net. The -preserve_boundary_in_flat_model only works with hierarchical design objects: modules, instances, ports, pins, and nets. If you attempt to use this option with objects other than hierarchical design objects, then the tool issues a warning and ignores the option.

  For module attributes, this option preserves all pins of all instances of the module, if the attribute is set to a non-default value. For instance attributes, this option preserves all pins of the instance, if the attribute is set to a non-default value. For port attributes, this option preserves only the corresponding pin on each instance of the port's module, if the attribute is set to a non-default value. These preserved hierarchical pins become buffers in the flat model and are no-faulted. By default, hierarchical pins are not preserved. For net attributes, this option preserves the net boundary with a buffer in the flat model which drives the original fanout of the net and provides a Gate_pin having the name of the net.

  If you set an attribute on a pin of a design model, instance, port, pin, or net by enabling this option, the flat model needs to be reconstructed. The attribute is not lost during flattening.

  Note that by default, only the boundaries of library cells are preserved in the flat model. (Those are the locations where faults are added for ATPG, for example.) So the boundaries of higher-level modules in the design are not preserved in the flat model by default. When you use the “-preserve_boundary_in_flat_model on” switch, each hierarchical pin or net and boundary of an instance is preserved in the flat model with place-holder buffers corresponding to each pin or net. A gate_pin object then exists in the flat model to match the hierarchical pin or net. The instance names of the pin or net and gate_pin are the same.

  To preserve the boundary of a specific hierarchical design object in the flat model, register an attribute using the register_attribute command, configure this attribute using “set_attribute_options -preserve_boundary_with_flat_model on,” then apply it to the design objects using the set_attribute_value command. Alternatively, you may apply the "preserve_boundary" pre-registered attribute to design objects using the set_attribute_value

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

command, if you do not want to register and configure your own attribute. See "Example 5" on page 2303 for details.

- -applies_to_child_instances off| on

  An optional switch that defines if the attribute is also visible on the instances below the instances or the instances of the modules on which the attribute is defined. This option can only be set to true on attributes defined for objects of type instance or module.

- -display_in_gui {off|on}

  An optional switch and literal pair that specifies whether the attribute(s) specified by the name argument are visible in the Flat and Hierarchical Schematic windows of Tessent Visualizer.

  off — The attribute is not visible in the Flat and Hierarchical Schematic windows.

  on — The attribute is visible in the Flat and Hierarchical Schematic windows.

- -gui_marking_index *marking_index*

  An optional switch and value pair that specifies the marking index that Tessent Visualizer uses when displaying the attribute(s). The marking index is an integer (1-10) that maps to one of ten colors. Tessent Visualizer uses the specified color as a background when it displays the attribute. The default is 1, which maps to light blue.

- -silent

  An optional switch that suppresses error messages if one or more attributes in attribute_name_list do not exist.

Examples

Example 1

The following example shows how to configure the attribute named power_domain in order for it to be visible in all child instances of the instance for which it is defined:

```
set_attribute_options -name power_domain -object_types instance
        -applies_to_child_instances on
```

Example 2

This example demonstrates how an attribute value is written or not written into the ICL module representation based on the value of the set_attribute_options -export_during_write setting. This example assumes that the following ICL module has been read in:

```
Module  block {
   DataInPort din;
   DataOutPort dout;
   Attribute speed = "100";
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Executing the following commands in Tessent Shell, using the default set_attribute_options settings, produces the module definition that follows:

register_attribute -name speed -object_type icl_module -value_type string
set_attribute_value [get_icl_modules block] -name speed -value "200"
register_attribute -name added_by -object_type icl_module -value_type string
set_attribute_value [get_icl_modules block] -name added_by -value "John"

 # report the ICL module with the default attribute export_during_write setting
report_icl_modules -module block

```
Module  block {
   DataInPort din;
   DataOutPort dout;
   Attribute speed = "100";
}
```

Executing the following additional commands in Tessent Shell, using the set_attribute_options auto setting, produces the module definition that follows:

 # set the attribute to auto export
set_attribute_options -name speed -export_during_write auto
set_attribute_options -name added_by -export_during_write auto
report_icl_modules -module block

```
Module  block {
   DataInPort din;
   DataOutPort dout;
   Attribute speed = "200";
}
```

Executing the following additional commands in Tessent Shell, using the set_attribute_options on setting, produces the module definition that follows:

 # set the attribute to always export
set_attribute_options -name speed -export_during_write on
set_attribute_options -name added_by -export_during_write on
report_icl_modules -module block

```
Module  block {
   DataInPort din;
   DataOutPort dout;
   Attribute speed = "200";
   Attribute added_by = "John";
}
```

Executing the following additional commands in Tessent Shell, using the set_attribute_options off setting, produces the module definition that follows:

 # set the attribute to always export
set_attribute_options -name speed -export_during_write off
set_attribute_options -name added_by -export_during_write off
report_icl_modules -module block

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module  block {
   DataInPort din;
   DataOutPort dout;
}
```

Example 3

The following example highlights the different usages of the set_attribute_options command and the set_attribute_value command:

register_attribute -name third_party_ip -value boolean -object_type module
set_attribute_options -name third_party_ip -preserve_boundary_in_flat_model on
set_attribute_value ram64 -name third_party_ip

The set_attribute_options command preserves the boundaries of all modules with the user-defined attribute, third_party_ip.

The set_attribute_value command preserves the boundaries of all instances of ram64 because the user-defined attribute third_party_ip was applied to the ram64 module.

Related Topics

get_attribute_list

get_attribute_option

get_attribute_value_list

register_attribute

report_attributes

reset_attribute_value

set_attribute_value

unregister_attribute

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_attribute_value

Context: unspecified, all contexts

Mode: all modes

Sets an attribute's value for the objects specified in ***object_spec***.

Usage

set_attribute_value ***object_spec*** **-name** ***attribute_name*** [-value *attribute_value*] [-silent]

Description

Sets an attribute's value for the objects specified in ***object_spec***. This command returns a collection of objects on which the attribute value has been set.

If the command fails to set the attribute on any element, an error message is given to the user and the command returns TCL_ERROR. This can be suppressed by using -silent (No error messages and no TCL_ERROR).

Table 8-2 lists some of the attributes that you can specify with the set_attribute_value command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-2. User-Specifiable Attributes

Name	Definition	Object	Values
default_fault_classification_0	String that provides an initial fault classification for the gate_pin on which it is set. This attribute corresponds to stuck-at-0 or slow-to-rise. You use the set_attribute_value command to set the desired default fault classification. During the flow, when the add_faults and read_faults commands are called, or when the tool internally adds faults for scan-based test pattern generation, the fault sites marked with this attribute are initialized with the specified default fault classification, instead of UC. This attribute only applies to the stuck fault model and transition fault model. It also applies to the UDFM fault model with no condition (for example, port faults) so that UDFM continues to work as a superset of stuck/transition. If this is set on a pin that is no-faulted or a non-fault site, it is ignored. This attribute is useful to identify gate_pins which are AU during scan-based test generation, but tested by some other test mode. You identify those faults using DI.<label>, where label identifies the test mode in which those faults are covered. For example, Tessent memory BIST uses the DI.mbisr value to identify the fault in the BISR register which are not testable by scan-based test generation, but covered in the memory BIST test. See Example 4 below for a typical use model	pin or gate_pin	Any pre-defined fault class and subclass (if fault class is AU or DI). Default value is UC.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-2. User-Specifiable Attributes (cont.)

Name	Definition	Object	Values
default_fault_classification_1	String that provides an initial fault classification for the gate_pin on which it is set.This attribute corresponds to stuck-at-1 or slow-to-fall. You use the set_attribute_value command to set the desired default fault classification. During the flow, when the add_faults and read_faults commands are called, or when the tool internally adds faults for scan-based test generation, the fault sites marked with this attribute are initialized with the specified default fault classification, instead of UC. This attribute only applies to the stuck fault model and transition fault model. It also applies to the UDFM fault model with no condition (for example, port faults) so that UDFM continues to work as a superset of stuck/transition. If this is set on a pin that is no-faulted or a non-fault site, it is ignored. This attribute is useful to identify gate_pins which are AU during scan-based test generation, but tested by some other test mode. You identify those faults using DI.<label>, where label identifies the test mode in which those faults are covered. For example, Tessent memory BIST uses the DI.mbisr value to identify the fault in the BISR register which are not testable by scan-based test generation, but covered in the memory BIST test. See Example 4 below for a typical use model	pin or gate_pin	Any pre-defined fault class and subclass(if fault class is AU or DI). Default value is UC.
ignore_for_graybox	Specifies a primary output/bidi pin to exclude from graybox analysis. By default, this attribute is set to false for all primary output/bidi pins, hence they are considered by the graybox analysis as backtracing origins. Setting this attribute on an output/bidi pin of a sub-module or an input pin of any module has no effect on graybox analysis. You must exclude the scan output pins of core chains (or channel output pins if it is a design containing EDT hardware) from graybox analysis using this attribute because the tool cannot automatically determine which output/bidi pins are for core chains. For more information about this attribute, refer to the analyze_graybox command description.	pin	true, false (default)
preserve_in_graybox	Specifies instances to include in the graybox netlist. For more information about this attribute, refer to the analyze_graybox command description.	instance, net	true, false (default)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-2. User-Specifiable Attributes (cont.)

Name	Definition	Object	Values
is_hard_module	When true, prevents a module from having any changes to its internal logic during test logic insertion, meaning that the module is written to the scanDEF file unexpanded (the scanDEF file references the hard macro's module instance rather than any flip-flops inside).	module	true, false (default)
variable_logic_infix	When the "%s" token is specified in "set_insert_test_logic_options -logic_infix", the tool embeds the attribute as a string in the instance names of the specified -logic_type.	scan element	string with no special character

Arguments

- ***object_spec***

  A required value that specifies a Tcl list of one or more object names, or a collection of one or more objects.

  Tip

  If your ***object_spec*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- **-name *attribute_name***

  A required switch and value pair that specifies the name of the attribute to set.

  When an initial fault classification for a pin is specified with the default_fault_classification_0 or default_fault_classification_1 attributes (for example, DI.USER), the fault is first analyzed through DI/UU/TI/BL fault analysis and is only added by its specified name (DI.USER) if it is not classified as a DI/UU/TI/BL fault.

- -value *attribute_value*

  An optional switch and value pair that specifies the value to which to set the attribute. You can omit this switch for Boolean attributes as shown in "Example 3" on page 2303.

- -silent

  An optional switch that suppresses error messages.

Examples

Example 1

The following example sets the is_hard_module attribute to true for all modules whose names start with mem_.

```
set_attribute_value [get_modules mem_*] -name is_hard_module
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{mem_a mem_b mem_c}
```

Example 2

This example sets the user-defined attribute clock_domain to clka on pin u1/u2/ck.

```
set_attribute_value u1/u2/ck -name clock_domain -value clka
```

```
{u1/u2/ck}
```

Example 3

The following example excludes clk1 and clk2 from X-constraining during core-level retargetable pattern generation of the CPU core.

```
SETUP> set_context patterns -scan
SETUP> set_current_mode -type internal
SETUP> read_verilog CPU.v
SETUP> read_cell_library atpglib
SETUP> set_current_design
SETUP> set_attribute_value {clk1 clk2} -name is_excluded_from_isolation_constraints
SETUP> set_system_mode analysis
ANALYSIS> write_core_description CPU.tcd -replace
ANALYSIS> create_patterns
ANALYSIS> write_patterns CPU_stuck.retpat.qz -replace
ANALYSIS> write_faults CPU_stuck.faults.qz -replace
...
```

Example 4

This example uses the trace_flat_model command to find all the fault sites in the combinational fanout of a port and initialize them as “DI.my_test_mode”. This port is not testable by scan-based test generation, but it is covered by another test mode called “my_test_mode”. Instead of ending up as AU, the fault sites are marked as DI.my_test_mode.

```
SETUP> set_context patterns
SETUP> read_cell_library atpgff.atpg_lib
SETUP> read_verilog design.v
SETUP> set_current_design
SETUP> create_flat_model
SETUP> set gate_pins [trace_flat_model \
                -from my_test_port \
                -direction forward \
                -controllability connected \
                -tag_condition "fault_site"]
SETUP> set_attribute_value $gate_pins -name default_fault_classification_0 \
                -value DI.my_test_mode
SETUP> set_attribute_value $gate_pins -name default_fault_classification_1 \
                -value DI.my_test_mode
```

Example 5

The following example preserves the boundary of an instance:

```
set_attribute_value [get_instance edt_i] -name preserve_boundary -value true
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

If you want to automatically preserve the boundary of design objects when you apply a user-defined attribute, without having to apply the "preserve_boundary" attribute, use the "set_attribute_options -preserve_boundary_in_flat_model on" command to configure your user-defined attribute after you register it.

Related Topics

get_attribute_list

get_attribute_option

get_attribute_value_list

register_attribute

reset_attribute_value

set_attribute_options

report_attributes

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_au_analysis

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the ATPG uses the ATPG untestable information to place ATPG untestable faults directly in the AU fault class.

Usage

set_au_analysis {**ON** | **OFf**}

Description

The set_au_analysis command specifies whether the ATPG process can use the ATPG untestable information. Upon invocation of the tool, the AU analysis is set to On; therefore, the ATPG process uses the ATPG untestable information. Once the tool places a fault in the AU fault class, it removes the fault from the active fault list and does not simulate it. This prevents the ATPG process from identifying the faults as possibly-detected during an ATPG run. However, you may use a switch on the compress_patterns command to identify possible detections of AU faults.

Arguments

- **ON**

  A literal that specifies for the tool to use the ATPG untestable information to place ATPG untestable faults directly in the AU fault class during any future ATPG processes. This is the invocation default.

- **OFf**

  A literal that specifies for the tool not to use the ATPG untestable information to place ATPG untestable faults directly in the AU fault class during any future ATPG processes.

Examples

The following example specifies not to use ATPG untestable information during the ATPG run:

```
set_system_mode analysis
set_au_analysis off
create_patterns
```

Related Topics

compress_patterns

delete_atpg_constraints

read_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_bidi_gating

Context: dft -scan, dft -test_points

Mode: setup

Specifies how bidirectional (bidi) pins are controlled during scan chain shifting to prevent potential bus contention or to ensure an on/off state during testing.

Usage

set_bidi_gating [**OFf** | **ON** | **Scan**] [-Control {SEn | TEn}] [-Direction Input | Output] [-Top {ALl | *primary_bidi_pin*...}] [-Force_gating]

Description

When enabled, test logic is inserted for bidi pins as necessary to control the enable signal and the input/output direction as specified.

By default, when the enable signal of a bidi pin is directly controlled by a primary input, by TIE0, or by TIE1, no gating is necessary, and a force statement for the primary input is added to the load_unload procedure in the new procedure file. This behavior can be overridden by using the -Force_gating switch.

You can also specify which enable signal (TEN or SEN) enables bidi pins.

Arguments

- **OFf | ON | Scan**

  Required literal that specifies whether to insert test logic to control the enable lines of bidi pins during scan chain shifting. For more information on scan chain shifting, see "Test Logic Insertion" in the *Tessent Scan and ATPG User's Manual*. Literal options include:

  OFf — no test logic is inserted to control bidi pins. Default setting.

  ON — test logic is inserted as necessary to control bidi pins.

  Scan — inserts test logic on the scan I/O bidi pins to control the direction of the bidi pin for scan shifting and ensure the success of scan chain tracing. Scan output bidi pins are gated to be in output mode, while all other bidi pins are gated to be in input mode (Z state on the tester).

- -Control SEn | TEn

  An optional switch and literal pair that specifies the enable signal used to control bidi pins. Options include:

  SEn — scan_enable signal. Default setting.

  TEn — test_enable signal.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Direction Input | Output

  An optional switch and literal pair that specifies the direction of the bidi pins specified by the -Top switch. Options include:

  Input — bidi pins are gated so they are in input mode. Default setting.

  Output — bidi pins are gated so they are in output mode.

- -Top ALl | *primary_bidi_pin*...

  An optional switch and literal or repeatable string pair that specifies which bidi pins are controlled with the specified enable signal. Options include:

  ALl — all bidi pins. Default setting.

  primary_bidi_pin — a specified primary bidi pin. Test logic is inserted to ensure that these bidi pins are controlled as specified.

- -Force_gating

  An optional switch that adds test logic to the enable lines of bidirectional pins that are directly controlled by primary inputs, by TIE1, or by TIE0. When the enable line is directly controlled by a primary input, the tool adds the force statement for this primary input to the load_unload procedure in the procedure file.

Examples

Example 1

The following example uses the set_bidi_gating command to insert test logic to control all bidi pins used for scan I/O via the SEN signal, and reports the gated bidi pin.

```
add_clocks 0 clk
set_tristate_gating on
set_bidi_gating scan
set_system_mode analysis
add_scan_mode unwrapped -si_connections c1 bidi_in1/X \
        -so_connections blkB1/blkA/utri2/A
analyze_scan_chains
```

Example 2

The following example uses the force_gating switch to insert gating logic controlled by the TEN control signal on the enable line of /uio1 and reports the gated tri-state devices. /uio1 is a bidirectional device driving primary inout port dinout[1]; its enable signal is directly controlled by the primary input /io_control1.

```
set_bidi_gating on -control ten -direction input -top dinout[1] -force_gating
```

Related Topics

report_test_logic

report_control_signals

set_test_logic

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_tristate_gating

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_bist_chain_test

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Disables all capture clock activity for the specified number of patterns in the fault simulation while running the chain test.

Usage

set_bist_chain_test [None | *number_of_patterns*]

Description

By default, the tool runs a 32-pattern chain test and disables all capture clock activity for those 32 patterns, which means that the unload values exactly match the load values during those patterns.

Hybrid TK/LBIST Flow

The tool automatically includes this command and literal in the generated logicBIST fault simulation dofile—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- None

  An optional literal that specifies that the tool should not disable the capture clock activity since there is no chain test being run.

- *number_of_patterns*

  An optional integer that specifies for the tool to disable all capture clock activity for the number of chain test patterns that are to be driven through the scan chains, which is 32 by default.

Related Topics

read_patterns

set_random_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_bist_debug

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Sets up a trace of LFSR values during a pattern's shift cycles.

Usage

set_bist_debug **-None** | [***LFSR_name***… {***pattern_number*** | **All**}]

Hybrid TK/LBIST Flow Usage

set_bist_debug **-None** | [***LFSR_name***… {***pattern_number*** | **All**}] |
[**-decompressor** [*block_name*] {***pattern_number*** | **All**}]

Description

To facilitate signature mismatch debugging and other diagnostics, the set_bist_debug command enables the tracing of all specified MISR and PRPG values associated with a given pattern, or all BIST patterns. If low-power hardware is enabled in the hybrid TK/LBIST flow, the low-power hold register value is traced along with its corresponding PRPG.

Arguments

- **-None**

  A switch that clears the list of PRPGs and MISRs the tool is currently monitoring.

- ***LFSR_name***

  A repeatable string that specifies the name of the MISRs and PRPGs whose values the tool traces. To list all MISRs and PRPGs in the design, use the report_lfsrs command.

  Of the next two arguments, you must choose one.

- ***pattern_number***

  An integer that specifies the pattern number for which the MISR or decompressor values are traced.

- **All**

  A literal that specifies to write out the value of the specified MISRs and PRPGs only at the end of the pattern, for every pattern.

Hybrid TK/LBIST Flow Arguments

- **-decompressor**

  A required switch that takes an optional *block_name* to specify that decompressor values for this block should be reported. (The decompressor is acting as a PRPG during logicBIST.) When using a single EDT module, the *block_name* is optional but is required in the modular EDT flow. If you do not specify *block_name*, the tool reports decompressor values for all blocks.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- *block_name*

  An optional string that specifies the name of the EDT block containing the decompressor to evaluate. The block_name must be one you defined previously using the add_edt_blocks command.

Examples

Example 1

The following example initiates a trace on the values of both the MISR and the PRPG for the pattern, 61, and shows the resulting transcript:

```
//  command: set_bist_debug misr prpg 61

//  command: simulate_patterns
//  ------------------------------------------------------------------
//  Simulation performed for #gates = 5732  #faults = 418
//  system mode = fault simulation    pattern source = 63 BIST patterns
//  ------------------------------------------------------------------
//  #patterns  test      #faults  #faults    # eff.    # test     process
//  simulated  coverage  in list  detected   patterns  patterns   CPU time
//  begin bist patterns: capture clock = /clk, observe point = MASTER
begin debug data for PRPGs pattern = 61
   PRPG         prpg    0   1000001111111010
   PRPG         prpg    1   0100000111111101
   PRPG         prpg    2   1001010011111110
   PRPG         prpg    3   0100101001111111
   .
   .
   .
   PRPG         prpg   14   0101011110010010
   PRPG         prpg   15   0010101111001001
   PRPG         prpg   16   1010000111100100
//   63         88.68%      387      31      5      0      0.04 sec
begin debug data for MISRs pattern = 61
   MISR         misr    1   1110110100100111
   MISR         misr    2   0111011010111110
   MISR         misr    3   0011101100111111
   MISR         misr    4   0001110111111111
   .
   .
   .
   MISR         misr   16   1100100000101111
   MISR         misr   17   1100100000101111
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example initiates a trace on the values of the PRPG and MISR for all patterns:

```
//  command: set_bist_debug misr prpg all

//  command: simulate_patterns
//  ---------------------------------------------------------------------
//  Simulation performed for #gates = 5912  #faults = 513
//  system mode = fault simulation    pattern source = 63 BIST patterns
//  ---------------------------------------------------------------------
//  #patterns  test      #faults  #faults    # eff.    # test     process
//  simulated  coverage  in list  detected   patterns  patterns   CPU time
//  begin bist patterns: capture clock = /clk, observe point = MASTER
   PRPG         prpg    0   0001001000001001
   PRPG         prpg    1   0011001000011010
   PRPG         prpg    2   0010111111001111
   PRPG         prpg    3   0110011101111011
   .
   .
   .
   PRPG         prpg   29   0001101111101000
   PRPG         prpg   30   0100010010110000
   PRPG         prpg   31   1110101100101111
   MISR         misr    0   0101010100001101
   MISR         misr    1   1101011101111111
   MISR         misr    2   1101010001101001
   MISR         misr    3   1110000010111001
   .
   .
   .
   MISR         misr   29   0010001111001110
   MISR         misr   30   1111000010000110
   MISR         misr   31   1111001001110101
   PRPG         prpg   32   1000110000011000
   PRPG         prpg   33   0101010011111101
   PRPG         prpg   34   0100001010111010
   .
   .
   .
   PRPG         prpg   60   1000001111111010
   PRPG         prpg   61   1010000111100100
   PRPG         prpg   62   0000000110101110
//   63         86.05%      488      25      3      0      0.10 sec
   MISR         misr   32   1000101100100001
   MISR         misr   33   0010101011010011
   MISR         misr   34   1001100110111101
   .
   .
   .
   MISR         misr   60   1011110010101001
   MISR         misr   61   0011100101011010
   MISR         misr   62   0001111111111100
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

The following example initiates the trace on the values of the decompressor, which is used as a PRPG in the hybrid TK/LBIST flow, for pattern 4 and shows the resulting transcript. In this example, low power is enabled.

```
// command: set bist debug -decompressor 4

begin debug data for DECOMPRESSORs pattern = 4
  DECOMP  piccpu_edt_i.lfsm_vec   0  10011100100110010101001100100 10
  LP      piccpu_edt_i.lfsm_vec   0  01001100000001100100100011001 10
  DECOMP  piccpu_edt_i.lfsm_vec   1  01001110010010001010100110010 01
  LP      piccpu_edt_i.lfsm_vec   1  01001100000001100100100011001 10
  DECOMP  piccpu_edt_i.lfsm_vec   2  10100111011001000101010011001 00
  LP      piccpu_edt_i.lfsm_vec   2  01001100000001100100100011001 10
  DECOMP  piccpu_edt_i.lfsm_vec   3  01010111101101100010101001100 10
  LP      piccpu_edt_i.lfsm_vec   3  01001100000001100100100011001 10
  DECOMP  piccpu_edt_i.lfsm_vec   4  00101111100110110001010100110 01
  LP      piccpu_edt_i.lfsm_vec   4  01001100000001100100100011001 10
  DECOMP  piccpu_edt_i.lfsm_vec   5  10010111110010011000101010011 00
  LP      piccpu_edt_i.lfsm_vec   5  01001100000001100100100011001 10
  DECOMP  piccpu_edt_i.lfsm_vec   6  01001011101001001100010101001 10
  LP      piccpu_edt_i.lfsm_vec   6  01001100000001100100100011001 10
.
.
.
```

Related Topics

add_lfsrs

report_lfsrs

set_bist_trace

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_bist_trace

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables the tracing of PRPG and MISR values associated with each BIST pattern to facilitate signature mismatch debugging and other diagnostics.

Usage

set_bist_trace [-Nolfsr | {-Lfsr *filename* [-Replace] [-Interval *interval*] [-Offset *offset*]}]

Hybrid TK/LBIST Flow Usage

set_bist_trace [-prpg_data_file *filename* -misr_data_file *filename*]
{[-prpg_pload_file *filename* -misr_pload_file *filename*] [-range *beg_int end_int]}*
[-low_power] [-replace]

Description

The traced values are logged in *filename* in the following format:

```
0 PRPG prpg 1111111111111111
0 MISR misr 11111111111111111111
1 PRPG prpg 1001010001011000
2 PRPG prpg 0101111001100101
…
32 PRPG prpg 1100100000000110
1 MISR misr 11100111011000010110
2 MISR misr 11001110011100001110
3 MISR misr 01011001100111011001
…
32 MISR misr 01011001100111011001
32 PRPG prpg 1100100000000110
…
```

The first element on each line is the pattern number, followed by the simulated element type (MISR or PRPG). The third element is the name of the simulated element. The names shown in the previous example are the default names assigned by the add_lfsrs commands included in the BIST-Controller-phase-generated fault simulation driver. You can edit the driver file to rename the elements.

Note

Do not modify or delete any header information in *filename*. Microcoded testbench instructions are generated based on information in this header.

The sequence of activities that occur within the simulation of the BIST controller determines the order of the lines in *filename*.

If you issue multiple set_bist_trace commands before the simulate_patterns command, only the last one issued is executed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

To reduce the volume of data written to *filename*, use the -Interval and -Offset switches. See the Example 1 section for this command.

Arguments

- -Nolfsr

  An optional switch that disables LFSR value tracing. This is the default.

- -Lfsr *filename*

  An optional switch and string pair that enables tracing of PRPG and MISR values during a BIST run and writes those values to the *filename* file.

- -Replace

  An optional switch that replaces the contents of *filename* if that file already exists.

- -Interval *interval*

  An optional switch and integer pair that writes the PRPG and MISR values to *filename* for the patterns beginning at 0 and separated by the interval given by the *interval* integer. The values for the first and last patterns are always written.

- -Offset *offset*

  An optional switch and integer pair that writes the PRPG and MISR values to *filename* beginning at the pattern number given by the *offset* integer. The values for the first and last patterns are always written.

Hybrid TK/LBIST Flow Arguments

- -prpg_data_file *filename* -misr_data_file *filename*

  Two optional switches and string pairs that enable tracing of the PRPG and MISR values for all patterns during a BIST run and write those values to the respective *filename* file. The files are written in a format compatible with the LV Flow.

- -prpg_pload_file *filename* -misr_pload_file *filename*

  Two optional switches and string pairs that enable tracing for a few important clock cycles of the PRPG and MISR values during a BIST run and write those values to the respective *filename* file. The files are written in a format compatible with the LV Flow and are used for parallel load simulations.

- -range *beg_int end_int*

  An optional switch and integer pair that defines a range of patterns to be written out to the PRPG/MISR files when using the -prpg_pload_file and -misr_pload_file options. By default, the full pattern range for a BIST run is written out.

- -low_power

  An optional switch that defines whether the hybrid TK/LBIST flow should have the low-power values written out to the PRPG file when using the -prpg_data_file and -prpg_pload_file options.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -replace

  An optional switch that replaces the contents of *filename* if that file already exists.

Examples

Example 1

The following example writes traced values to the *lfsr.trace* file, but only writes the values for every 256th pattern to conserve disk space.

```
set_bist_trace -lfsr lfsr.trace -replace -interval 256
```

In the preceding example, if you discover a problem begins after pattern 1530, you might narrow the search by using the following command:

```
set_bist_trace -lfsr lfsr.trace -replace -offset 1530 -interval 16
```

and then finish by using the following command:

```
set_bist_trace -lfsr lfsr.trace -replace -offset 1594
```

Example 2 (Hybrid TK/LBIST Flow)

The following example writes out PRPG and MISR data files, with low-power values:

```
set_bist_trace -prpg_data_file LV_WORKDIR/TOP.prpg_data_lbist \
    -misr_data_file LV_WORKDIR/TOP.misr_data_lbist \
    -low_power \
    -replace
```

Example 3 (Hybrid TK/LBIST Flow)

The following example writes out the PRPG and MISR parallel load data files, with low-power values, for the first 256 patterns:

```
set_bist_trace -prpg_pload_file LV_WORKDIR/TOP.prpg_pload_lbist \
    -misr_pload_file LV_WORKDIR/TOP.misr_pload_lbist \
    -low_power \
    -range 0 255 \
    -replace
```

Related Topics

set_bist_debug

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_boundary_scan_port_options

Context: dft

Mode: setup

Sets options relative to boundary scan on ports of the design.

Usage

set_boundary_scan_port_options ***port_spec*** { -cell_options *cell_option* | -packed_pin_name *port_name_list* } |
-pin_order_file *filename* | -pad_io_ports *pad_io_ports*

Description

The -pin_order_file and -pad_io_ports options apply to all ports.

When the set_dft_specification_requirements -boundary_scan option is set to on, the options defined with this command are considered by the check_design_rules and create_dft_specification commands.

Arguments

- *port_spec*

  An optional string that specifies a Tcl list of names of one or more ports, or a collection containing one or more ports of the current design.

- -cell_options *cell_option*

  An optional literal pair that specifies an option to associate with the specified ports. The permitted values are:

```
normal | no_bscan_cell | dont_touch | no_connect | analog |
compliance_enable0 | compliance_enable1 |
{input_only |output_only} | no_capture_core_signal  | clock |
sample_only | add_dot6_from_pad_cell | {combined_bidir_cells |
separated_bidir_cells}
```

  See Table 13-5 on page 4105 for a description of these values.

Note

The input_only option works in conjunction with the clock or sample_only option if you specify a list. For example: “{input_only sample_only}” or “[input_only sample_only]”.

The output_only option works in conjunction with the no_capture_core_signal option if you specify a list. For example: “{output_only no_capture_core_signal}” or “[output_only no_capture_core_signal]”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -packed_pin_name *port_name_list*

  An optional literal pair that specifies the pin names for the package pin mapping. The length of port_name_list must match the length of port_spec.

- -pin_order_file *filename*

  An optional value pair that is used to reference an existing pin order file. When this option is used, the specified pin_order_file is validated during the check_design_rules command instead of being created. If you create this file from a different tool or you re-order the one that was created from a previous run of Tessent Shell, you must point to your file using this option to have it used. You can avoid having to manually reorder this file by reading in a DEF file using the read_def command. The pin order filename is stored in the DftSpecification/BoundaryScan/pin_order_file property when create_dft_specification is run.

- -pad_io_ports pad_io_ports

  An optional value pair used to specify the ports on the current design that are to be equipped with boundary scan cells. This option is only used when the design level is physical_block or sub_block as specified using the set_design_level command. The pad_io_ports string is a Tcl list of names of one or more ports, or a collection containing one or more ports of the current design.

Examples

The following example uses the -cell_options option to set the sample_only option on a set of ports. Notice that if you have power and ground ports in your design and the design level is top, as specified by the set_design_level command, you must set the function attribute on them to power and ground as shown here.

```
> set_boundary_scan_port_options [get_ports HIO*] \
          -cell_options sample_only
> set_attribute_value [get_ports vdd*] -name function -value power
> set_attribute_value [get_ports vss*] -name function -value ground
```

Related Topics

get_boundary_scan_port_option

report_boundary_scan_port_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_bsdl_extraction_options

Context: dft (with no subcontext)

Mode: setup

Sets values for various BSDL attributes in the BSDL file extracted using the BSDL extraction flow.

Usage

set_bsdl_extraction_options
[-pulse_min_duration *time*]
[-train_minimum_count *int*]
[-train_max_duration *time*]
[-manufacturer_id_code *id*]
[-part_number_code *partnum*]
[-version_code *version*]
[-bonding_configuration_enables *port_pin_spec*]

Description

Sets values for pulse duration, train tests, and the device ID register for the extracted BSDL. See "AIO_EXTEST_Pulse_Execution" and "AIO_EXTEST_Train_Execution" in the IEEE 1149.6 standard and "Device Identification Register" in the IEEE 1149.1 standard.

You can also use this command to specify additional bonding configurations for the extracted BSDL file that are not present in any tcd_bscan description of the embedded BScan block. This is useful to specify additional bonding configuration enables that select a different Device ID value.

Note

The -bonding_configuration_enables option does not support BSDL extraction in 2.5D designs. Specify these values in the die BSDL file.

Arguments

- -pulse_min_duration *time*

  An optional switch and value pair that defines the minimum time (in seconds) to use between the rising and the falling edge during the AC tests. This value is included in the BSDL file using the AIO_EXTEST_Pulse_Execution attribute.

- -train_minimum_count *int*

  An optional switch and value pair that defines the minimum number of pulses for the train test. This value is included in the BSDL file using the AIO_EXTEST_Train_Execution attribute.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -train_max_duration *time*

  An optional switch and value pair that defines the maximum time for the train test. This value is included in the BSDL file using the AIO_EXTEST_Train_Execution attribute.

- -manufacturer_id_code *id*

  An optional switch and binary pair to specify the manufacturer ID code section of the 32-bit device ID. *id* must be eleven bits.

- -part_number_code *partnum*

  An optional switch and binary pair to specify the part number code section of the 32-bit device ID. *partnum* must be sixteen bits.

- -version_code *version*

  An optional switch and binary pair to specify the version code section of the 32-bit device ID. *version* must be four bits.

- -bonding_configuration_enables *port_pin_spec*

  An optional switch and string pair to specify the bonding configuration enable pins/ports. The *port_pin_spec* argument can be a Tcl list of one or more pins/ports or a collection of one or more pins/ports.

  Restriction

  The enable signals you use to change the device ID must be static signals.

Related Topics

delete_bsdl_extraction_option

get_bsdl_extraction_option

report_bsdl_extraction_options

Prerequisites for BSDL Extraction

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_burnin_options

Context: patterns -scan, patterns -scan_retargeting

Mode: analysis

Configures the burn-in pattern output.

Usage

set_burnin_options [-ssh_icl_instances *ssh_icl_instances*] [-sequence *sequence*] [-repetitions {load_chains | *int*}] [-payload_loop_count {off | *int*}]

Description

The set_burnin_options command performs optional configuration for the burn-in pattern.

Arguments

- -ssh_icl_instances *ssh_icl_instances*

  An optional switch and string pair that specifies the SSH ICL instances to target with the burn-in pattern. The default is to target “all” patterns.

- -sequence *sequence*

  An optional switch and string pair that specifies the binary sequence to use for the burn-in payload for all active SSHs. By default, the tool uses “1100”.

  For example, with the “1100” sequence, the first two packets drive the bus bits to “1” and the last two packets drive the bus bits to “0”.

- -repetitions {load_chains | *int*}

  An optional switch and literal or integer pair that specifies the number of times to repeat the payload. The default number of repetitions is 1. When you use the “load_chains” value, the tool automatically calculates a number of repetitions that is sufficient to generate activity on all EDTs. This is useful to observe design activity during simulation.

  Note

  The “load_chains” option generates enough repetitions to observe at least some activity in all scan chains. However, the number of repetitions generated may not necessarily fill up the entire scan chain. If you need more simulation cycles, manually specify a larger number of repetitions.

- -payload_loop_count {off | *int*}

  An optional switch and literal or integer pair that controls whether the burn-in payload is in a loop with the specified count. The default value is “off”, which does not use an explicit loop around the payload; instead, the burn-in payload loops in infinite shift mode without an ssn_end procedure. This option is specifically intended for patterns that are intended to be run on the tester.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example shows a snippet from a dofile that uses the default burn-in sequence (and declares it explicitly) and changes the number of repetitions, first for a verification pattern and then for a regular burn-in pattern:

```
…
check_design_rules

set_burnin_options -sequence 1100
set_burnin_options -repetition load_chains
write_patterns kekelo_burnin_sequence_1100_load_chains.v \
     -verilog -pattern_sets burnin_loop -replace -parameter_list {}
write_patterns kekelo_burnin_sequence_1100_load_chains.stil \
     -stil    -pattern_sets burnin_loop -replace -parameter_list {}

set_burnin_options -repetition 3
write_patterns kekelo_burnin_repetitions_3.v \
     -verilog -pattern_sets burnin_loop -replace -parameter_list {}
write_patterns kekelo_burnin_repetitions_3.stil \
     -stil    -pattern_sets burnin_loop -replace -parameter_list {}
```

Related Topics

get_burnin_option

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_bus_handling

Context: dft -edt, patterns -scan

Mode: analysis

Specifies the bus contention results that you desire for the identified buses.

Usage

set_bus_handling {**Pass** | **Fail** | **Abort**} {**-All** | **-Pass** | **-Fail** | **-Bidi** | **-ABort** | [-Instances *instance_name*...] | ***bus_gate_id#***... | ***net_pathname...***}

Description

The set_bus_handling command preassigns the contention check handling result that you desire during simulation for the buses that you specify. Upon invocation, the tool automatically calculates the bus contention handling as documented under the set_contention_check command description. The tool rejects (from the internal test pattern set) ATPG-generated patterns that can cause bus contention. The set_bus_handling command lets you override the automatic contention calculations, thereby changing whether or not the tool performs a simulator-based check using such a pattern.

The tool resets the bus contention handling back to the automatically calculated value whenever you make a change to the modeling that requires the tool to perform a complete re-analysis of the contention mutual exclusivity (such as changing the net resolution or the pin constraints).

Caution

Overriding the automatically calculated contention check handling results can cause trouble downstream if there is a problem with the design that required modification due to the bus contention.

Arguments

- **Pass**

  A literal that specifies for the tool to not perform bus contention evaluations on the buses that you identify and to treat them as if they had passed. This enables the tool to retain patterns that it would otherwise reject due to a contention check failure.

- **Fail**

  A literal that specifies for the tool to treat the buses that you identify as if they had failed the bus contention evaluations. This causes the tool to reject patterns that it would otherwise retain due to passing the contention check.

- **Abort**

  A literal that specifies the bus aborted the bus contention evaluations before determining whether the bus passed or failed. This can be used with the analyze_bus -Drc_check command to verify ATPG constraints which you have added to correct bus failures.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-All**

  A switch that specifies for the tool to change all buses to a specified state reset.

- **-Pass**

  A switch that specifies for the tool to change to specified state for the buses that have previously passed the E10 DRC rule.

- **-Fail**

  A switch that specifies for the tool to change to specified state for the buses that have previously failed the E10 DRC rule.

- **-Bidi**

  A switch that specifies for the tool to change to specified state for bidi buses identified by the E10 DRC rule.

- **-ABort**

  A switch that specifies for the tool to change to specified state for the buses that have previously aborted the E10 DRC rule.

- -Instances *instance_name*

  An optional switch and repeatable string that specifies the instances where the included buses are to be changed to the specified state.

- ***bus_gate_id#***

  A repeatable integer that specifies the gate identification numbers of the buses whose contention handling you want to override. If a bus is cascading, you must specify the dominant bus.

- ***net_pathname***

  A repeatable string that specifies a net name of the buses to be changed to a specified state.

Examples

The following example turns the bus contention checking off, enabling the bus to pass the evaluations. However, this action can cause trouble in the future if there is a problem with the design that required modification due to the bus contention.

report_bus_data 321

```
/FA1/ha1/XOR1/OUT/ (321) handling=fail type=strong #Drivers=4
  Learn Data:poss_X=yes, poss_Z=yes, poss_mult_drivers_on=yes
  BUS Drivers: 156(SW) 252(SW) 307(SW) 308(SW)
```

set_bus_handling pass 321
report_bus_data 321

```
/FA1/Ha1/Xor1/OUT/ (321) handling=pass type=strong #Drivers=4
  Learn Data:poss_X=yes, poss_Z=yes, poss_mult_drivers_on=yes
  BUS Drivers: 156(SW) 252SW) 307(SW) 308(SW)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

report_boundary_scan_port_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_bus_simulation

Context: dft -edt, patterns -scan

Mode: analysis

Specifies whether the tool uses global or local bus simulation analysis.

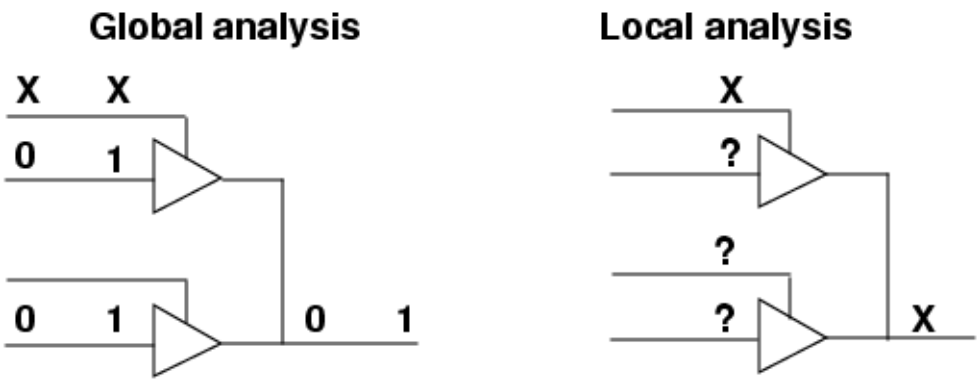
Usage

set_bus_simulation [Global | Local]

Description

This command simplifies typical back end verification flows by providing the option to turn off global analysis and use only the values that can be determined by inspecting the immediate input gates (tri-state devices or TSDs) driving a bus. Figure 8-1 shows an example of how the setting of this command affects pattern creation.

Figure 8-1. Global Versus Local Bus Simulation Analysis Example



Global analysis assumes a consensus value (0 or 1) is driven onto the bus when one or more of the tri-state enables is X, and all inputs to the tri-states are the same (either all 0 or all 1). This most closely matches real hardware behavior. Local analysis assumes an X is driven onto the bus if any of the tri-state enables is X, regardless of the values on the inputs to the tri-states. It is more pessimistic but more closely matches typical Verilog UDP behavior.

This command is provided to resolve compatibility issues with tools that do not perform global analysis. For example, if your design includes instances of TSDs for gating signals onto a bus, and their behavior in a timing-based simulator is as shown on the right in the preceding figure, using global analysis for pattern creation can result in simulation mismatches when you perform timing-based verification of the test patterns. You can avoid these mismatches by using this command to specify local bus simulation analysis for pattern creation.

Tip

The report_environment command displays the analysis type currently in effect.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- Global

  Specifies for the tool to perform global bus simulation analysis. This is the default.

- Local

  Specifies for the tool to perform local bus simulation analysis.

Related Topics

report_environment

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_bypass_chains

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup

Specifies a list of scan chains to connect in series to form a particular EDT bypass chain.

Usage

set_bypass_chains {**-Bypass_chain_number** ***chain_number***}
[-Edt_chains {*chain_name*…}] [-Pins *bypass_input bypass_output*]

Description

The set_bypass_chains command enables customization of the configuration of the bypass chains; for example, to meet unique routing requirements.

If you use this command to customize the bypass chains, you must specify all the EDT bypass chains.

Arguments

- **-Bypass_chain_number** ***chain_number***

  A required switch and positive integer pair that identifies the EDT channel to connect to the specified scan chains to form an EDT bypass chain.

  Note

  When you specify the number of channels using the set_edt_options command -Channels switch, the channels are numbered consecutively starting at 1.

- -Edt_chains {chain_name…}

  An optional switch and repeatable string pair that specifies the internal scan chains to connect in series to form the specified bypass chain. The chains are connected from channel input to channel output in the order specified.

- -Pins *bypass_input bypass_output*

  An optional switch and double string that specifies the input and output pins for bypass chains that exist in the design netlist before EDT logic is generated.

  Note

  Scan chains and bypass chains must use the same input and output pins.

Examples

Example 1

The following example sets up the EDT logic for a design with eight scan chains named chain1 through chain8. The design is setup with two EDT channels and chains 1, 5 and 6 are connected in series to channel 1 to form bypass chain 1 and the results display. Any number in the Bypass# column with no connections next to it represents an undefined bypass chain.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_edt_options -channels 2
set_bypass_chains -bypass_chain 1 -edt_chains chain1 chain5 chain6
report_bypass_chains

```
//    Reporting user-defined bypass chains:
//
//    Bypass#  EDT chains
//    -------  ----------
//    1        chain1, chain5, chain6
//    2        no connections
//
```

The following commands define the remaining bypass chain and displays the completed user-defined bypass configuration.

set_bypass_chains -bypass_chain 2 -edt_chains chain2 chain3 chain4 chain7 chain8
report_bypass_chains

```
//    Reporting user-defined bypass chains:
//
//    Bypass#  EDT chains
//    -------  ------
//    1        chain1, chain5, chain6
//    2        chain2, chain3, chain4, chain7, chain8
//
```

Example 2

The following example shows a dofile segment that sets up the scan chains and bypass chains for EDT logic and then shows the resulting EDT logic file.

```
// Internal scan chains
add_scan_chains chain1 grp1 scan_in1 scan_out1
add_scan_chains  chain2 grp1 scan_in2 scan_out2
add_scan_chains  chain3 grp1 scan_in3 scan_out3
add_scan_chains  chain4 grp1 scan_in4 scan_out4
//
// Bypass scan chains
set_edt_options -channels 2 -bypass use_existing_bypass_chains
set_bypass_chains -bypass_chain 1 -pins scan_in1 scan_out1 -edt_chains \
chain1 chain2
set_bypass_chains -bypass_chain 2 -pins scan_in2 scan_out2 -edt_chains \
chain3 chain4
```

Muxes are added for only the internal scan pins that are also bypass chain pins as shown in the following EDT logic file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
module TOP_edt_bypass_logic (edt_bypass, edt_channels_in, edt_channels_out,
                             edt_scan_in, edt_scan_out, edt_bypass_in, edt_bypass_out);
   input        edt_bypass;
   input  [1:0] edt_channels_in;
   output [1:0] edt_channels_out;
   output [3:0] edt_scan_in;
   input  [3:0] edt_scan_out;
   input  [3:0] edt_bypass_in;
   input  [1:0] edt_bypass_out;


   assign edt_scan_in[0] = edt_bypass ? edt_channels_in[0] : edt_bypass_in[0];
   assign edt_scan_in[1] = edt_bypass ? edt_channels_in[1] : edt_bypass_in[1];
   assign edt_scan_in[2] = edt_bypass_in[2];
   assign edt_scan_in[3] = edt_bypass_in[3];

   assign edt_channels_out[0] = edt_bypass ? edt_scan_out[0] : edt_bypass_out[0];
   assign edt_channels_out[1] = edt_bypass ? edt_scan_out[1] : edt_bypass_out[1];
endmodule
```

Related Topics

report_bypass_chains

reset_bypass_chains

set_edt_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_capture_clock

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the capture clock for random pattern simulation or, optionally, for all ATPG pattern simulation.

Usage

set_capture_clock [*primary_input_pin*] [-Atpg [-ALlow_multiple_cycles]] | ANY]

Description

This command affects the capture clock that is used during DRC simulation and may be useful for designs with certain types of test logic that must be initialized properly to pass scan chain tracing.

The set_capture_clock command specifies the name of the capture clock the tool uses during random pattern simulation. You can specify the name of either a specific pin or a clock procedure in a test procedure file that identifies the pin. In either case, the pin must be a currently defined clock pin. Also, the capture clock that you specify cannot have a pin constraint.

If you do not specify a capture clock with this command, the tool sets the capture clock to none. If there is no specified capture clock and there is only one clock in the circuit that is not a set or reset line, the tool sets that clock as the capture clock during rules checking and displays a warning message that identifies the capture clock.

When you add a capture clock, the tool checks for pulse-always or pulse-in-capture clocks. If any exist, the tool does not create the capture clock and issues an error. For example:

```
add_clocks 0 CLK1 -pulse_in_capture
add_clocks 0 CLK2
set_capture_clock CLK2 -atpg
```

```
//  Error: Clock CLK2 cannot be defined as capture clock in the presence
//  of pulse-in-capture clock CLK1.
```

When you add a pulse-always or pulse-in-capture clock, the tool checks for a user-defined capture clock, a warning message is issued, and the user-defined capture clock is reset to none. For example:

```
add_clocks 0 CLK2
set_capture_clock CLK2 -atpg
add_clocks 0 CLK1 -pulse_always
```

```
//  Warning: set_capture_clock is reset as consequence of adding a pulse-
in-capture clock.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The set_clock_options command's -pulse_always and -pulse_in_capture switches behave the same way as add_clocks. The warning or error is reported and the capture clock resets when a user-defined capture clock exists.

These two checks only report when you define synchronous pulse-always or pulse-in-capture clocks with a capture clock specified. You can define an asynchronous free running clock in the presence of a capture clock

You can use the report_environment command to list the capture clock and the report_clocks command to identify the current list of clocks.

Note

This command applies only to random pattern generation unless you include the -Atpg switch; then it applies to all scan patterns created during the ATPG process.

The set_capture_clock command specifies a particular clock as the capture clock during random pattern generation or, optionally, ATPG. For random pattern generation alone, the command takes effect only if you previously issued a "simulate_patterns -source random" command.

When you have specified a capture clock with this command, the tool may pulse other defined clocks, but it does not use them for capture. You can specify the capture clock by providing the name of a specific pin in a test procedure file that identifies a currently defined clock pin. The capture clock that you specify cannot have a pin constraint.

You can undo a previously specified capture clock choice by issuing this command without any arguments. This makes all clocks available for clocking and capture.

Note

During ATPG, the capture clock specified with this command is ignored if one or more named capture procedures are available and their use is enabled with "set_capture_procedure on" (the default).

Arguments

- *primary_input_pin*

  An optional string that specifies the name of the primary input pin you want to assign as the capture clock.

- -Atpg

  An optional switch that directs the tool to use the specified capture clock as the only capture clock for all patterns created during the ATPG process, not just random patterns. This option also forces the tool to pulse the capture clock only once per capture cycle per pattern, unless you also use the -allow_multiple_cycles switch, causing the tool to generate combinational and clock sequential patterns (no RAM sequential or chain test patterns). If specified in Setup mode, this switch turns on some additional design rule checks (DRCs).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When you use this switch in Setup mode, all named capture procedures (NCPs) are disabled by default when entering a non-Setup mode. In addition, the create_patterns command can then create only patterns with the specified capture clock. After the tool enters a non-Setup mode, you can manually enable NCPs by using the "set_capture_procedure On" command. The tool then checks to ensure that each of the NCPs pulse the capture clock once and only once. The tool does not enable those NCPs that do not satisfy this clock sequence.

Note

When using Tessent TestKompress (EDT On), you must specify this switch to enable the command to work properly with EDT.

- -ALlow_multiple_cycles

  An optional switch that enables the tool to create patterns with multiple pulses of the capture clock. Using this switch disables the DRC stability check that uses the capture clock, thus enabling ATPG to create patterns with multiple clock pulses of the specified capture clock. This switch is available only when you also use the -atpg switch.

- ANY

  An optional string that specifies any clock can be a capture clock. This is the default if *primary_input_pin* is not specified.

Examples

The following example specifies a capture clock:

```
add_clocks 1 clock1
set_capture_clock clock1
set_system_mode analysis
set_random_patterns 612
```

Related Topics

add_clocks

delete_clocks

read_patterns

report_clocks

report_environment

set_capture_procedures

simulate_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_capture_procedures

Context: dft -edt, patterns -scan

Mode: analysis

Enables or disables the tool's use of a specified subset of named capture procedures when generating test patterns.

Usage

set_capture_procedures {ON | OFf} {-All | -Created_capture_procedure | -Loaded_capture_procedure | [*procedure_name*…]}

Description

By default, the tool uses all named capture procedures defined in the test procedure file, except any that fail to pass DRC. The tool never uses capture procedures that fail DRC.

This command is useful when you want to use a subset of the defined capture procedures for certain types of faults because the remaining capture procedures may be unable to detect the faults.

Note

The set_capture_procedures command applies to test generation only, not to fault simulation. If you generate test patterns using "set_capture_procedure off", for example, fault simulation still simulates those test patterns even if you issue a "set_capture_procedure on" command prior to fault simulation.

Arguments

- ON

  A literal that enables use of the specified named capture procedures during test generation. This is the default. If one or more named capture procedures is enabled with this option, the tool does not use the default capture procedure; it uses the named capture procedure(s) exclusively. In this case, if the conditions to test a fault require a particular sequence of clocks or cycles that is not defined in one of the named capture procedures, the fault is untestable.

  Note

  As shown in the examples, the use of the "on" option to specify a subset of the defined capture procedures must be preceded by "set_capture_procedure off -all".

- OFf

  A literal that disables the use of the specified named capture procedures during test generation. If you use this option with "-all" to disable all named capture procedures, ATPG uses the default capture procedure, if present. Otherwise, it runs as if there are no capture procedures defined in the test procedure file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Use of the "on" option to specify a subset of the defined capture procedures must be preceded by "set_capture_procedure off -all".

- -All

  A switch that enables the use of all capture procedures defined in the test procedure file. This is the default.

- -Created_capture_procedure

  A switch that enables the use of only the named capture procedures internally created from the create_capture_procedures command.

- -Loaded_capture_procedure

  A switch that enables the use of only the named capture procedures loaded from the read_procfile command or the procedure file associated with the scan group definition.

- *procedure_name...*

  A repeatable string that enables the use of only the named capture procedures with the user-specified name.

Examples

Example 1

The following example loads a fault list, disables the use of all capture procedures, enables just the capture procedure named c1_capture, and then creates patterns:

```
read_faults clock1_fault_list
set_capture_procedures off -all
set_capture_procedures on c1_capture
create_patterns
```

Example 2

The following example creates named capture procedures from generated patterns, and then disables use of loaded capture procedures:

```
create_patterns
create_capture_procedures -pattern internal
set_capture_procedures off -loaded_capture_procedure
```

Related Topics

create_capture_procedures

delete_capture_procedures

report_capture_procedures

set_drc_handling

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_capture_timing_options

Context: all contexts

Mode: all modes

Sets the capture_clock period and other timing aspects of the capture procedure during SSN scan operations for the current physical block.

Usage

set_capture_timing_options
-mode_type {**internal** | **external**}
[-capture_clock_period *time*]
[-capture_clock_edge1_percentage *int*]
[-capture_clock_edge2_percentage *int*]
[-capture_clock_high_pulse_width *same_as_shift_clock* or *1..99*]
[-pi_setup_extra_cycles *int*]
[-pi_hold_extra_cycles *int*]
[-pi_force_percentage *int*]
[-po_measure_percentage *int*]
[-min_capture_window_in_ssn_bus_cycles auto | *int*]
[-generated_timeplate_name *timeplate_name*]

Description

The set_capture_option command controls timing aspects of the capture procedure during SSN scan operations. The scope of this command is limited to the current physical block, as set by the set_current_physical_block command. By default, the current physical block includes the entire design.

The reference page section of the set_load_unload_timing_options command describes how the ScanHost is designed with fully programmable timing. The capture clock is generated by the ScanHost node when using internal capture (see the set_current_mode command), or for retargeted scan patterns. The default spacing of the capture clock pulses is large enough to enable staggered capturing during low-speed stuck-at ATPG. Figure 8-9 in set_load_unload_timing_options shows the timing checks affected by the settings of this command.

Note

Only the capture clock period and the capture_clock_high_pulse_width are relevant for internal capture usage.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- **-mode_type** {**internal** | **external**}

  A required switch and literal that specifies the capture mode. Choose one of the following:

  internal — The PIs and POs are isolated (that is, the physical block is wrapped) and the capture clock is generated by the ScanHost node

  external — The PIs and POs on the physical block are connected to the top-level. The TCK clock is used to generate the slow capture clock pulses.

- -capture_clock_period *time*

  An optional switch and value pair that specifies the period of the capture clock. The default values are as follows:

  internal mode — 40ns.

  external mode — 40ns.

  Choose from the following time units:

  s — seconds.

  ms — milliseconds.

  us — microseconds.

  ns — nanoseconds. This is the default time unit.

  ps — picoseconds.

  fs — femtoseconds.

- -capture_clock_edge1_percentage *int* — **external mode only**

  An optional switch and value pair that specifies the edge1 percentage of the capture clock. The default value is 50 percent.

- -capture_clock_edge2_percentage *int* — **external mode only**

  An optional switch and value pair that specifies the edge2 percentage of the capture clock. The default value is 99 percent.

- -capture_clock_high_pulse_width {*same_as_shift_clock* | *1..99*} — **internal mode only**

  An optional switch and value pair that specifies the high pulse width of the capture clock. The default value is *same_as_shift_clock*. Because the default shift_clock period is 10 ns, and the default capture clock period is 40 ns, the capture clock is high for 5 ns and low for 35 ns when using all default setting, including this current switch.

- -pi_setup_extra_cycles *int* — **external mode only**

  An optional switch and value pair that specifies the number of extra cycles in the PI setup. The default value is 1.

- -pi_hold_extra_cycles *int* — **external mode only**

  An optional switch and value pair that specifies the number of extra cycles in the PI hold. The default value is 1.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -pi_force_percentage *int* — **external mode only**

  An optional switch and value pair that specifies the PI force percentage. The default value is 0.

- -po_measure_percentage *int* — **external mode only**

  An optional switch and value pair that specifies PO measure percentage. The default value is 45.

- -min_capture_window_in_ssn_bus_cycles auto | *int*

  An optional switch and literal or value pair that creates a constraint on the capture window size, in addition to any other constraints. You can specify a value indicating a number of bus clock cycles from 1 to MAX_INT. If the computed capture window is smaller than the specified size, the capture window is increased to meet this minimum requirement. If the tool cannot meet this requirement, it reports an SSN_R11 DRC error.

  When writing the parallel testbench, the tool pulses the SSN, pulse_always, and pulse_in_capture clocks at least as many times as you specify with this switch during the capture window.

  If you only need to control the capture window number of cycles and not the number of times the capture clock is pulsed, use the set_core_instance_parameters command to set the capture_packets parameter to 1. This avoids a larger capture window than requested, because the tool can only add capture cycles based on this option and cannot reduce the size of the capture window calculation based on the other SSH timing specifications.

  Note

  This switch is only available for -mode_type internal; the tool reports an error if you use this switch with external mode.

  Note

  This switch should be necessary only in cases where the OCC you use requires a minimum number of pulses on any of the SSN clocks, pulse_always clocks, or pulse_in_capture clocks, because they are also being used as a free-running clock source for the capture clocks. This is an uncommon situation.

- -generated_timeplate_name *timeplate_name*

  An optional switch and string pair that specifies a base name for the timeplate that the tool uses when generating the default timeplate during system mode transition. The base name can consist only of letters, digits, and underscores, and cannot start with a digit. Each command using this switch must specify a unique base name.

Examples

The following example sets the internal capture clock period to 10ns for the physical block BlockD.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_current_physical_block BlockD
set_capture_timing_options -mode_type internal -capture_clock_period 10
```

Related Topics

get_capture_timing_option

report_capture_timing_options

reset_capture_timing_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_cdp_test_data

Context: patterns -silicon_insight

Mode: all

Sets or changes data values in a specified characterization and debug package (CDP) test.

Usage

set_cdp_test_data ***test_name*** {{{**-patch_user_pattern *pattern_name***
-variable_name *variable_name* [-variable_type read | write | any]} |
-dynamic_parameter_value *dynamic_parameter_name*}
-new_value *new_value*}

Description

The set_cdp_test_data command specifies data values for a specified CDP test.

Arguments

- ***test_name***

  A required string that specifies the CDP test to set the information on.

- **-patch_user_pattern *pattern_name* -variable_name *variable_name***
  [-variable_type read | write | any]

  Two required switch and literal pairs, with an optional additional switch and literal pair, that specify a variable in a particular pattern to patch with a new value. You can optionally specify whether to patch a variable that is read, a variable that is written, or a variable that is either. Variable names typically begin with a pattern set name in the following format: "*<pattern_set_name>*::*<register_name>*".

- **-dynamic_parameter_value *dynamic_parameter_name***

  A required switch and literal pair that specifies a dynamic parameter in which to set a new value.

- **-new_value *new_value***

  A required switch and literal pair that specifies the new value to set in the variable or parameter specified using the other arguments.

Examples

The following example patches the read variable chip::block4_I1.tdr_G.R with a value of "10011110":

```
set_cdp_test_data ICLNetwork -patch_user_pattern ICLNetwork.stil \
   -variable_name chip::block4_I1.tdr_G.R -variable_type read -new_value "10011110"
```

The following example sets the dynamic parameter TestCreator with a value of "John Doe":

```
set_cdp_test_data MemoryBist_P1 -dynamic_parameter_value TestCreator \
   -new_value { John Doe }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

get_cdp_test_data

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_cdp_test_diagnosis_option

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Enables including or excluding MemoryBist instruments from the bitmap diagnosis report for a specific TestStep wrapper or all TestStep wrappers of the specified CDP test, after the CDP diagnostic patterns are generated.

Usage

set_cdp_test_diagnosis_option ***test_name*** [-test_step *test_step*]
{**-include_instrument_list *include_instrument_list*** |
-exclude_instrument_list *exclude_instrument_list*}

Description

Only the instruments that were part of the original CDP diagnostic patterns can be included.

This setup applies to any diagnosis performed on the specified CDP test using the execute_cdp_test command for the current interactive SID session. Using this command can significantly speed up bitmap diagnosis reporting when trying to isolate failures for a reduced number of instruments while ignoring instruments with catastrophic failures, by eliminating the need for potentially costly regeneration of CDP diagnostic patterns.

This command does not affect the results and execution of the GO pattern for the specified CDP test.

Note

After you use this command to modify the scope of diagnosis for a given CDP test, do not use the -force_diagnose_on_failure switch with the execute_cdp_test command. The -force_diagnose_on_failure switch overrides any setup you have performed with this command.

Arguments

- ***test_name***

  Sets options for the named CDP test. A CDP test corresponds to a Patterns wrapper name in the PatternsSpecification.

- -test_step *test_step*

  Specifies the name of the test step wrapper from within the CDP test Patterns wrapper in the PatternsSpecification for which the specified instruments are included in or excluded from bitmap diagnosis. If this option is not specified, the specified instruments are included in or excluded from bitmap diagnosis for all TestStep wrappers within the CDP test Patterns wrapper.

  If you specify different TestStep wrappers for a given CDP test with multiple uses of this command and argument, the effect is cumulative. However, once you use this command

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

without using this argument for that CDP test, the tool discards the previous cumulative settings.

- **-include_instrument_list *include_instrument_list* | -exclude_instrument_list *exclude_instrument_list***

  Includes or excludes the specified instruments during diagnosis. If the <*include_instrument_list*> is empty, the diagnosis excludes all instruments. If the exclude_instrument_list is not specified, the test includes all instruments.

Examples

Example 1

The following example includes, in bitmap report diagnosis, all instances of instrument block_B_inst1.block_B_rtl_tessent_mbist_c22_controller_inst that are present in the MemoryBist_P1 CDP test but excludes all other instruments:

```
set_cdp_test_diagnosis_option MemoryBist_P1 -include_instrument_list \
        { block_B_inst1.block_B_rtl_tessent_mbist_c22_controller_inst }
```

Example 2

The following example excludes, in bitmap report diagnosis, instrument block_B_inst1.block_B_rtl_tessent_mbist_c22_controller_inst in the run_time_prog TestStep of the MemoryBist_P1CDP test but includes all other instruments in this TestStep. All other TestStep entries present in the MemoryBist_P1 CDP test are not affected by the instrument changes:

```
set_cdp_test_diagnosis_option MemoryBist_P1 -test_step run_time_prog \
        -exclude_instrument_list \
        { block_B_inst1.block_B_rtl_tessent_mbist_c22_controller_inst }
```

Example 3

The following example includes, in bitmap report diagnosis, all instruments present in the MemoryBist_P1 CDP test:

```
set_cdp_test_diagnosis_option MemoryBist_P1 -exclude_instrument_list {}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_cell_library_options

Context: unspecified, all contexts

Mode: setup

Sets the specified options for library parsing and design elaboration.

Usage

set_cell_library_options [-prefer_cell_library {on | off}] [-force]
[-remove_bad_pad_modes {off | on}]
[-report_duplicate_models {off | on}]
[-report_floating_nets {off | on}]
[-report_simulation_functions {off | on}]
[-report_undefined_nonscan_model_references {off | on}]
[-report_clock_gaters {off | on}]
[-report_errors_only {off | on }]
[-augment_dft_cell_selections {single_selection_only | on | off }]
[-create_default_dft_cell_selection {on | off}]
[-default_dft_cell_selection_name *default_dft_cell_sel_name*]
[-liberty_ignore_dont_use { on | off }]
[-liberty_populate_when_ports_differ {on | off }]
[-flatten_hierarchy_within_cells { on | off }]
[-populate_regexp_only_if_all_ports_match {off | on}]

Description

The set_cell_library_options command specifies options to control library parsing and design elaboration. When you run the set_cell_library_options command without arguments, it reports the current library options settings.

Arguments

- -prefer_cell_library {on | off}

  An optional switch and literal pair that controls the tool's behavior when reading in a design that has both a Verilog module and a cell library model with the same name. By default the tool chooses the cell library model.

  If your design has been elaborated, you cannot change the -prefer_cell_library option without losing setup data. If you want to change this option after elaboration, you must use the -force switch to explicitly indicate that it is alright for the tool to discard previously defined setup information.

  When you specify "off" before issuing the set_current_design command, the tool chooses the Verilog module.

- -force (available only in contexts that operate in no_rtl mode)

  An optional switch that specifies to suppress the error message that normally prevents the accidental deletion of previously defined setup information. If you execute the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

"set_cell_library_options -prefer_cell_library option" command after your design has been elaborated, the tool normally displays an error message indicating that if it executes the command, the setup information you have already specified is discarded. You use this switch to suppress this error message and indicate that it is acceptable for the tool to delete the previously defined setup data.

- -remove_bad_pad_modes {off | on}

  An optional switch that specifies that any mode Warning about a failed pad check is removed. If all modes are found bad, another Warning is issued that the cell is not available downstream as a pad cell .

```
//  Warning: Model 'differential_pad' has pad_function 'pad_io_inv' declared for
//           port 'PADN', but no pad_function 'pad_io' declared. An IO_inv must
//           have the IO also declared in the same mode.
//           Removing the bad (and only) mode from model 'differential_pad'.
//  Warning: Model 'differential_pad' will not be available downstream as a pad
//           cell because its only mode (set of port pad attributes) was found bad
//           and removed.  See previous messages.
//           Model from line 754 file 'data/libcomp_atpg.lib'
```

  By default, only a Warning is issued for a failed pad check, such as:

```
//  Warning: Model 'differential_pad' has pad_function 'pad_io_inv' declared for
//           port 'out2', but no pad_function 'pad_io' declared. An IO_inv must
//           have the IO also declared in the same mode.
//           Model from line 754 file 'data/libcomp_atpg.lib'
```

  When pad attribute checks are done, a check is made for the consistency of the set of attributes in each mode. If only pin attributes are specified, a single mode is created reflecting exactly those pad attributes for the pad mode checks. When an inconsistency is found in a mode, messages (Notes and Warnings) are issued to inform you of potential issues. A Warning indicates the mode is considered bad, but by default, that mode remains and therefore its attributes can be populated onto instances of the pad for use in test applications, such as boundary scan.If you specify "-remove_bad_pad_modes on", it causes all bad modes to be removed, so that a bad mode's attributes cannot be populated onto instances of the pad. If no modes remain, an additional Warning is issued indicating that the model with no remaining good modes is not available as a pad cell for downstream applications.

- -report_duplicate_models {off | on}

  An optional switch that specifies to issue a separate note for each duplicate model parsed by read_cell_library. Otherwise a summary of the total number of duplicate models is issued by default.

- -report_floating_nets {off | on}

  An optional switch that specifies to issue a separate note for each floating net in each model parsed by read_cell_library. Otherwise a summary of the total number of models with floating nets and the total number of floating nets is issued by default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -report_simulation_functions {off | on}

  An optional switch that specifies to report specifics about each model where simulation_function is parsed but not subsequently learned. These models cannot be used for test insertion of test logic into the netlist.

  The default is “off” which results in a summary message about the number of such models parsed. Clock gating cells are mentioned in a separate message because they are more frequent and are handled differently. You can change their simulation_functions to enable downstream DFT-C messaging that is more useful for users.

  The following is an example of the clock gating summary message when this switch is “off.”

  ```
  //  Warning: Parsed 'simulation_function = clock_gating_..' inside
  //  13 models which did not have two enables.//  Prepending
  'func_only_' to the clock_gating_..
  //  simulation_function.//  'set_cell_library_options -
  report_simulation_function on' to get
  //  per model Warning.
  ```

  The following is an example of the summary message all other simulation_functions when this switch is “off.”

  ```
  //  Warning: Parsed simulation_function inside 14 models which had
  //  no simulation_function learned.//  Will not be available for
  test insertion or DFT use downstream.//  'set_cell_library_options -
  report_simulation_function on' to get
  //  per model Warning.
  ```

  When this switch is “on” per model messages are issued. The following shows an example of per model clock gating messaging.

  ```
  //  Warning: Model 'clkgand' declared as
  //  simulation_function = 'clock_gating_and', but does not have
  //  two enables.//  Changing simulation_function to
  'func_only_clock_gating_and'.//  Model from line 69504 file
  'atpg.lib'
  ```

  The following shows an example of the per model messaging for other simulation_functions.

  ```
  //  Warning: Model 'latch_cell' parsed 'simulation_function = latch'
  //  but did not learn any simulation_function.
  //  Model from line 76076 file 'atpg.lib'
  ```

- -report_undefined_nonscan_model_references {off | on}

  For a scan cell, the “nonscan_model = ...;” statement can be used to specify each nonscan model/cell that the scan cell can replace prior to stitching scan cells into scan chains. When using regular expressions to generate nonscan_model statements, such statements can be generated for nonscan models that do not exist (are undefined).

  Each undefined nonscan_model reference can be reported by issuing the “set_cell_library_options command -report_undefined_nonscan_model_references {on}”

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

command. The default for this switch is “off” which issues a summary message stating the total number of undefined nonscan_model references in the entire file.

- -report_clock_gaters {off | on}

  Although read_cell_library processing can learn clock gaters, it cannot distinguish the system functional enable from the test/shift enable using the simulation analysis it uses. That is because they are typically ORed and functionally there is no way to distinguish them. You should declare one of them as the “func_enable” (or the “func_enable_inv” if it enables with 0), or the other as “test_enable” (or “test_enable_inv” if it enables with 0), or both. That is done automatically if the attributes are populated from read_liberty (this is the recommended method).

  When neither enable has an attribute, if exactly one of the two enable port names starts with “T” or “t”, followed by “E” or “e”, then it assumes that is the test enable, and the other the functional enable.

  When at least one clock gater has no parsed port attribute for either of the enables, then clock gater messaging occurs. This option to set_cell_library_options controls the messaging.

  When it is off (default), a summary message is issued, as follows:

  - The number of clock gaters that were not accepted as valid due to undeclared enable port functions.
  - The number of clock gaters that had port functions guessed using port names due to undeclared enable port functions.
  - A message to issue “set_cell_library_options -report_clock_gaters on” for a detailed message per clock gater.

  When you issue “set_cell_library_options -report_clock_gaters on”, an individual message is reported for each clock gater with no enable port attributes.

- -report_errors_only {off | on}

  An optional switch and literal pair that specifies to suppress warnings and notes and report error messages only. The default is off.

- -augment_dft_cell_selections { single_selection_only | on | off }

  An optional switch and literal pair that specifies whether to augment parsed dft_cell_selection information by adding any legal function not specified within that dft_cell_selection.

  For example, if a dft_cell_selection has nand cell but no xor cell, but a parsed model performs the xor function, then the model is added as an xor cell in the dft_cell_selection and can be used for test insertion and execution of the get_dft_cell command.

  By default, augmentation occurs only if a single dft_cell_selection is parsed in all read_cell_library files. When augmentation occurs augmented entries are not written out by the write_cell_library command, and the resulting dft_cell_selections are exactly as parsed. For more information, see the write_cell_library command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -create_default_dft_cell_selection { on | off }

  An optional switch and literal pair that determines whether a default dft_cell_selection is created when no dft_cell_selection is parsed.

- -default_dft_cell_selection_name *default_dft_cell_sel_name*

  An optional switch and string that sets the default dft_cell_selection to use for test insertion and the execution of the get_dft_cell command. Without this command, when two or more dft_cell_selections are parsed, test insertion uses the last parsed dft_cell_selection, and get_dft_cell creates an error unless a dft_cell_selection_name is specified with the get_dft_cell command. This option sets the default dft_cell_selection to use for both test insertion and the execution of the get_dft_cell command.

  The *default_cell_sel_name* must be the same as one of the names from the "dft_cell_selection (dft_cell_selection_name) ( )" statements parsed when the read_cell_library command is executed. If *default_cell_sel_name* is "" then the behavior reverts to the behavior before any execution of the "set_cell_library_options -default_dft_cell_selection_name" command.

- -liberty_ignore_dont_use { on | off }

  An optional switch and literal pair that determines whether "cell_type = prohibited;" is placed inside the corresponding Tessent model representing a Liberty cell containing "dont_use" inside the Liberty cell group for that cell. In Liberty, "dont_use" indicates that a synthesis tool should not insert that cell into a netlist when optimizing, and the default of -liberty_ignore_dont_use is "off", which causes "cell_type = prohibited;" to be placed inside the model so that Tessent test tools do not insert such a cell to create test logic needed for testing. To prevent the inclusion of "cell_type = prohibited" inside the corresponding (same name) Tessent model for a Liberty cell containing "dont_use", you must issue this command and set this switch to off before the read_liberty command containing the cells, to have any impact on the resultant Tessent cell library models.

- -liberty_populate_when_ports_differ {on | off }

  When a read_liberty cell has a name that matches a read_cell_library model, then if the port names also match, the information in the liberty cell is populated onto the model. However, if the port names do not match, the default is to not populate the information. Whether this switch is off (which is the default) or on, the mismatching ports are reported. If the switch is off, the report is a Warning. If the switch is on, the report is a Note.

  For example:

```
//  Warning: Not populating read_cell_library model 'SDFC' from
//      read_liberty cell 'SDFC' due to differing ports.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Output when the switch is off (default) when ports differ:

```
//  Warning: Not populating read_cell_library model 'SDFC' from
//       read_liberty cell 'SDFC' due to differing ports.
//       Model port 'GND' is missing in the Liberty model.
//       Model port 'VDD' is missing in the Liberty model.
//       Liberty from line 4 file '../data/SDFC.lib'.
//       Model from line 83 file '../data/libcomp.atpg'.
//  Note: 1 read_liberty cells had unmerged attributes due to
//       differing ports. If after checking the mismatching ports
//       reported above, population from those cells is desired,
//       'set_cell_library_options -liberty_populate_when_ports_differ on'
//       before read_liberty.
```

Output when the switch is on when ports differ:

```
//  Note: Populating read_cell_library model 'SGFD' from
//        read_liberty cell 'SGFD' despite differing ports.
//        Model port 'GND' is missing in the Liberty model.
//        Model port 'VDD' is missing in the Liberty model.
//        Liberty port 'XTRA' is missing in the
//        read_cell_library  model.
//        Liberty from line 23 file '../data/SGFD.lib'.
//        Model from line 71 file '../data/libcomp.atpg'.
```

- -flatten_hierarchy_within_cells { on | off }

  An optional switch and literal pair that specifies to flatten the internal hierarchy of library cells. The default for this switch is off.

  When you specify “set_cell_library_options -flatten_hierarchy_within_cells on”, then as all the read_cell_library models are parsed the tool replaces their internal “instance = <submodel>” by the contents of that submodel. The tool reports a Warning that the flattened models should not be used in normal flows due to missing internal names and mismatches with Verilog hierarchy, but only for special flows such as generating UDFM.

  Note

  Any model containing a RAM or ROM at any level of hierarchy is not flattened. Also, macros are not flattened.

  You must use this switch to flatten the hierarchy before the models have been processed. The tool will report an Error if you do so after the models have been processed.

- -populate_regexp_only_if_all_ports_match {off | on}

  An optional switch and literal pair that when set to on, specifies that the attributes defined in a model_attributes wrapper are only applied to a library cell if all the ports specified in the model_attributes wrapper match to a port on the library cell. The default for this switch is off.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

In the following example, no Liberty Cell inside liberty_source_file, or any Liberty Cell of a subsequent “read_liberty” command, have “cell_type = prohibited;” placed inside the corresponding Tessent read_cell_library model due to a “dont_use” in the Liberty syntax.

set_cell_library_options -liberty_ignore_dont_use on
read_liberty liberty_source_file
read_cell_library corresponding_tessent_models_filename
write_cell_library final_tessent_cell_library_with_liberty_info_merged_filename

In Liberty source given to read_liberty :

```
cell("AN2") {
    dont_use : true ;
```

Tessent write_cell_library for Liberty cell default model output :

```
model AN2 ..
...
    cell_type = prohibited;
    simulation_function = and;
    ...
```

Tessent write_cell_library for Liberty cell after specifying “set_cell_library_options -liberty_ignore_dont_use on”:

```
model AN2 ..
...
    cell_type = and;
    simulation_function = and;
...
```

Related Topics

delete_cell_library

delete_design

read_cell_library

read_verilog

set_current_design

set_design_include_directories

set_design_macros

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_chain_test

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies binary sequences for chain test.

Usage

Available when EDT is Off

set_chain_test [-SUppress_capture_cycle {OFf | ON}]
[{-SEquence *bit_string* | OFf} ... [-APpend | -REplace]}]

Available when EDT is On

set_chain_test [-SUppress_capture_cycle {OFf | ON}]
[{-SEquence {*bit_string* | OFf} ... [-APpend | -REplace]}]
[-Type {All | Nomask [with_1hot] | Mask [XOr_only | 1Hot_only]}]
[-NOMASK_First {ON | OFf}]

Description

Use this command to add one or more custom sequences to the current chain test or to replace the current chain test sequences with all new sequences.

When you generate one-hot chain patterns in either of the following cases:

- Multiple channels compact different numbers of chains
- Multiple EDT blocks contain different numbers of chains

the tool changes to non-observe mode and reports “<no chains observed>” for the smaller compactor channel or number of chains.

This disables observation after all chains have been observed once to reduce unnecessary messaging for chains that would otherwise be observed multiple times.

Note

When using SSN, the tool automatically suppresses capture cycles during chain test.

Arguments

- -SUppress_capture_cycle {OFf | ON}

  An optional switch and literal pair that suppresses the capture cycle from the chain test patterns. In the capture cycles, constrained pins are forced to their constrained values and cell constraints are justified.

  By default, chain test patterns contain a capture cycle when there is no C1 violation. In the case where DRC detects C1 violations on the level-sensitive ports or set/reset ports, the tool automatically suppresses the capture cycle for the chain test patterns in order to avoid

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

disturbing those C1-violated cells and prevent simulation mismatches. You can manually change the default setting (set_chain_test -suppress_capture_cycle off) to force the chain test patterns to include the capture cycle. In this situation, the tool masks out the scan cells with a C1 violation on either synchronous ports or set/reset ports to prevent mismatches, because the C1-violated cells on a synchronous or set/reset port may cause the loading value to be disturbed in the capture cycle. This masking effect could impact the tool's capability for diagnosing the chain test failure but should not impact the overall test quality.

This switch also suppresses the capture cycle of the first pattern (edt_setup) before chain and scan test patterns when the low-power EDT decompressor or input channel pipelining is in use. For more information, see "Low-Power Test" in the *Tessent TestKompress User's Manual*.

- -SEquence *bit_string* | OFF

  An optional, repeatable switch and string pair that specifies a binary sequence to shift into the scan chains, starting with the most significant bit.

  When EDT is off, if the length of a specified bit string sequence is shorter than a scan chain, the specified sequence is repeated to fill the rest of the scan chain. If the specified sequence is longer than the scan chain, the extra less significant bits are truncated.

  When EDT is on, the tool attempts to get the sequence to occur once in one-hot patterns. This is not a repeating sequence.

  Specifying OFF disables custom sequences. This enables you to revert back to using the default chain test.

- -APpend (Available when EDT is off)

  An optional switch that adds the specified sequences to the tool's current chain test sequences.

- -REplace (Available when EDT is off)

  An optional switch that replaces the tool's current chain test sequences with the specified bit sequences. This is the default.

- -Type All | Nomask [with_1hot] | Mask [XOr_only | 1Hot_only] (Available when EDT is On)

  An optional switch and literal pair that specifies the type of EDT chain test patterns that should be saved to minimize effort during simulation run time. The chain test patterns are saved when the write_patterns command runs. The possible literal values for -Type are:

  All — Saves chain test patterns with all types of masking. This option includes all non-masking and masking patterns as well as patterns to test all unused decoder values when all chain patterns are selected. As a result, the total number of patterns returned when you select this option may be greater than the sum of all non-masking and masking patterns. For more information on unused patterns, see "EDT Logic and Chain Testing" in the *Tessent TestKompress User's Manual*.

  Nomask — Saves only the non-masking chain test patterns.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

with_1hot — Saves the one-hot chain patterns in addition to the non-masking chain patterns.

Caution

Do not use the "Nomask" option alone when saving your production test patterns. (The "Nomask with_1hot" option is the default option and recommended usage. This caution applies only to the "Nomask" option without the one-hot masking patterns, using the "with_1hot" qualifier.) If you save the production pattern set using the "set_chain_test -type nomask" option, it is very difficult to identify the failing scan chain, and automatic chain diagnosis is not possible.

Mask — Saves only the masking chain pattern.

XOr_only — For the Xpress compactor, saves only flexible masking chain patterns.

Note

This does not apply to the basic compactor.

1Hot_only — Saves only one-hot masking chain patterns.

If you do not save one-hot patterns (Nomask or Mask [XOr_only] options), Tessent Diagnosis may not have enough information to run chain diagnosis successfully. Without one-hot patterns, chains in a compactor are scanned out together and the tool cannot identify the specific chain that contains the error. If you do not want to save one-hot patterns but still want to run chain diagnosis, you must inform Tessent Diagnosis of the failing chains using the following commands:

```
read_patterns scan_patterns
diagnose_failures <failure_file> -faulty_chain <faulty_chain_name>\
   <faulty_chain_type>
```

This approach assumes internal failing chain information was reported through another method. For more information, see the read_patterns and diagnose_failures commands.

For more information on masking patterns, see "Understanding Scan Chain Masking in the Compactor" in the *Tessent TeskKompress User's Manual*.

- -NOMASK_First {ON | OFf} (Available when EDT is On)

  An optional switch that specifies whether to place EDT non-masking chain patterns first. The default is off, which means that one-hot masking chain patterns are ordered first. The -Nomask_First on option is only valid with -type all; the option is ignored if used with -type mask.

Examples

Example 1

Assume a design contains two scan chains:

- chain1 is ten scan cells long

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- chain2 is eight cells long

The following example saves the tool's standard chain test patterns to the file *my_chain_test.ascii*. The sequences (minus X-padding) that this file contains appear after the example commands.

```
set_system_mode analysis
write_patterns my_chain_test.ascii -pattern_sets chain -replace
```

Pattern 0

chain1: 0011001100

chain2: 00110011

Example 2

The following example replaces the current chain test sequences with three new sequences and saves the resultant chain test patterns:

```
set_system_mode analysis
set_chain_test -sequence 01 -sequence 1
set_chain_test -sequence 1001 -append
write_patterns my_chain_test.ascii -pattern_sets chain -replace
```

For this example, the *my_chain_test.ascii* file contains the following three chain test sequences (minus X-padding):

Pattern 0

chain1: 0101010101

chain2: 01010101

pattern 1

chain1: 1111111111

chain2: 11111111

pattern 2

chain1: 1001100110

chain2: 10011001

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_checkpointing_options

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables checkpointing and specifies the associated options.

Usage

set_checkpointing_options [OFf | ON {[-PATTERN_FIle *filename*]
[-PATTERN_FOrmat {Ascii | Binary}] [-Faultlist_file *filename*] [-Replace]
[-Period *period*]}]

Description

Checkpointing is when the tool automatically saves test patterns at regular periods, referred to as checkpoints, throughout the pattern creation process. This is useful when ATPG takes a long time and there is a possibility it could be interrupted accidentally.

The set_checkpointing_options command specifies the filename and CPU time period in which the tool writes test patterns during test pattern generation.

You can save the fault list with the -Faultlist_file option. In the case of an interruption, if you have saved the generated patterns, you can use this fault list to avoid the resimulation step. When you specify the -Faultlist_file option, the tool saves the checkpointed fault list, including the fault class and sub-class of each fault, to the specified file. Aborted faults are also written to the file.

You can use the set_checkpointing_options command multiple times within a session and each time modify only the options you want to change. The tool retains your unmodified settings from the previous commands.

Note

Although you can issue the set_checkpointing_options command in any mode, the tool only writes out a checkpoint file in analysis mode with the set_checkpointing_options command set to On.

For additional information on checkpointing, refer to "Checkpointing Setup" in the Tessent Scan and ATPG User's Manual.

Arguments

- OFf

  A literal that specifies for the tool not to use the checkpointing functionality during test pattern generation. Patterns are not written to any file. This is the invocation default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ON

  A literal that specifies for the tool to use the checkpointing functionality. The tool writes test patterns that it generates to the file that you specified with the set_checkpointing_options command.

- -PATTERN_FIle *filename*

  An optional switch and string pair that specifies the name of the file into which you want to write the test patterns during test pattern generation.

- -PATTERN_FOrmat {Ascii | Binary}

  An optional switch and literal pair that enables the pattern files to be saved in binary or ASCII format. Binary is the default format.

- -Faultlist_file *filename*

  An optional switch and string pair that enables the fault list to be saved.

- -Replace

  An optional switch that forces the tool to overwrite the file if it already exists. If you use the -Replace option and the tool does not create any new test patterns, the tool does not update the file.

- -Period *period*

  An optional switch and integer pair that specifies the number of minutes of CPU time between each write of the test patterns. The default is 100 minutes.

Examples

The following example stores the generated test patterns every 120 minutes of CPU time in a file called *my_checkpoint_file*. After each 120-minute interval, the tool creates a new file until the ATPG process ends.

```
set_system_mode analysis
set_checkpointing_options on -pattern_file my_checkpoint_file -period 120
create_patterns
```

Related Topics

create_patterns

write_faults

write_flat_model

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_clock_controller_pins

Context: dft -logic_bist

Mode: setup

Specifies the connection information for the clock controller pins.

Usage

set_clock_controller_pins
{shift_clock | shift_clock_en | shift_capture_clock | scan_en |
| capture_en | lbist_en |
capture_procedure_index | diag_clock_en }
{***internal_pin_name***… | -no_connection} [-append]

Description

Use the set_clock_controller_pins command with the Hybrid TK/LBIST flow. Refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

If you issue this command multiple times with the same pin type, then the last specified pin name(s) are used. To specify multiple pin names corresponding to a pin type, specify all the names in the same command, or use the -append switch to add to the list of already specified pins.

The pins you specify with this command are connected to the single top-level LogicBIST controller. This command is global in context and not specific to the current EDT block.

Manual specification of the clock controller pins is not required when Tessent OCC core instances have been added. In this case, only the capture_procedure_index connection is required.

Alternatively, you can specify the -no_connection switch instead of internal_pin_name in an RTL IP-creation flow (skeleton flow), where the clock controller is not yet available in the design.

Note

The internal_pin_name string and **-no_connection** switch are mutually exclusive.

Arguments

- **shift_clock *internal_pin_name***

  An optional literal and repeatable string that specifies the clock controller shift clock input. This pin is to be driven by the LogicBIST shift clock output from the controller in LogicBIST mode.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **shift_clock_en *internal_pin_name***

  An optional literal and repeatable string that specifies the clock controller shift clock enable input. This pin is to be driven by the LogicBIST shift clock enable output from the controller in LogicBIST mode.

- **shift_capture_clock *internal_pin_name***

  An optional literal and repeatable string that specifies an input on the clock controller that needs to receive a clock that pulses during shift and capture, unlike the shift_clock that only pulses during the shift mode operation. This pin is driven by the LogicBIST shift capture clock output from the controller in LogicBIST mode. If this pin is not specified, then the output port is not created.This option is mutually exclusive with shift_clock. Specifying shift_capture_clock clears any previously specified shift_clock connections.

- **scan_en *internal_pin_name***

  An optional literal and repeatable string that specifies the clock controller scan enable input. This pin is to be driven by inverted LogicBIST capture enable output from the controller in LogicBIST mode.

- **capture_en *internal_pin_name***

  An optional literal and string that specifies the capture enable trigger LBIST clock enable input. This is typically connected to LBIST capture enable output.

- **lbist_en *internal_pin_name***

  An optional literal that specifies the LogicBIST enable pins.

- **capture_procedure_index *internal_pin_name***

  An optional literal that specifies the connections to the signal that identifies the NCP used for the current pattern. Then internal_pin_name string identifies the pins in the clock controller for the NCP index specified from MSB to LSB.

  A capture_procedure_index pin type has to be specified once per clock controller to add the list of NCP index pins for that clock controller. It is not possible to group the connections for the capture_procedure_index pin type for all clock controllers in a single command. The -append switch must be used to add to the pin specification for prior controllers.

- **diag_clock_en *internal_pin_name***

  An optional literal and repeatable string that specifies the clock controller diagnosis clock enable input. The LogicBIST controller diag_clock_en output drives this pin. If you are using the shift_clock_en clock controller pin, you need a diag_clock_en connection to perform LogicBIST diagnosis. You can only use the diag_clock_en signal when using “set_edt_options -single_bypass on.” When using shift_clock, the tool injects TCK during diagnosis through the shift_clock pin; if you want to inject TCK, you should use this pin.

- -no_connection

  An optional switch used instead of internal_pin_name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When you use this switch, you do not need to specify the design clock controller pin names and no connections are made. The pins are available at the LogicBIST IP boundary. You can use this switch in an RTL IP-creation flow, where the clock controller is not yet available in the design.

- -append

  An optional switch used to append to a previous list of pin names, instead of replacing. Without the -append switch, repeat usage of the set_clock_controller_pins command overwrites the previous list of pins for the specified pin type.

Examples

Example 1

The following example specifies using controller/scan_en as the scan enable input pin on the clock-controller interface:

set_clock_controller_pins scan_en controller/scan_en

Example 2

In this example, assume that the clock controller is not available in the design. The following example specifies the clock controller's shift clock but does not make any connection to the clock-controller interface:

set_clock_controller_pins shift_clock -no_connection

Example 3

This example connects the lbist_en output of the LogicBIST controller to the LogicBIST enable pin on multiple clock controllers. Note that the arguments should be specified in a list surrounded by braces.

**set_clock_controller_pins lbist_en **
{cc_clk1/lbist_mode cc_clk2/lbist_mode cc_clk3/lbist_mode}

Example 4

This example shows how to connect the NCP index pins of multiple clock controllers using the -append switch.

Note

You must specify a connection for each clock controller separately as the capture_procedure_index pin type requires a list of pin names be declared for each clock controller.

```
set_clock_controller_pins capture_procedure_index \
      [get_pins cc_clk1/ncp*]
set_clock_controller_pins capture_procedure_index \
      [get_pins cc_clk2/ncp*] -append
set_clock_controller_pins capture_procedure_index \
      [get_pins cc_clk3/ncp*] -append
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Issuing a report_clock_controller_pins command shows the following:

```
//  command: report_clock_controller_pins
//
//  PinType                   Connection
//  -------                   ----------
//  LBIST enable              cc_clk1/lbist_mode cc_clk2/lbist_mode
//                            cc_clk3/lbist_mode
//  Capture procedure index   cc_clk1/ncp_index[1] cc_clk1/ncp_index[0]
//                            cc_clk2/ncp_index[1] cc_clk2/ncp_index[0]
//                            cc_clk3/ncp_index[1] cc_clk3/ncp_index[0]
//
```

Related Topics

report_clock_controller_pins

set_lbist_controller_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_clock_controls

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables or disables the clock control definitions in the test procedure file.

Usage

set_clock_controls {**ON** | **OFf**}
[-ENFORCE_off_conditions_outside_capture_if_pulse_in_capture_source {ON | OFf}]

Description

When enabled, the clock definitions are used for ATPG. Clock control definitions cannot be used simultaneously with Named Capture Procedures (NCPs).

For more information, see "Support for Internal Clock Control" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- **ON**

  Required literal that enables clock control definitions. This is the default setting unless Named Capture Procedures (NCPs) are enabled in the test procedure file. When NCPs are enabled, the internal clocks are automatically disabled.

- **OFf**

  Required literal that disables clock control definitions. Unless the internal clocks are controlled by NCPs, turning clock control off leads the tool to treat the internal clocks as directly controllable and ignore the constraints and condition bits imposed on ATPG by the clock control definitions. Any generated patterns are invalid because the clock controller requirements modeled using the clock control definitions are completely ignored by ATPG and simulation. The ability to turn off clock controls is provided only for debug purposes, and is typically used to determine if a fault is untestable because of the clock control constraints (and if the fault is testable if those constraints are not present).

- -ENFORCE_off_conditions_outside_capture_if_pulse_in_capture_source {ON | OFf}

  An optional switch and keyword pair that enables turning off the following default behavior: when the source clock of a clock control is an pulse-in-capture clock, the tool uses condition bits to ensure that the internal clock is off in cycles beyond those generated by ATPG even if the source clock is pulsed beyond that point. This prevents disturbing the captured values.

  When generating patterns in the presence of internal clocks controlled by using clock control definitions, the tool must either decide whether to use condition bits to suppress additional internal clock pulses beyond the sequential depth of the generated pattern or assume that turning off the source clock suppresses further internal clock pulses. For example, consider that the clock control definition for a given internal clock specifies three condition bits: B1 to control the clock pulsing in the first cycle, B2 to control it pulsing in

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

the second cycle, and B3 to control it pulsing in the third cycle. If ATPG generates a one-cycle pattern that pulses this clock, the tool encodes B1=1. The tool then has the option of encoding B2=0 and B3=0, or leaving them unspecified to be random-filled.

If the clock control definition has no source clock defined or has an asynchronous free-running clock, then the tool must encode those two additional condition bits to ensure that any pulses the OCC receives beyond the first pulse are suppressed and the internal clock only pulses once. On the other hand, if the clock control definition has a source clock that can be controlled by the tester and is now defined as pulse-in-capture or pulse-always, this source clock is pulsed only once, and there is no need to waste encoding capacity to encode 00 into those two bits because the internal clock pulses only once regardless of the values loaded into B2 and B3.

The tool decides whether to use condition bits to suppress further internal clock pulses as follows:

If a clock control definition has no source clock, or its source clock or clocks are asynchronous free-running, the tool applies the condition bits needed to suppress additional internal clock pulses beyond the sequential depth of the generated pattern. You have no way to change this.

If generating retargetable patterns ("set_current_mode -type internal"), the tool applies the condition bits to suppress additional internal clock pulses. You cannot change this because the clocks at the core boundary typically do receive additional clock pulses when retargeted and merged with patterns for other cores.

If the source clock or clocks defined in a clock control block are pulse-in-capture, the tool applies the condition bits to suppress additional internal clock pulses. You can override this behavior by using "set_clock_controls on -enforce_off_conditions_outside_capture_if_pulse_in_capture_source off." This reduces the number of specified bits and may improve EDT compression. However, if you use this option, you must be sure that the source clock is not pulsed additional times than ATPG has pulsed it. Do not use the "set_external_capture_options -fixed_cycles" or "set_external_capture_options -minimum_cycles" commands when writing your patterns, since those options pulse the pulse-in-capture clocks when adding the padding cycles. Note that capture cycle padding is used for the type-3 LPCT controller, and as such, you should not use "set_clock_control on -enforce_off_conditions_outside_capture_if_pulse_in_capture_source off" in that case either.

If none of the previous conditions are satisfied and the source clock or clocks are defined and completely under ATPG control with no constraints, then ATPG does not apply the condition bits to suppress additional internal clock pulses beyond the sequential depth of the generated pattern.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example is a debugging flow that disables all clock control definitions in the test procedure file and generates test patterns:

```
set_clock_controls off
create_patterns
```

Related Topics

report_clock_controls

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_clock_gating

Context: dft -scan, dft -test_points

Mode: setup

Specifies clock gating cells whose unconnected test-enable ports need to be connected to either the scan enable signal or a specified signal (pin).

Usage

set_clock_gating {ON | OFf } [*object_spec*]
[-test_enable *port_name* [-INVert]]
[-functional_enable *functional_enable* [-INVert]]
[-clock_port *clock_port_name*]
[-clock_out_port *clock_out_port_name*]

Description

Note

When wrapper analysis is used, and the clock gater drives wrapper cells, then the port is connected to a composite signal that consists of scan_enable or'ed with the appropriate mode enable (for example, ext_mode_enable). Alternatively, when multiple scan enables are used, the appropriate scan enable signal are selected based on the type of scan cells each clock gater is driving.

The tool considers an unconnected port to be a port that is not connected to any net or is tied low (TIE0), tied high (TIE1), tied unknown (TIEX), or tied high impedance (TIEZ).

The tool does one of the following:

- Automatically checks if the test-enable port of each recognized clock gating instance or module is unconnected and, if unconnected, connects it to the scan enable signal.
- Automatically checks if the specified port of each specified instance is unconnected and, if unconnected, connects it to the scan enable signal.

Note

This command is cumulative. Every “set_clock_gating on” command invocation adds to the list of clock gaters that are connected, and every “set_clock_gating off” command invocation removes clock gaters from the list.

Arguments

- ON

  A literal that specifies to connect the unconnected test-enable port of the provided instances or modules. If no *object_spec* is provided, it applies to all identified clock gaters. This is the default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- OFf

  A literal that specifies to not connect the unconnected test-enable port of the provided instances or modules. If no *object_spec* is provided, it applies to all identified clock gaters.

- *object_spec*

  A Tcl list of one or more instances or modules or a collection of instances or modules.

- -test_enable *port_name* [-INVert]

  An optional switch and string pair that specifies the unconnected port of the clock gating cell to be connected to either the scan enable signal or to the specified signal.

  -INVert — By default, the expected input value for the unconnected port is set to 1 during the shift cycle. This switch specifies that the expected input value is set to 0 during the shift cycle.

  The -test_enable option is required when you issue the command with the on switch and provide *object_spec*. For the command to succeed, the port names specified with this option must exist on all the modules/instances defined with *object_spec*. To add modules that have different names for their test-enable or clock pins, issue the command multiple times.

  The default tool behavior is equivalent to "set_clock_gating on", meaning that all identified clock gaters are connected. The clock gaters are initially identified either through auto-identification or are inferred from the cell library.

  The tool marks the unconnected clock gaters and identifies their test-enable pins according the to following priority:

  - First, each clock gater explicitly provided with the "set_clock_gating on" command is marked, and the port specified by the -test_enable switch is identified as its test_enable pin.
  - Next, every instance whose module's cell_type equals clock_gating_and or clock_gating_or is marked, and the port with the test_enable or test_enable_inv attribute is identified as test-enable. If the clock gater is not driven by any synchronous clock it is not marked.
  - Finally, all auto-identified clock gater instances are marked with one of their unconnected inputs identified as test-enable.

- -functional_enable *functional_enable* [-INVert]

  An optional switch and string pair that specifies the functional enable port of the clock gating cell.

  -INVert — An optional switch which specifies that the active polarity of the functional enable port is 0 instead of 1

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When the functional enable is not specified for a user-defined clock gater, and the tool is not able to determine it, an error is issued during system mode transition:

```
//  Error: Identification of functional enable pin for the following
clock gaters failed. Please use the -functional_enable switch to
set_clock_gating command to specify the port.
//              <list of clock gaters>
//              …
```

- -clock_port *clock_port_name*

  An optional switch and string pair that specifies the clock port of the clock gating cell. Use this option when the clock gating device is to be connected to an enable signal associated with the clock domain.

- -clock_out_port *clock_out_port_name*

  An optional switch and string pair that specifies the clock out port of the clock gating cell. Use this option when the clock gating device has multiple outputs.

Examples

Example 1

The following command defines a clock gater instance /inst/cgc1 and forces the tool to connect its test enable port “te” to the scan enable or clock gating enable signal defined with the set_clock_gating_enable command.

set_clock_gating on /inst1/cgc1 -test_enable te

Example 2

You can use the following commands to prevent the tool from fixing specified clock gaters. The first command prevents fixing any identified clock gater below the inst1 instance. The second command prevents fixing any clock gater of module whose name matches the expression CLKG* .

set_clock_gating off /inst1

set_clock_gating off [get_modules CLKG*]

Related Topics

report_clock_gating

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_clock_gating_enable

Context: dft -scan, dft -test_points

Mode: setup

Associates clocks that control clock gating devices with enable signal drivers.

Usage

set_clock_gating_enable ***enable_signal_driver_pathname*** [*primary_input*]
[-Active {High | Low}]
[-Clock_domain *clock_signal_driver_pathname*…]

Description

The set_clock_gating_enable command assumes that the clocks controlling the clock gating devices (whose enable ports are to be automatically connected) were added to the design with the add_clocks command.

The tool connects the unconnected enable ports of clock gating devices to the enable signal associated with the controlling clocks of these clock gating devices. The clock gating devices, the enable ports of the clock gating devices, and their clock ports must be specified with the set_clock_gating command.

Arguments

- ***enable_signal_driver_pathname***

  A required string, single-element Tcl list or a collection that specifies either a new top-level port, an existing top-level enable port or an existing internal output pin to be connected to the unconnected enable port of a clock gating device.

  The enable_signal_driver_pathname argument can be either a top-level enable port or an internal instance pin. If an internal instance pin is specified, the tool attempts to locate its primary input driver by tracing back to a primary input via a simple path; that is, through only inverters or buffers. If automatic identification of the primary input driver fails, the value of the primary_input option is used, if provided. Identification of the primary input is needed to generate the ATPG dofile and test procedure files.

- primary_input

  An optional Tcl list or a collection that specifies a top-level enable port for an internal instance pin provided by the enable_signal_driver_pathname argument. The specified top-level port is used when generating the ATPG dofile and test procedure files.

- -Active {High | Low}

  An optional switch and literal pair that specify whether the enable signal is active low or active high. The default setting is high.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Clock_domain *clock_signal_driver_pathname*…

  An optional switch and a repeatable string that specifies the defined clock(s) controlling clock gating devices whose unconnected enable ports are to be connected to the specified enable signal's driver.

Examples

The following example specifies the default clock gating enable signal to be used when the tool does not find an enable signal associated with the specified clock domain:

```
set_clock_gating_enable x
set_clock_gating_enable y -clock_domain a
```

In this example, the signal x is used to enable all clock gaters except for those in clock domain a, which uses signal y.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_clock_gating_options

Context: dft -scan

Mode: setup

Specifies clock gater handling options for scan insertion.

Usage

set_clock_gating_options [-fix_when_func_en_always_active {on | off}]
[-fix_type_b_driving_le_flops {on | off}]

Description

Use this command to control clock gater handling for scan insertion.

Arguments

- -fix_when_func_en_always_active {on | off}

  An optional switch and literal pair that enables the tool to determine if the test enable input controlling the functional mode of clock gater cells stays active during shift and capture cycles. When set to on, the tool controls the test enable input of clock gater cells to guarantee correct scan chain operation. When set to off (default), the tool does not modify the existing test enable connection. You must ensure that the functional mode stays active during at least one pattern generation or simulation phase.

  Note

  If the tool determines that the test enable stays active during shift and capture cycles, it does not modify the connection. This switch affects the functional enable pin only.

- -fix_type_b_driving_le_flops {on | off}

  An optional switch and literal pair that defines the tool handling of type-B clock gaters driving leading edge flops. When set to off, which is the default, the tool reports S1 rule violations for such cases. When set to on, the tool connects the test enable pin of such clock gaters to ltest_en DFT signal or the signal defined with “set_scan_signals -ten”. After scannability DRC is done, the tool prints out the list of clocks that need to pulse in test setup to propagate the enable value through the clock gaters. The automation is limited to making connections to the clock gater and reporting the clocks. It is your responsibility to pulse these clocks in test setup.

  Note

  Connecting the test enable input of a clock gater to an always-on driver blocks fault propagation through the functional enable pin. To prevent the test coverage loss, consider avoiding the insertion of type-B clock gaters driving leading edge flops.

  For details, see Potential DRC Violation Clock Gater Embedding (Type-B) in the *Tessent Scan and ATPG User’s Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following examples both show the behavior with regards to a clock gater instance "cgc_fe_on" with the functional enable port FE tied on and the unconnected test enable port TE. Moreover, the clock gater drives some wrapper cells. Another clock gater "cgc_te_on" has its test enable pin pre-connected and constrained on. As seen in the following examples, the -fix_when_func_en_always_active switch does not affect this clock gater.

Example 1

set_clock_gating_options -fix_when_func_en_always_active on

set_system_mode analysis

…

analyze_scan_chains

report_clock_gating

```
-----------------------------------------------------------------------------------------
Clock Gating  Test Enable  Shift                Capture       Origin        Status
Instance      Port         Control              Control
-----------------------------------------------------------------------------------------
cgc_fe_on     TE           scan_en              ext_ltest_en  Cell library  Unconnected
cgc_te_on     TE           Existing connection  Functional    Cell library  Pre-connected
```

As shown in the report, the test enable port TE of clock gater "cgc_fe_on" gets connected to the scan_en signal to ensure proper shift operation. The scan enable signal is combined through an OR gate with ext_ltest_en for proper operation of the wrapper cells.

Example 2

set_clock_gating_options -fix_when_func_en_always_active off

set_system_mode analysis

…

analyze_scan_chains

report_clock_gating

```
-----------------------------------------------------------------------------------------
Clock Gating  Test Enable  Shift                Capture       Origin        Status
Instance      Port         Control              Control
-----------------------------------------------------------------------------------------
cgc_fe_on     TE           Existing connection  Functional    Cell library  Unconnected
cgc_te_on     TE           Existing connection  Functional    Cell library  Pre-connected
```

This report shows that the tool does not touch the test enable port of clock gater "cgc_fe_on". Instead it relies on the existing connections and constraints that make the clock properly reach the scan elements in the clock gater's fanout, both for scan shift and wrapper cell operation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_clock_off_simulation

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables or disables the simulation where all clock primary inputs are at their "off" value, other primary inputs have been forced to values, and state elements are at the values scanned in or resulting from capture in the previous cycle.

Usage

set_clock_off_simulation **ON** | **OFf**

Description

When simulating this event, the capture data for inputs is provided to leading edge triggered flip-flops.

Tip

If you use ATPG Expert (create_patterns), you do not need to issue this command for better handling of DRC violations. ATPG Expert automatically enables it when needed. You may get better results by letting ATPG Expert decide when to use this command except in the case of transparent latch handling.

Note that the set_clock_off_simulation command affects ATPG (subsequent create_patterns commands) but not logic simulation (subsequent simulate_patterns commands).

Arguments

- **ON**

  A literal that performs ATPG for all clock primary inputs at their "off" values.

- **OFf**

  A literal that performs ATPG for all clock primary inputs at their active values. This is the default.

Examples

The following example enables clock_off simulation prior to creating patterns:

```
set_system_mode analysis
set_clock_off_simulation on
create_patterns
```

Related Topics

set_split_capture_cycle

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_clock_options

Context: dft, patterns

Mode: setup

Enables you to specify multiple options for a single clock.

Usage

set_clock_options ***object_spec*** [-label]
[[-period *period* [s | ms | us | ns | ps | fs]]
| [-reference *ref_obj_spec* [-reference_inv *ref_inv_obj_spec*]
[-freq_multiplier *freq_multiplier*] [-freq_divider *freq_divider*]]]
[-pulse_always {ON | OFf} | -pulse_in_capture {ON | OFf}] [-off_state {0 | 1}]
[-dft_inject_node *dft_inject_node*] [-no_dft_connections]

Description

The set_clock_options command enables you to edit options for an existing clock without having to delete the clock and create a new clock with the add_clocks command.

Arguments

- ***object_spec***

  A required string that specifies an existing clock using the name of a port, pin, pseudo_port, label, or a collection of one port, or one pin, one pseudo_port, or one label object. The *object_spec* cannot be a branch.

- -label *label*

  An optional switch and string pair that specifies an arbitrary alias name for the clock.

- -period *period*

  An optional switch and real number pair that specifies the base clock period. Setting the clock period changes the clock type from synchronous source to asynchronous source. Setting the clock period to a null string changes the clock type from asynchronous source to synchronous source.

- [s | ms | us | ns | ps | fs]

  An optional literal that specifies a time unit if you use the -period switch. If you do not use this switch, the default is “ns.”

- -reference *ref_obj_spec* **(“patterns -ijtag” only)**

  An optional switch and string pair that specifies the source primary input port that is connected to the internal clock pin you specified with the *object_spec* argument. You use this switch only for an existing source clock on a pseudo_port, not for a port. Setting the *ref_obj_spec* to a null string changes the clock type from generated to synchronous source.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -reference_inv *ref_inv_obj_spec* **("patterns -ijtag" only)**

  An optional switch and string pair that specifies a second primary input port if the -reference pin is a differential pin. You can use this switch only if you also use the -reference switch or if the *object_spec* is already a generated clock.

- -freq_multiplier *freq_multiplier* **("patterns -ijtag" only)**

  An optional switch and string pair that defines the multiply factor of the clock frequency between the reference pin and the internal clock pin specified with the *object_spec* argument. You can use this switch only if you also use the -reference switch or if the *object_spec* is already a generated clock.

- -freq_divider *freq_divider* **("patterns -ijtag" only)**

  An optional switch and string pair that defines the division factor of the clock frequency between the reference pin and the internal clock pin specified with the *object_spec* argument. You can use this switch only if you also use the -reference switch or if the *object_spec* is already a generated clock.

- -pulse_always {ON | OFf}

  An optional switch and literal pair that specifies the *object_spec* as a pulse-always clock. A pulse-always clock is pulsed in every cycle. The clock you specify also pulses with any other clock, ignoring the specified clock restriction setting. You can use this switch only when *object_spec* is an existing source clock. This switch has no default value.

  Note

  SSN bus clocks are not permitted to be specified as pulse-always. If you specify an SSN bus clock as pulse-always, the tool reports an SSN_R23 error during DRC.

- -pulse_in_capture {ON | OFf}

  An optional switch and literal pair that specifies the *object_spec* as a pulse-in-capture clock. A pulse-in-capture clock is always pulsed during the capture window. You can use this switch only when *object_spec* is an existing source clock. This switch has no default value.

- -off_state {0 | 1}

  An optional switch and literal pair that specifies the off state of the clock. You cannot use this switch if *object_spec* is an existing async source or if you use the -period switch. This switch has no default value.

- -dft_inject_node *dft_inject_node* (**"dft" only**)

  An optional switch and string pair that specifies where the DFT object affecting the clock (for example, a TCK injection multiplexer) should be inserted along the clock path. You can introspect the status of this switch with the "get_clock_option -dft_inject_node" command and switch.

- -no_dft_connections

  A Boolean switch that instructs the tool to not infer DFT connections to this clock. When choosing the clock for a dedicated wrapper cell or a test_point in the insert_test_logic

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

command, the clock used by the majority of the flip-flops in the the fan-in or fanout of the node is normally used. If that clock was added with the -no_dft_connections switch, it is not chosen and instead the next majority clock is used. If all clocks the fan-in or fanout were added with the -no_dft_connections switch, the largest clock domains in the design without this switch are selected.

Examples

The following example creates a synchronous clock with a label, then uses the set_clock_options command to change the clock to asynchronous and assign a different label name:

SETUP> add_clocks 0 I3 -pulse_always -label sync3

SETUP> report_clocks

```
User-defined Clocks (1):
Sync and Async Clocks
=====================
 ------  ------  ---------  -------------  --------
 Name    Label   Off State  Constraints    Internal
 ------  ------  ---------  -------------  --------
 'I3'    sync3       0      Pulse always       No
```

SETUP> set_clock_options I3 -label async3 -period 100

SETUP> report_clocks

```
User-defined Clocks (1):
Sync and Async Clocks
=====================
 ----  ------  ---------  ----------------  --------  ------------
 Name  Label   Off State    Constraints     Internal     Period
 ----  ------  ---------  ----------------  --------  ------------
 'I3'  async3     -        Asynchronous       No        100.00ns
```

Related Topics

add_clocks

delete_clocks

get_clock_option

get_clocks

report_clocks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_clock_restriction

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether ATPG can create patterns with more than one active capture clock.

Usage

set_clock_restriction {**ON** | **OFf** | **Domain_clock** [-Edge_interaction
| -Any_interaction]} [-PULSE_interacting_clocks {OFf | ON}]
[-MAX_SKEW {Half_cycle |
One_cycle | Multiple_cycles}]
[-SAME_clocks_between_loads {OFf | ON}]
[-COMPatible_clocks_between_loads {OFf |ON}]
[-IGNORE_unobservable_cells {ON | OFf}]

Description

The set_clock_restriction command changes the default behavior of ATPG regarding the creation of test patterns that have more than one active clock line.

Note

Pulse-always clocks are excluded from clock restriction checking when you choose ON. For example, if you choose ON, and two pulse-always clocks are defined, then only one clock can be used in addition to the pulse-always clocks.

If you choose Domain_clock, the tool forces pulse-always clocks to be compatible with other clocks.

Arguments

- **ON**

  A literal that specifies for the ATPG to create only patterns with, at most, a single active clock.

  The ATPG treats equivalent clocks as a single clock. However, it treats multiple active clocks as a conflict condition and performs an exhaustive search for conditions necessary to detect the fault with no more than one active clock. Faults that the ATPG detects at primary outputs (POs) that connect to clocks must also satisfy this condition.

  During the bus contention prevention analysis portion of the ATPG, Tessent FastScan and Tessent TestKompress turn off any clock pins that the ATPG does not require for fault detection.

- **OFf**

  A literal that specifies for the ATPG process to create patterns with as many clocks on as it requires to detect faults. Using this option may cause race conditions due to multiple active clocks. Prevent these race conditions by specifying the On argument.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For Tessent FastScan and Tessent TestKompress — These tools concurrently apply any clocks they require for a given pattern. They display a message at the end of the ATPG run to indicate the number of patterns that had more than one active clock. When you change the clock restriction to off, these tools reset the ATPG untestable faults to undetected-uncontrolled.

- **Domain_clock**

  A literal that specifies for ATPG to create patterns with compatible clocks on. This enables the tool to generate patterns that pulse compatible scan chain capture or system clocks at the same time. Normally, only one scan chain capture or system clock is pulsed in a cycle. The tool treats two clocks as compatible if their domains are completely independent of each other; that is, they can be in their "on" state at the same time without causing clock skew problems or race conditions.

  The tool automatically analyzes circuit data from DRC to determine the compatible clocks. Therefore, if you use this option prior to DRC, the tool defers running it until DRC.

  Note

  The tool's analysis of clock domains only considers Tie-0 or Tie-1 gates that are Tie-0 or Tie-1 in both combinational and sequential ATPG.

  For information about how the -same_clocks_between_loads and -compatible_clocks_between_loads switches interact with this literal, see the examples.

- -Edge_interaction

  An optional switch that, when the Domain_clock setting is in effect, specifies for the tool to determine if two clocks are compatible (C) or incompatible (I) based on their edge interactions, as summarized in Table 8-3. The table covers the case where a source memory element is clocked by one clock, a sink memory element is clocked by a different clock, and the source and sink interact (the sink can capture data from the source).

  Note

  Take care when interpreting what the "C" and "I" mean: "C" means only that the tool treats the result of the particular combination as being unaffected by clock skew. "I" means the tool treats the result of the particular combination as susceptible to skew.

  You can run the set_clock_restriction command with only this switch, and the tool retains whatever clock restriction setting (On, Off, or Domain_clock) is currently in effect. This switch has no effect on the analysis used for ATPG unless Domain_clock is in effect; however, it does affect the output of the report_clock_domains command.

  This is the default upon invocation of these tools.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-3. Default Clock Compatibility Summary at a Clock Domain Boundary

		Sink			
		LE[1]	**TE**[2]	**AH(TE)**[3]	**AL(LE)**[4]
Source	**LE**[5]	I	C	C	I
	TE[6]	C	I	I	C
	AH(LE)[7]	I	C	C	I
	AL(TE)[8]	C	I	I	C

1. LE = Edge sensitive element captures on PI clock's leading edge
2. TE = Edge sensitive element captures on PI clock's trailing edge
3. AH(TE) = Active high, level sensitive element captures on PI clock's trailing edge
4. AL(LE) = Active low, level sensitive element captures on PI clock's leading edge
5. LE = Edge sensitive elementchangesonPIclock'sleadingedge
6. TE = Edge sensitive elementchangesonPIclock'strailingedge
7. AH(LE) = Active high, levelsensitiveelementchangesonPIclock's leadingedge
8. AL(TE) = Active low, levelsensitiveelementchangesonPIclock's trailingedge

ATPG Expert automatically sets the appropriate switch based upon the fault type. For stuck-at faults, ATPG Expert always sets "set_clock_restriction domain_clock-edge_interaction".

- -Any_interaction

  An optional switch that, when the Domain_clock setting is in effect, specifies for the tool to do a more conservative analysis of the clocking relationship between interacting source and sink memory elements in different clock domains. If the source is clocked by one clock, the sink is clocked by a different clock, and the two elements interact (the sink can capture data from the source), this switch causes the tool to treat the two clocks as incompatible. In other words, all the conditions shown in Table 8-3 result in a determination of incompatibility.

  You can run the set_clock_restriction command with only this switch, and the tool retains whatever clock restriction setting (On, Off, or Domain_clock) is currently in effect. This switch has no effect on the analysis used for ATPG unless Domain_clock is in effect; however, it does affect the output of the report_clock_domains command.

  ATPG Expert automatically sets the appropriate switch based upon the fault type. For transition faults, ATPG Expert always sets "set_clock_restriction domain_clock -any_interaction -compatible_clocks_between_loads on".

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -PULSE_interacting_clocks {OFf | ON}

  An optional switch and literal pair that specifies whether the interacting clocks can be pulsed simultaneously during test. By default, the tool turns off the pulsing of interacting clocks. The ON option enables pulsing pairs of interacting clocks during test generation.

  Note

  This switch is effective only when you also specify the Domain_clock option.

  When this switch is ON, the tool masks some state elements located at the inter-clock domain in order to avoid simulation mismatches. Therefore, you notice more X values at the state elements than when this switch is OFF. The behavior of cross-domain path masking depends on the setting of the max clock skew, as described below for the -Max_skew switch.

  Note

  The tools treats user-defined compatible clocks (created with the add_synchronous_clock_group command) as compatible clock domains and does not mask out cross-clock domain paths between compatible clocks.

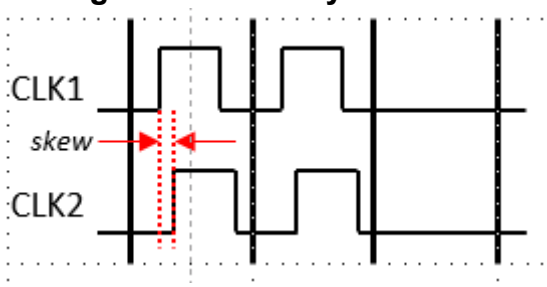
- -MAX_SKEW {Half_cycle | One_cycle | Multiple_cycles}

  An optional switch and literal pair that specifies the maximum skew between interacting clocks pulsed during capture. The maximum skew determines whether the observation state element should be masked or if fault simulation can give credit for detection. This option is effective only when the -Pulse_interacting_clocks option is enabled.

  The maximum skew settings impact fault simulation as follows:

  - Half_cycle — Only masks cross-clock domain paths of the same edge. Figure 8-2 shows half-cycle skew:

Figure 8-2. Half-Cycle Skew

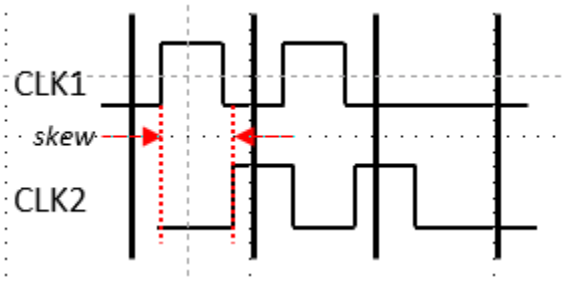


  - One_cycle — Masks cross-clock paths of the same clock edge, leading edge to trailing edge of the same cycle, and trailing edge to the leading edge of the next cycle. It enables the transition to be tested by launching at the leading (trailing) edge of one clock and capturing at the next cycle leading (trailing) edge of another clock.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 8-3 shows one-cycle skew:

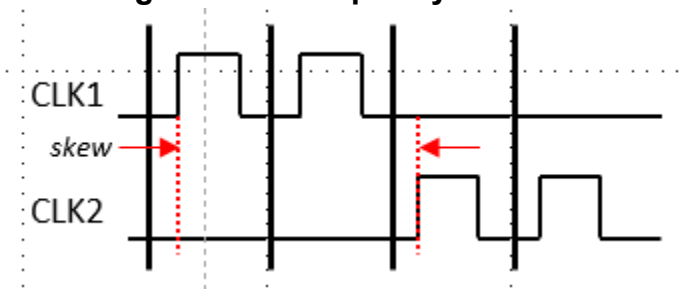
Figure 8-3. One-Cycle Skew



- Multiple_cycles — Masks all cross-clock domain paths. Tool assumes multiple clock cycles between all cross-clock domain paths and simulates them as asynchronous.

  Figure 8-4 shows multiple-cycle skew:

Figure 8-4. Multiple-Cycle Skew



The default maximum skew settings are as follows:

- Static fault models (stuck-at and bridging) — Default setting is Half_cycle, although the default is One_cycle when you also use the -Any_interaction switch.
- At-speed fault models (transition and path delay) — When the -same_clocks_between_loads switch is disabled, interacting clocks may be pulsed in adjacent cycles. In this case, the default setting is One_cycle, which enables transition from one edge to the same edge in the next cycle. When the -same_clocks_between_loads switch is enabled, interacting clocks are pulsed in every cycle and all clock interactions should be considered invalid. To reduce the chance of simulation mismatches, the default setting is Multiple_cycle.

You can use the -max_skew switch to overwrite the default clock skew to either prevent simulation mismatch (by specifying more clock skew than the default) or obtain better test coverage (by specifying less clock skew than the default). However, when changing the max skew to be less than the default, it must match with the design clock timing; otherwise, there is the possibility of simulation mismatch.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -SAME_clocks_between_loads {OFf | ON}

  An optional switch and literal pair that specifies whether to use the same clocks in every cycle between load/unload operations (during capture cycles). This switch is useful for at-speed test generation when you want each pattern to test for delay faults in only one clock domain. When clocks are pulsed between loads, they are all pulsed for every cycle between loads.

  Generally, each test pattern created with this switch enabled pulses the same clocks in every test cycle that pulses clocks. Multiple load patterns are an exception: for a multiple load pattern, the switch applies to each load/unload interval independently. Therefore, the clocks used in a particular interval might not be the clocks used in another interval in the same multiple load pattern.

  When enabled, this switch has no effect on pattern simulation. However, the tool transcribes a warning message if a pattern violates the restriction.You can run the set_clock_restriction command with only this argument, and the tool retains whatever other clock restriction setting (On, Off, or Domain_clock) is currently in effect.

  OFf — A literal that disables the requirement to use the same clocks in every cycle within a load/unload interval during pattern creation. This is the invocation default.

  ON — A literal that specifies to use the same clocks in every cycle within each load/unload interval during pattern creation.

  Refer to "Example 2" on page 2381 for example waveforms using this switch. Compare this to "Example 3" on page 2382 and "Example 4" on page 2383, which do not use this switch.

- -COMPatible_clocks_between_loads {OFf | ON}

  An optional switch and literal pair that specifies that only compatible clocks can pulse between load/unload operations (during capture cycles). This switch is used with the domain_clock option.

  OFf — A literal that disables the compatible clock requirement. When set, incompatible clocks cannot pulse in the same cycle, but they can pulse at different capture cycles.

  ON — A literal that specifies that only compatible clocks can pulse within capture cycles. When set, incompatible clocks never pulse between load/unload operations.

  The "-compatible_clocks_between_loads on" switch and literal pair is not compatible with the -edge_interaction switch or -pulse_interacting_clocks .When you specify "-compatible_clocks_between_loads on" and -edge_interaction, the tool reports an error message. It also reports an error message when you specify "-compatible_clocks_between_loads on" and -pulse_interacting_clocks.

  Tip

  If you want to create patterns that test only intra-clock domain faults, and you want to enable multiple non-interacting clock domains to be tested simultaneously in order to reduce pattern count, run the following command prior to creating patterns:

  ```
  set_clock_restriction domain_clock -any_interaction
  -compatible_clocks_between_loads on
  ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Refer to "Example 3" on page 2382 for example waveforms using this switch. Compare this to "Example 2" on page 2381 and "Example 4" on page 2383, which do not use this switch.

- -IGNORE_unobservable_cells {ON | OFf}

  An optional switch and literal pair that bypasses checking any unobservable nonscan or scan cells for clock interaction purposes. You can run the set_clock_restriction command with only this argument, and the tool retains whatever other clock restriction setting (On, Off, or Domain_clock) is currently in effect.

  ON — A literal that enables bypassing the check for unobservable nonscan or scan cells.

  OFf — A literal that disables bypassing the check for unobservable nonscan or scan cells.

  For example, if two clocks interact on a state element that is unobservable due to pin constraint or cell constraint, then the tool does not consider these two clocks incompatible for this statement when this switch is set to ON.

Examples

Example 1

The following example checks the current clock restriction setting, adds the additional restriction that the tool use the same clocks in every cycle within each load/unload interval during pattern creation, then changes the basic setting from Domain_clock to "on":

report_environment

```
...
clock restriction =       Domain_clock
...
```

set_clock_restriction -same_clocks_between_loads on
report_environment

```
...
clock restriction =       Domain_clock, same_clocks_between_loads
...
```

set_clock_restriction on
report_environment

```
clock restriction =       ON, same_clocks_between_loads
```

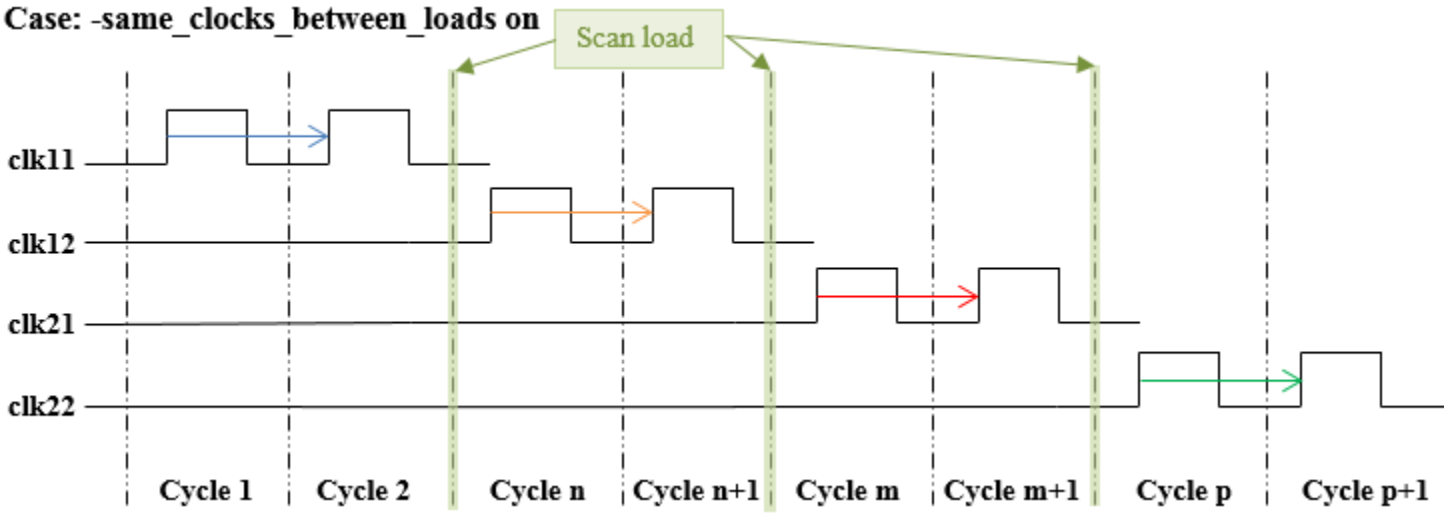
Example 2

In the following example, the tool uses the same clocks in every cycle with a load/unload interval during pattern creation. See Figure 8-5 for example waveforms. The green highlights between Cycle 2/Cycle n, Cycle n+1/Cycle m, and Cycle m+1/Cycle p represent load cycles that occur between them.

set_clock_restriction on -same_clocks_between_loads on

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 8-5. Same Clocks Between Loads On

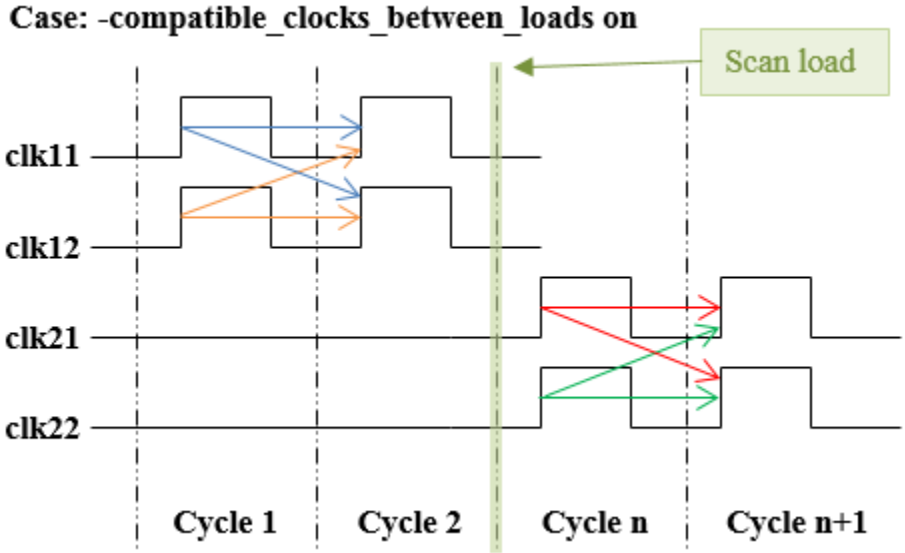


Example 3

In the following example, the tool only allows compatible clocks to pulse within capture cycles. Incompatible clocks do not pulse between load/unload operations. See Figure 8-6 for example waveforms. In this case, clk11 and clk12 are compatible clocks. The green highlights between Cycle 2/Cycle n represents a load cycle that occurs between them.

set_clock_restriction domain_clock -compatible_clocks_between_loads on

Figure 8-6. Compatible Clocks Between Loads On



For at-speed fault models, when you are not using named capture procedures, and there are no user defined settings or you are not calling create_patterns with the -override switch, the tool automatically turns on hold_pi and masks all primary outputs. It also automatically sets clock restriction to "domain_clock" with -any_interaction and -compatible_clocks_between_loads on.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

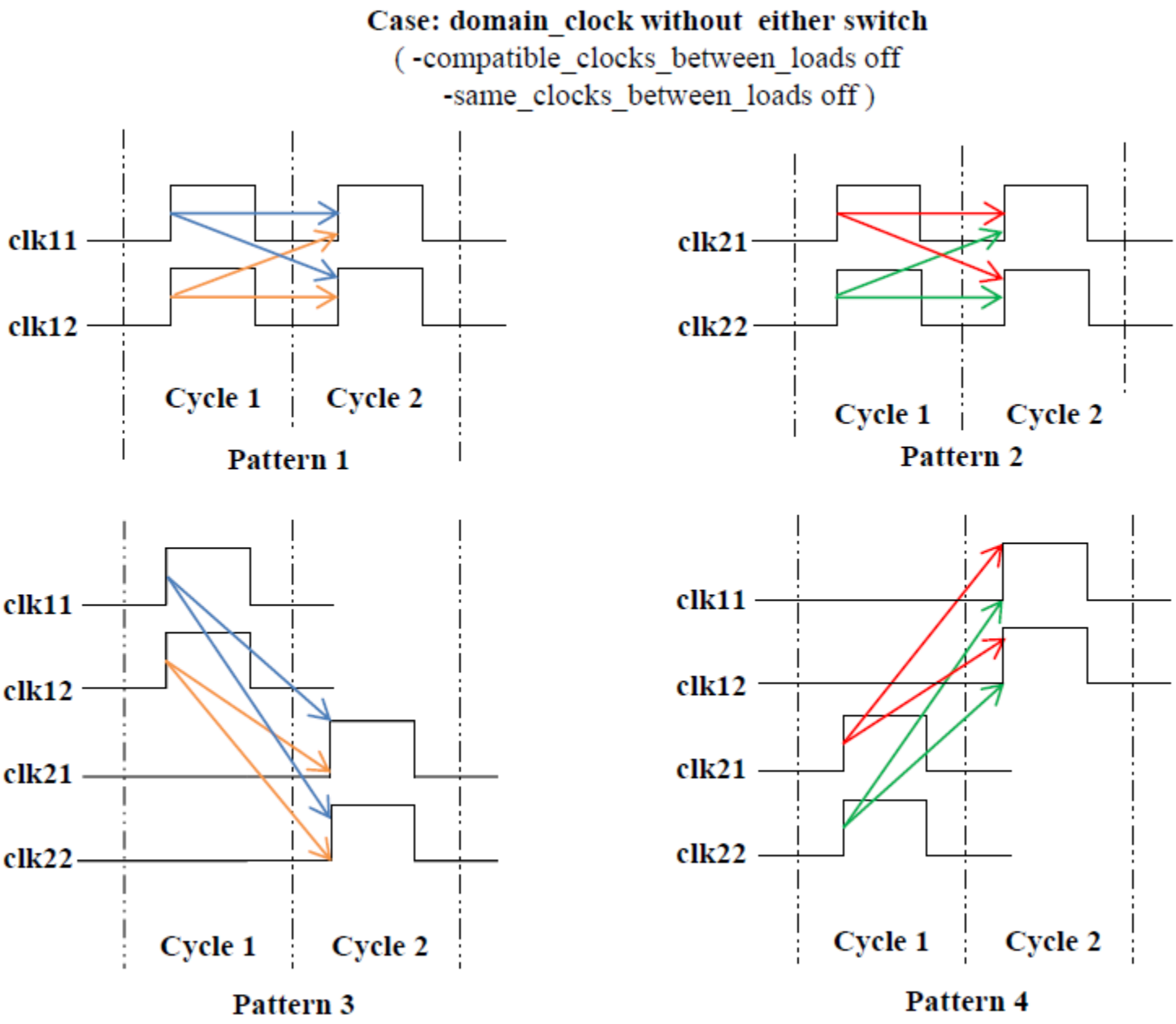
Example 4

In the following example, the tool is not required to use the same clocks in every cycle within a load/unload interval during pattern creation, because neither the -same_clocks_between_load nor the -compatible_clocks_between_loads switch is specified with “on.” The tool also disables the compatible clock requirement, so that incompatible clocks do not pulse in the same cycle but can pulse at different capture cycles. See Figure 8-7 for example waveforms. In this case, clk11 and clk12 are compatible clocks and clk21 and clk22 are compatible clocks, but clk11 and clk12 are incompatible with clk21 and clk22.

set_clock_restriction domain_clock

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 8-7. domain_clock Alone



When using create_patterns to create transition fault patterns, by default the tool sets the clock restrictions to domain_clock (edge interaction) and the -compatible_clocks_between_loads switch to on. It also sets the set_transition_holdpi and set_output_masks to on.

Related Topics

add_synchronous_clock_group

report_clock_domains

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_compactor_connections

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup

Specifies the scan chains for the tool to connect to a particular scan channel output.

Usage

set_compactor_connections {**-CHANnel** ***channel_number***} {**-CHAIns** {***chain_name***…}}

Description

The set_compactor_connections command specifies a list of scan chains whose outputs you want the tool to compact into the specified EDT scan channel output. The tool makes these connections using one or more stages of XOR gates in the compactor circuitry. If you use this command to specify the compactor connections for one scan channel and group of scan chains, you must use it for all the channels and chains in the design.

When generating the EDT IP, the tool uses the sequence of add_scan_chains commands to connect the EDT hardware to the scan chains. You can change the order of the add_scan_chains commands in your dofile to change how they are connected to the decompressor and compactor. However, this method changes both the decompressor and compactor connections for a particular chain.

Alternatively, you can use the set_compactor_connections command to override the tool's default connections and explicitly define the connections between scan chains and compactor. Using this command enables you to change the compactor connections without changing the default decompressor connections to those chains.

If you have dual configurations, you can still define the compactor connections using the set_compactor_connections command, but only for the configuration that uses all scan channels. The set_compactor_connections command is not tied to a specific configuration since you only need to define connections once for each channel.

Arguments

- **-CHANnel** ***channel_number***

  A required switch and positive integer pair that identify the scan channel to connect to the specified group of scan chains.

  When you specify the number of channels using the set_edt_options command -Channels switch, the tool numbers the channels consecutively starting at 1.

- **-CHAIns** ***chain_name***…

  A required switch and repeatable string pair that specify the internal scan chains whose outputs the tool should connect through the compactor circuitry to the specified scan channel output.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

For a design with eight scan chains named “chain1” through “chain8”, the following example sets up the EDT logic to have two scan channels. It then connects the outputs of chain1, chain5 and chain6 through the compactor to channel 1 and displays these connections. Any channel listed with “no connections” next to it still needs to be defined. The example goes on to define the remaining compactor connections and displays the completed configuration.

set_edt_options -channels 2
set_compactor_connections -channel 1 -chains chain1 chain5 chain6
report_compactor_connections

```
//  Reporting user-defined compactor connections:
//
//  Channel   Chains
//  -------   ------
//     1      chain1, chain5, chain6
//     2      no connections
//
```

set_compactor_connections -channel 2 -chains chain2 chain3 chain4 chain7 chain8
report_compactor_connections

```
//  Reporting user-defined compactor connections:
//
//  Channel   Chains
//  -------   ------
//     1      chain1, chain5, chain6
//     2      chain2, chain3, chain4, chain7, chain8
//
```

Example 2

In the following example, compactor connections are defined for a dual configuration. Note, the connections are only defined for the configuration that uses all scan channels.

set_current_edt_configuration config_high
set_edt_options -input 1 -output 1
set_current_edt_configuration config_low
set_edt_options -input 3 -output 3
set_compactor_connections -channel 1 -chains ...
set_compactor_connections -channel 2 -chains ...
set_compactor_connections -channel 3 -chains ...

Related Topics

report_compactor_connections

reset_compactor_connections

set_edt_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_config_value

Context: unspecified, all contexts

Mode: all modes

Sets the value of an element in the configuration data.

Usage

set_config_value ***config_spec***
[-in_wrapper *wrapper_object_spec*]
[-id {*position* | id_name | all} | -value_id {*position* | value_name | all}]
{*values* | -specify_default | -unspecify | -comment *comment* [-end_of_line]}
[-partition *partition*]

Description

Note

Use the add_config_element command instead of set_config_value to set a value on a repeatable property such as the "connection" wrapper in the example below. The -value option is mandatory in this case.

```
add_config_element /DftSpecification(top,rtl)/IjtagNetwork/
HostScanInterface(tap)/Tap(main)/HostIjtag(test_config)/
Tdr(test_config_tdr)/DataOutPorts/connection(0) -value PWR_MOD_INST/P0
```

See the get_attribute_value_list command for strategies for introspection of object types.

Arguments

- ***config_spec***

  A required value that specifies the name of a configuration element or a collection containing one configuration element. When ***config_spec*** is a collection, you cannot use the -in_wrapper option because the object contains all of the information about the configuration element. To use the -in_wrapper option, the ***config_spec*** value must be a name, and the name must be relative to the specified wrapper element.

  You can use a positional *id* for any level of hierarchy. You can also replace the id of a leaf name by the position.

  For example, the path "tmp(1)/<0>/wrp<1>" means "tmp(1)/wrp(abc)/wrp(def)" if "wrp(abc)" is the first element the tool finds inside "tmp(1)", and "wrp(def)" is the second instance of a wrapper with a base name of "wrp" inside "tmp(1)/wrp(abc)".

- -in_wrapper *wrapper_object_spec*

  An optional switch and value pair that specifies the parent wrapper in which the configuration element exists. The *wrapper_object_spec* value is the name of a wrapper or a collection containing a wrapper element. The name in ***config_spec*** is relative to the specified parent wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -id *position* | id_name | all

  An optional switch and value pair that specifies that the data to set is the *id* of a wrapper or of a repeatable property. When using -id all, the command expects a well-formatted Tcl list unless the configuration element has metadata with only one non-repeatable Id declared for it. In this case, the supply the value as a Tcl string, which is the case for most wrappers and repeatable properties you set.

  When you have configuration elements with more than one *id*, you can set one *id* at a time using either its position or, if it has metadata, the *id* of the Id wrapper that declares the *id*. A position is represented with an integer surrounded by <>. For example, <0> is the first id, <1> is the second, and so on.

- -value_id *position* | value_name | all

  An optional switch and value pair that specifies that the data to set is the value of a property, repeatable property, or is a row inside a DataWrapper or a CsvWrapper. If you do not specify the -id or -value_id options, "-value_id all" is the default.

  When using "-value_id all", the tool expects a well-formatted Tcl list unless the configuration element has metadata and there is only one non-repeatable value declared for it. In this case, the tool expects the value to be a Tcl string, which is the case for most properties you set.

  When you have configuration elements with more than one value, you can set one value at a time using either its position or, if it has metadata, the *id* of the Value wrapper that declares the value. A position is represented with an integer surrounded by <>. For example, <0> is the first value, <1> is the second, and so on.

- *values*

  A string or Tcl list of strings that defines the value where to set the Ids or Values of properties or wrappers. The tool also accepts collections where a value is expected and automatically extracts the name of the object, or list of object names.

  You use a list only when using "-id all" or "-value_id all" unless the configuration element has metadata and there is only one non-repeatable id or value declared for it; otherwise, you use a string.

- -specify_default

  A Boolean option that specifies that the property is to be set to its default value. You can only use this option on properties.

- -unspecify

  A Boolean option that specifies that the property is to be unspecified and thus return to its default value. You can only use this option on properties.

- -comment *comment* [-end_of_line]

  A switch and value pair that specifies the string comment for a configuration element and, optionally, where it appears. Each configuration element can have two types of comments simultaneously: regular and end of line. However, only regular comments can be

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

introspected, while the end of line comments cannot. When the configuration data is written out, the regular comments appear on a separate line before it. The end of line comment appears on the same line of the corresponding configuration data but after the data.

-end_of_line—When you use the optional -end_of_line option with the -comment argument, the tool tags the comment as an end of line comment. When the configuration data is written out, the end of line comment is written out after the corresponding configuration data but on the same line. Each configuration element can only have one end of line comment. The last end of line comment overrides the previous end of line comment.

Tip

To clear the end of line comment, set the end of line comment to an empty string.

```
set_config_value <config_object> -comment "" -end_of_line
```

- -partition *partition*

  An optional switch and value pair that specifies the partition you want to introspect.

  partition can be one of the following:

  tcd — When you specify the -partition option with the tcd value, the tool accesses the Core wrappers comprising the Tessent Core Description syntax.

  meta — When you specify the -partition option with the meta value, the tool accesses the metadata configuration data defining the syntax of the DftSpecification, DefaultsSpecification/DftSpecification, and others.

  meta:tcd — When you specify the -partition option with the meta:tcd value, the tool accesses the metadata configuration data defining the syntax of the Core wrappers and others.

  When you do not specify the -partition option, the tool accesses the Default partition, which is where the DftSpecification and DefaultsSpecification/DftSpecification configuration data is located.

Examples

Example 1

The following example uses a foreach loop to set the properties inside a wrapper

```
> set int [get_config_el Interface \
 -in /DftSpecification(corea,rtl)/IjtagNetwork/HostScanInterface(ijtag)]
> foreach {prop val} {scan_in mySI scan_out mySO tck myTCK} {
  set_config_val $prop -in $int $val
}
> report_config_data $int
Interface {
  tck : myTCK;
  scan_in : mySI;
  scan_out : mySO;
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

This example uses set_config_value to set a list of cell name in the DftSpecification/ force_creation_of_rtl_cells property:

```
> set spec [create_dft_spec]
> set_config_value force_creation_of_rtl_cells -in $spec {and2 or2 inv}
> report_config_data $spec -l 1
DftSpecification(ctrl_core,rtl) {
  IjtagNetwork {
    // Not shown
  }
  force_creation_of_rtl_cells : And2, Or2, Inv;
  MemoryBist {
    // Not shown
  }
}
> get_config_value force_creation_of_rtl_cells -in $spec
And2 Or2 Inv
```

Example 3

The following example shows that you must use the add_config_element command when trying to add a value to a repeatable property.

set_config_value /DftSpecification(top,rtl)/IjtagNetwork/HostScanInterface(tap)/Tap(main)/ HostIjtag(test_config)/Tdr(test_config_tdr)/DataOutPorts/connection(0) PWR_MOD_INST/P0

```
// Error: Path </DftSpecification(top,rtl)/IjtagNetwork/
HostScanInterface(tap)/Tap(main)/HostIjtag(test_config)/
Tdr(test_config_tdr)/DataOutPorts/connection(0)> is invalid.
```

add_config_element /DftSpecification(top,rtl)/IjtagNetwork/HostScanInterface(tap)/ Tap(main)/HostIjtag(test_config)/Tdr(test_config_tdr)/DataOutPorts/connection(0) -value PWR_MOD_INST/P0

Related Topics

get_config_value

get_defaults_value

report_config_data

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_contention_check

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether or not contention checking is on and the conditions under which checks are performed. Contention checking is set to On when the tool is invoked.

Usage

set_contention_check {{**ON** | **OFf** | **CAPture_clock**} [-Warning | -Error] [-Bus | -Port | -ALl] [-BIdi_retain | -BIDI_Mask] [-NO_FLoating {BIdi_only | ALl_busses}] [-ATpg | -CATpg] [-NOVerbose | -Verbose | -VVerbose]}

Description

When a bused output of a tri-state driver (or switch) that is driving an X caused by an X on its enable is encountered, the contention is not reported if the data input to the driver is at the same level as other drivers on the bus. Also, the simulated value of the bus gate is resolved to a binary value if it can do so without tracing through additional bus gates.

Arguments

- **ON**

  A literal that performs contention checking during simulation. The tool does not propagate captured data effects. This is the default.

  If the set_split_capture_cycle command is on and there are multiple capture clocks in the design, the set_contention_check ON command performs a contention check not only before the capture but after the capture as well.

- **OFf**

  A literal that specifies to not perform contention checking during simulation.

  Tip

  If you create patterns using ATPG Expert (“create_patterns”), the tool automatically determines whether or not the -Atpg switch is necessary and sets it if simulation determines that too many patterns are being rejected due to contention; therefore, you do not need to specify “set_contention_check on -atpg”. This may, in some cases, be faster than setting it initially if a lot of fault coverage can be obtained using ATPG without it. If you are not sure if the switch is needed, it may be better to try running without it.

- **Capture_clock**

  A literal that performs contention checking both with and without propagating captured data effects. If a clock, read control, or write control line connects to a bus, the tool also performs bus contention checking with all clocks off prior to the application of the capture clock. The tool does not consider any contention patterns for fault simulation and does not place any of these patterns into the internal test pattern set.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tip

If you create patterns using ATPG Expert ("create_patterns"), the tool automatically determines if the -Catpg switch is necessary and uses it only when it is needed; therefore, all you need to specify is "set_contention_check capture_clock". This may, in some cases, be faster than setting it initially if a lot of fault coverage can be obtained using ATPG without it. If you are not sure if the switch is needed, it may be better to try running without it.

- -Warning

  An optional switch that displays a warning message but continues simulation if bus contention occurs during simulation. The warning message indicates the number of patterns the tool rejected in the current simulation pass of 64 patterns and also identifies the bus gate on which the bus contention occurred.

- -Error

  An optional switch that displays an error message and stops the simulation if bus contention occurs.

  You can debug contention errors by using the -Error switch to stop the simulation at the point of the first contention error.

  Using this option, you can then view the simulated values of all gates in the first bus contention pattern by using the report_gates command.

  The error message indicates the number of patterns the tool rejected in the current simulation pass of 64 patterns and also identifies the bus gate on which the bus contention occurred.

- -Bus

  An optional switch that performs contention checking of tri-state driver buses. This is the default.

  Tri-state logic enables several bus drivers to time-share a bus. However, if the circuit enables two bus drivers of opposite logic to drive the bus, physical damage can occur. This switch enables the tool to identify these conditions and notify you of their existence.

  The tool identifies buses with circuitry that prevents bus contention and does not check for bus contention problems. This eliminates false bus contention reporting when multiple inputs to a bus are at X. Bus contention that occurs on weak buses does not result in an E4 rules checking violation or pattern rejection during simulation. The tool continues to simulate them as an X state.

- -Port

  An optional switch that performs contention checking for multiple-port flip-flops and latches. The tool identifies and rejects patterns during which any multiple-port latch or flip-flop has more than one clock, set, or reset input active (or at X).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -BIDI_Retain

  An optional switch that rejects patterns during contention checking that cause the direction of bidirectional pins to change following the capture clock.

- -BIDI_Mask

  An optional switch that causes the fault simulator to modify the input values of bidi pins to avoid bus contention after the capture clock if a device pin changes from input mode to output mode as the clock is applied.

- -NO_FLoating {BIdi_only | ALl_busses}

  An optional switch and literal pair that checks for floating buses in addition to contention when bus contention checking is in effect. The Bidi_only option only checks for floating at buses associated with bidirectional pins. The All_busses option checks for floating at all buses. This switch enables the E11 DRC with handling set to warning if it was previously set to ignore. (If you have previously set E11 handling to note or error, your setting is retained).

  The floating bus checks are always performed during pattern simulation when bus contention checking is in effect. If -Atpg or -Catpg is included, floating buses are avoided during test generation.

  The following rules apply:

  - Floating checks are performed for strong buses only, not weak buses.
  - If a strong bus passes the E11 DRC, the switch does not apply to that bus.
  - If a strong bus drives a pull bus, the strong bus can float if the weak input of the pull bus is set to a known binary value.
  - This switch and the -Bidi_mask switch cannot be used together in the same command.
  - This switch does not require that a bus have a known binary value to pass the floating check. The bus may be driven to an unknown binary value.
  - If a bus is driven by a TIEX gate directly, the TIEX gate is treated as a source that can take on a Z value. As a result, it is impossible to avoid floating this bus, and ATPG cannot generate any test patterns.

- -ALl

  An optional switch that performs contention checking for both tri-state driver buses and multiple-port flip-flops and latches.

- -ATpg

  An optional switch that forces all buses to a non-contention state, which ensures that test patterns are not created causing bus contention.

  After completing normal test pattern generation for a fault, the tool forces all buses to a non-contention state. If the tool cannot satisfy this condition, given the conditions set by the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

original pattern, the tool aborts the fault, excludes the pattern from the final test set, and displays a message indicating the number of these aborted faults for each simulation pass. No attempt is made to change the original pattern.

The -Atpg option results in the additional effort by the test pattern generator, and you should use it only when necessary.

Tip

If you create patterns using ATPG Expert ("create_patterns"), you may get the best results by not explicitly specifying the -Atpg switch. ATPG Expert automatically determines if the switch is necessary and utilizes it only when it is needed.

- -CATpg

  An optional switch that uses clock sequential ATPG effort in the capture cycle as well as all other cycles. If neither this switch nor the -Atpg switch is specified, the default continues to be that there is no ATPG effort for bus contention in any cycle, and ATPG constraints are satisfied deterministically prior to the capture clock event only. Specifying the -Catpg switch without the Capture_clock option is equivalent to using just the -Atpg switch.

  Tip

  If you create patterns using ATPG Expert ("create_patterns"), you may get the best results by not explicitly specifying the -Catpg switch. ATPG Expert automatically determines if the switch is necessary and utilizes it only when it is needed.

- -Verbose

  An optional switch that reports the first reason for each pattern rejected (maximum of 64 messages per parallel pattern invocation).

- -VVerbose

  An optional switch that reports every reason for each pattern rejected (there is no limit to the number of messages per parallel pattern invocation). The messages are of the same type issued for the -Verbose switch.

  Note

  For large designs, this option may produce thousands of lines of output for each pattern simulated.

- -NOVerbose

  An optional switch that lets you turn off the -Verbose or -VVerbose effects that may have been set previously.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example performs contention checking on multiple-port flip-flops and latches, stops the simulation if any bus contention occurs, and displays an error message. The message indicates the number of patterns rejected and the bus gate on which the bus contention occurred:

```
set_contention_check on -port -error
set_system_mode analysis
create_patterns
```

Example 2

The following example performs contention checking on both multiple-port sequential gates and tri-state buses, stops the simulation if any bus contention occurs, and displays an error message that indicates the gate on which the contention occurred:

```
set_system_mode analysis
set_contention_check on -all
create_patterns
```

Related Topics

report_gates

set_bus_handling

set_gate_report

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_context

Context: unspecified, all contexts

Mode: all modes

Specifies the current usage context of Tessent Shell.

Usage

set_context **dft** [{-edt [-logic_bist]} | -scan | -test_points | {-scan -test_points}] [-rtl | -no_rtl] [-license *license_name*] [-design_identifier *string*] [-force]

set_context **patterns** [-ijtag | -scan | -scan_retargeting | -scan_diagnosis | -failure_mapping] [-silicon_insight] [-rtl | -no_rtl] [-license *license_name*] [-design_identifier *string*] [-force]

Description

You must set the context before you can invoke most other commands in Tessent Shell.

If you attempt to change context and have generated patterns, the tool prompts you to save the patterns with an error message similar to the following:

```
//  Error: Current test pattern set has not been saved. Save the
//  patterns or use the -force switch.
```

Arguments

- **dft**

  A required literal that sets the context to dft. In the basic dft context, you load your design in setup mode and then move to the insertion mode to execute design editing commands such as create_connections, create_instance, and move_connections. When used with the -scan or -edt options, the dft context is used to insert scan chains or insert the EDT controller.

  If your intent is to insert only an IJTAG network at your current design level, but this design level, including all non-physical child blocks, does not contain any Tessent signed instruments (MemoryBIST, LogicBIST, TestKompess, MissionMode), then you should select any one of the following licenses: IJTAG, MemoryBIST-LV, MemoryBIST-LVNS, MemoryBIST-TS, LogicBIST, TestKompress, or MissionMode. Otherwise, the tool chooses for you, in addition to whatever license you may have selected with the -license option.

- **patterns**

  A required literal that sets the context to patterns which provides functionality related to scan and IJTAG pattern generation, ICL extraction, and scan pattern diagnosis.

- -scan

  An optional switch that is available in both patterns and dft contexts. This sub-context specifies ATPG and simulation functionality for standard ATPG, Tessent TestKompress,

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

and LogicBIST patterns when specified with the patterns context, and for scan insertion when specified with the dft context.

- -scan_diagnosis

  An optional switch that is available only in the patterns context. This sub-context enables the Diagnosis capabilities. For more information, see the *Tessent Diagnosis User's Manual*.

  You can only read a flat model that was saved in the analysis mode in this context. If you read a netlist or a flat model that was saved in the setup mode, you cannot switch from setup to analysis mode.

- -edt

  An optional switch that is available in dft context. This sub-context specifies to create and insert EDT IP into the design. When you provide a scan-inserted netlist, the tool analyzes the scan chains and generates and inserts an appropriate EDT controller.

- -logic_bist

  An optional switch that is available in the dft context and must be specified in conjunction with the -edt switch. This switch enables the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

- -test_points

  An optional switch that is available in the dft context and specifies functionality for test point identification and insertion.

- -ijtag

  An optional switch that is available only in patterns context. This sub-context specifies functionality related to the IEEE 1687-2014 (IJTAG) standard such as generating IJTAG patterns by executing PDL commands such that they are retargeted to the current design boundary, and performing ICL extraction. When the set_current_design command is executed, the presence of an ICL module matching the specified current design name sets the context to ijtag retargeting. If there is no ICL module matching the specified current design name, the ICL extraction mode is enabled. Executing "get_context -extraction" returns 1 when in the extraction mode.

- -scan_retargeting

  An optional switch that is available in the patterns context. This sub-context enables scan pattern retargeting functionality for retargeting core-level test patterns at the top level. For more information, see "Scan Pattern Retargeting" in the *Tessent Scan and ATPG User's Manual*.

- -failure_mapping

  An optional switch that is available only in the patterns context. This sub-context specifies to reverse map top-level failure files for retargeted scan pattern results, which were generated from within the -scan_retargeting sub-context. Reverse mapping results in a set of core-level failure files that you can diagnose with Tessent Diagnosis. For more information, refer to the *Tessent Diagnosis User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -silicon_insight

  An optional switch that is available only in the patterns context. This sub-context enables the SiliconInsight capabilities. For more information, see the *Tessent SiliconInsight User's Manual for Tessent Shell*.

- -rtl | -no_rtl

  In dft context, these switches specify whether you are performing netlist editing or RTL editing. You must use the -rtl option if you are editing RTL. You can use either option when editing a netlist but be aware that the write_design usage model is different between these modes. If you want to write out your edited netlist into a single file, select the -no_rtl option.

  Note

  By default, the tool enables preserve escaping in RTL contexts and disables the escaping in non-RTL contexts. Switching between -rtl and -no_rtl contexts in the same tool invocation enables and disables escaping automatically.

  In the patterns -ijtag context, the -rtl switch is inferred if coming from the dft context and the -rtl option was used when setting the dft context. When entering the patterns -ijtag context from the unspecified context, -no_rtl is assumed. ICL Extraction makes use of quick synthesis to convert any RTL in the fan-in or fanout of ICL modules. See the description of the synthesize_before_analysis and exclude_from_synthesis Module Built-in Attributes if you are using a test_setup procedure for ICL extraction.

  In all other contexts, the -rtl switch cannot be used and the -no_rtl switch is optional because it is always inferred.

  In patterns -ijtag context with RTL support, during ICL extraction the input cones in front of design ports with matched ICL attributes are synthesized and flattened in order to perform extraction of the paths and end points, which results in an extracted ICL top level description. Checking and debugging of trace issues, IJTAG-related checks and ICL semantic checks are also done at this stage. The same is true for output cones driven by design ports with matched ICL attributes.

- -license *license_name*

  An optional switch and string pair that specifies a particular license to use. By default, Tessent Shell acquires the least expensive license available that provides the requested functionality. License names for MemoryBIST and BoundaryScan are accepted without the -LV, -LVNS, or -TS suffix and defaults to the LV version of the license.

  The TESSENT_LICENSE_ORDER environment variable modifies the priority and order of the available licenses. Any available licenses not explicitly listed in the value of the TESSENT_LICENSE_ORDER environment variable are appended to the list in their original order.

  Table 8-4 lists the supported license names for the available contexts and sub-contexts.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-4. Contexts Available by License

Context/Sub-Context	License Name(s)
dft -rtl dft -no_rtl	DesignEditor BoundaryScan-LV BoundaryScan-LVNS BoundaryScan-TS ScanPro IJTAG PLLTest MemoryBIST-LV MemoryBIST-LVNS MemoryBIST-TS SerdesTest TestKompress LogicBIST MissionMode
dft -edt	TestKompress
dft -logic_bist -edt	LogicBIST
dft -scan	Scan ScanPro
dft -test_points	ScanPro LogicBIST

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-4. Contexts Available by License (cont.)

Context/Sub-Context	License Name(s)
patterns -ijtag	IJTAG BoundaryScan-LV BoundaryScan-LVNS BoundaryScan-TS ScanPro PLLTest MemoryBIST-LV MemoryBIST-LVNS MemoryBIST-TS SerdesTest TestKompress LogicBIST MissionMode
patterns -scan	When EDT is used: TestKompress When EDT and LBIST are not used: TestKompress, FastScan When LBIST is used: LogicBIST
patterns -scan_retargeting	TestKompress
patterns -failure_mapping	Diagnosis
patterns -scan_diagnosis	Diagnosis
patterns -silicon_insight	SiliconInsight

If Tessent Shell is unable to acquire a license when setting the context, the tool returns an error message.

- -design_identifier *design_id*

  An option used to assign an identifier to the current design.

  A design_id must only use a combination of the following characters:

  - Alphabetic, both uppercase and lowercase. For example: “abc”, “ABC”, “AbC”, and so on.
  - Numeric. For example: “123”.
  - Underscore. For example: “A_1”.

  For dft context, if this option is unspecified, the identifier defaults to “rtl” in the -rtl context, and to “gate” in the -no_rtl context. If you use the -design_identifier switch, then you must specify a *design_id* value. In the case of an empty string, the design identifier falls back to

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

default value, which is gate or rtl in dft context or an empty string in the case of patterns context.

For patterns context, if this option is unspecified, the identifier defaults to either the value you previously set using set_context or to a null string if it was never set.

You only need to specify this option in the dft context when you are performing a second insertion pass on the same design. When you use the process_dft_specification command to insert all instruments in one insertion pass, the default value is adequate. In the dft context, the tool generates an error if module names matching *current_design_design_id_tessent_** are found already instantiated into the design during transition to analysis mode or when running the process_dft_specification command; this indicates that you are performing a second insertion pass into that same design and that a new design identifier value needs to be specified.

- -force

  An optional switch used to force Tessent Shell into the specified context.

Examples

Example 1

The following example demonstrates setting the context to dft. In this case, one of the dft licenses is checked out if one is not already checked out.

```
set_context dft -no_rtl
```

Example 2

The following example demonstrates setting the context to patterns -scan and requesting a TestKompress license. If the -license option is not specified, a FastScan license is first checked out if one is available. After that, a check is performed to determine whether LBIST or EDT modes are used; if they are, the Tessent FastScan feature is released and either the Tessent TestKompress or Tessent LBIST license is checked out, as appropriate.

```
set_context patterns -scan -license TestKompress
```

Related Topics

extract_icl

get_context

report_context

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_core_instance_parameters

Context: patterns -scan

Mode: setup, analysis

Changes the parameters of instruments previously added with the add_core_instances command.

Usage

set_core_instance_parameters
{**-instrument_type** {**edt** | **occ** | **lpct** | **lbist** | **ssh**} |
-cores ***core_names*** |
-modules ***module_objects*** |
-instances ***instance_objects*** |
-icl_modules ***icl_module_objects*** |
-icl_instances ***icl_instance_objects***}
[-parameter_values *parameter_list*]
[-silent]

Description

The instruments that the tool has extracted (that is, that are part of a core instance) have a limited set of parameters that the tool can override. Use this command to change these parameters.

Use the report_core_instance_parameters command to report the instrument configuration parameters and their values that were specified when the core instance was added with the -parameter_values switch of the add_core_instances command.

Note

The report_core_instance_parameters command lists only the instruments whose parameters can be modified. Use the report_core_parameters command to show every instrument and the parameters available if an instrument's parameters cannot be modified.

Arguments

- **-instrument_type edt | occ | lpct | lbist | ssh**

  A required switch and literal pair that changes all instances of a given type. Choose one from the following:

  edt

  occ

  lpct

  lbist

  ssh

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-cores *core_names***

  A required switch and string pair that changes the specified instrument parameters for the specified ***core_names***, which is a Tcl list of one or more core names, or a collection of one or more core names.

- **-modules *module_objects***

  A required switch and string pair that changes the specified instrument parameters for the specified ***module_objects***, which is a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects.

- **-instances *instance_objects***

  A required switch and string pair that changes the specified instrument parameters for the specified ***instance_objects***, which is a Tcl list of one or more object names (hierarchical names), or a collection of one or more objects.

- **-icl_modules *icl_module_objects***

  A required switch and string pair that changes the specified instrument parameters for the specified ***icl_module_objects***, which is a Tcl list of one or more object names (hierarchical names) or a collection of one or more objects.

- **-icl_instances *icl_instance_objects***

  A required switch and string pair that changes the specified instrument parameters for the specified ***icl_instance_objects***, which is a Tcl list of one or more object names (hierarchical names) or a collection of one or more objects.

- -parameter_values *parameter_list*

  An optional switch and list that specifies the instrument configuration parameter values to change. Table 8-5 shows the list of available instrument parameters by instrument type.

Table 8-5. Core Instance Parameters and Values by Instrument

Parameter	Value	Description
Instrument Type: EDT		
edt_bypass	on | off The default is off.	Defines whether the EDT is used in bypass mode. See "Compression Bypass Logic" for more information on the parameter and bypass in general.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-5. Core Instance Parameters and Values by Instrument (cont.)

Parameter	Value	Description
edt_configuration	A string that is the name of the configuration specified during IP creation. When used with the DftSpecification flow and the HighCompressionConfiguration wrapper, there are only two valid values: low_compression | high_compression The default is low_compression.	Defines which compression configuration is used by the EDT. See "Dual Compression Configurations" in the *Tessent TestKompress User's Manual* for more information on compression configurations.
edt_low_power_shift_en	on | off The default value is based on the EDT IP generation time user-defined power controller status.	Defines whether the EDT is used in low power shift mode. See the "Power Controller Logic" section in the *Tessent TestKompress User's Manual* for more information on edt_low_power_shift_en.
edt_single_bypass_chain	on | off The default is off.	Defines whether the EDT is used in single bypass mode. See "Dual Bypass Configurations" in the*Tessent TestKompress User's Manual* for more information on edt_single_bypass_chain.
lbist_low_power_shift_en	on | off The default is off.	Defines whether LBIST is used in low power shift mode.
used_input_channels	integer The default value is the maximum available input channels.	Defines how many input channels should be used by the EDT. See the description for used_input_channels in the set_edt_options command arguments in the *Tessent Shell Reference Manual*.
tessent_chain_masking	on | off The default is off.	Defines if Hybrid TK/LBIST chain masking should be used. See "Pattern Generation with Internal Chain Masking Hardware" in the *Tessent TestKompress User's Manual* for more information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-5. Core Instance Parameters and Values by Instrument (cont.)

Parameter	Value	Description
Instrument Type: OCC		
active_upstream_parent_occ	on | off The default is off.	Enables an active upstream parent OCC driving the fast clock input. For information, see "OCC" on page 4356.
capture_window_size *integer*	integer The default value is the maximum size supported by the OCC instance. For transition patterns, the default value is 2.	Specifies the maximum number of clock pulses during capture cycle. This value must not exceed the registers created during IP creation. By default, the tool creates an OCC able to pulse up to three times between scan loads, but you may not need to use them all. Set this parameter to a value smaller than default to reduce the number of OCC cells and therefore the number of bits that need to be encoded during pattern generation. When the OCC chains are driven by EDT, you free up encoding capacity to be used for the generated tests and may reduce pattern count.
clock_sequence *integer*	integer The default is 0.	Defines the clock sequence when you have set static_clock_control to internal. The integer value directly specifies the clock sequence. For example, for a 3-bit OCC, specifying "1" (binary "001") means a clock pulses in the first cycle. Specifying "5" (binary 101) means clock pulses in the first and third cycles.
fast_capture_mode	on | off The default is off.	Defines whether the fast capture clock is used during capture. See "Operating Modes" in the *Tessent Scan and ATPG User's Manual* for information on this parameter.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-5. Core Instance Parameters and Values by Instrument (cont.)

Parameter	Value	Description
parent_mode	on | off The default is off.	Defines whether the OCC is used in parent mode. When the default off is used, the OCC is used in standard mode. See "Parent-Mode Operation" section in the *Tessent Scan and ATPG User's Manual* for more information on this parameter.
static_clock_control	off | internal | external The default is off.	Defines whether the OCC is in static control mode. When the default off is used, the OCC is dynamically programmed on a per ATPG pattern basis. During ATPG this parameter can only be set to off.
Instrument Type: LBIST		
async_set_reset_pattern_count	integer The default is 0.	Defines the number of asynchronous set/reset patterns the tool applies after any chain test patterns and before the regular scan patterns. This parameter is present only if the LBIST controller has the asynchronous set/reset pattern hardware.
chain_test_pattern_count	integer The default is 0.	Defines the number of chain test patterns the tool applies before the asynchronous set/reset patterns and the regular scan patterns. This parameter is present only if the LBIST controller has the chain test pattern hardware.
pattern_set	scan | chain The default is scan.	Defines whether the pattern set is scan or chain.
named_capture_procedure_timeplate	string The default is the empty string.	Specifies the name of the timeplate used by LBIST named capture procedures. If this parameter is not defined, the first valid timeplate is used.
Instrument Type: LPCT		

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-5. Core Instance Parameters and Values by Instrument (cont.)

Parameter	Value	Description
reset_control	on | off The default is off.	Controls the value being loaded into the LPCT Type-3 reset control cell. See "LPCT Controller-Generated Scan Enable" in the *Tessent TestKompress User's Manual*.
scan_en_control	on | off The default is off.	Controls the value being loaded into the LPCT Type-3 scan enable control cell. See "LPCT Controller-Generated Scan Enable" in the *Tessent TestKompress User's Manual*.
Instrument Type: SSH		
capture_packets	integer The default is the maximum value supported by the hardware, not exceeding 7.	Defines the number of capture clock pulses generated by the ScanHost node in the internal capture mode.
on_chip_compare_enable	on | off The default is off.	Controls whether or not the ScanHost performs on-chip compare when equipped with on-chip compare hardware. See OnChipCompareMode/present in ScanHost.
on_chip_compare_group	integer The default is 1.	Controls whether or not the ScanHost instance contributes to a status group. See the Note below this table for details. This property is valid only when the on_chip_compare_enable property is set to on.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-5. Core Instance Parameters and Values by Instrument (cont.)

Parameter	Value	Description
sticky_status_global	0 | 1 The default is 1.	Indicates the status of the ScanHost pass/fail sticky bit when equipped with on-chip compare hardware. This property is valid only when the on_chip_compare_enable property is set to on. This property is also valid when The OnChipCompareMode/ sticky_bit_resolution property of the DFT Specification was set to output_chain or ssh when the hardware was generated. See ScanHost.
sticky_status_per_chain_output	0 | 1 The default is 1.	Indicates the status of the ScanHost chain output pass/fail sticky bits when equipped with on-chip compare hardware. This property is valid only when the on_chip_compare_enable property is set to on. The OnChipCompareMode/ sticky_bit_resolution property of the DFT Specification must also have been set to output_chain when the hardware was generated. See ScanHost.

Note

See Figure 13-109 and the preceding text in ScanHost for more details on status groups. By default, all instances of the same core type contribute to the same status group. Diagnosis is supported in this case. Valid values for on_chip_compare_group are:

- 0 — The ScanHost instance does not contribute to any status group. The pass/fail result for that ScanHost instance is only available via the IJTAG sticky bit unloaded

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

at the end of the test. Diagnosis is not supported when the on_chip_compare_group property is set to 0. It is not recommended to use this setting.

- *N* — The ScanHost instance will contribute to the Nth status group, where *N* is an integer greater than 0 and less than the number specified by OnChipCompareMode/status_groups. The default is 1.
- If there is more than one group, the groups must begin with 1 (default) and increment sequentially from there.

- [-silent]

  An optional switch that specifies to suppress warning messages.

Examples

Use the following command to enable on-chip compare on all ScanHost instances in the design:

```
set_core_instance_parameters -instrument_type ssh -parameter_values
{on_chip_compare_enable on}
```

Related Topics

add_core_instances

import_scan_mode

report_core_instance_parameters

report_core_parameters

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_critical_area_options

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables you to specify options for the critical area handling.

Usage

set_critical_area_options [-reporting {on | off}]

Description

You can use critical area coverage values along with the traditional fault coverage values for a full chip coverage measurement. Each UDFM file generated by Tessent CellModelGen or the extract_fault_sites command contains a total critical area (TCA) value per fault. This command enables you to specify critical area handling options for pattern generation and diagnosis.

Note

Using this command requires an Automotive-Grade ATPG license feature. See "Feature Entry in the License File" in the *Managing Tessent Software* manual.

Arguments

- -reporting {on | off}

  An optional switch and literal pair that activates the reporting of the TCA coverage. The default is off.

  Activating this switch reports additional critical area-related information, affecting the following commands:

Command	TCA Reporting
report_statistics	Displays additional columns listing the final TCA values per fault category and the final TCA coverage numbers
report_udfm_statistics	Displays a TCA and a TCA [%] column for each row
write_faults	Displays a TCA column for each fault
report_faults	Displays a TCA column for each fault
create_patterns	Additionally displays a cumulative TCA test coverage value in the statistics transcription for each batch of patterns simulated
identify_redundant_faults	Displays an additional column listing the incremental TCA test coverage

Examples

Example 1

The following example shows the log file output when using

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_critical_area_options -reporting on

in conjunction with

report_statistics

```
                                Statistics Report
                                UDFM Static Faults
--------------------------------------------------------------------------------
Fault Classes                     #faults           #faults              TCA
                                   (total)      (total relevant)        (total)
----------------------------  --------------- ------------------  -----------------
  FU (full)                        241131           240347             1627952
  --------------------------  --------------- ------------------  -----------------
  UC (uncontrolled)                381 ( 0.16%)      same  ( 0.16%)     1786 ( 0.11%)
  UO (unobserved)                 1500 ( 0.62%)      same  ( 0.62%)     7829 ( 0.48%)
  DS (det_simulation)           210569 (87.33%)      same  (87.61%)  1391149 (85.45%)
  DI (det_implication)            7658 ( 3.18%)      same  ( 3.19%)    88641 ( 5.44%)
  PU (posdet_untestable)           729 ( 0.30%)      same  ( 0.30%)     7140 ( 0.44%)
  PT (posdet_testable)            1241 ( 0.51%)      same  ( 0.52%)     8053 ( 0.49%)
  UU (unused)                      925 ( 0.38%)      same  ( 0.38%)    10692 ( 0.66%)
  TI (tied)                        390 ( 0.16%)      same  ( 0.16%)     2688 ( 0.17%)
  BL (blocked)                      12 ( 0.00%)      same  ( 0.00%)       52 ( 0.00%)
  RE (redundant)                  6607 ( 2.74%)      same  ( 2.75%)    23859 ( 1.47%)
  AU (atpg_untestable)           11119 ( 4.61%)     10335  ( 4.30%)    86063 ( 5.29%)
--------------------------------------------------------------------------------
Fault Sub-classes
  --------------------------
  AU (atpg_untestable)
    UDN   (undriven)               23 ( 0.01%)      same  ( 0.01%)      159 ( 0.01%)
    BB    (black_boxes)          1132 ( 0.47%)      same  ( 0.47%)    14312 ( 0.88%)
    EDT   (edt_blocks)            784 ( 0.33%)        deleted          4334 ( 0.27%)
    PC*   (pin_constraints)         8 ( 0.00%)      same  ( 0.00%)      104 ( 0.01%)
    TC*   (tied_cells)           3866 ( 1.60%)      same  ( 1.61%)    25289 ( 1.55%)
    SEQ   (sequential_depth)      977 ( 0.41%)      same  ( 0.41%)    12586 ( 0.77%)
    Unclassified                 4329 ( 1.80%)      same  ( 1.80%)    29277 ( 1.80%)
  UC+UO
    AAB   (atpg_abort)           1051 ( 0.44%)      same  ( 0.44%)     5134 ( 0.32%)
    Unclassified                  830 ( 0.34%)      same  ( 0.35%)     4481 ( 0.28%)
  *Use "report_statistics -detailed_analysis" for details.
--------------------------------------------------------------------------------
Coverage
  --------------------------
  test_coverage                        94.00%              94.32%            93.51%
  fault_coverage                       90.91%              91.21%            91.37%
  atpg_effectiveness                   98.96%              98.96%            99.16%
--------------------------------------------------------------------------------
#test_patterns                                                                 920
  #basic_patterns                                                              174
  #clock_sequential_patterns                                                   406
  #multiple_load_patterns                                                      340
#simulated_patterns                                                            993
CPU_time (secs)                                                               1120.9
--------------------------------------------------------------------------------
```

Example 2

The following example shows the log file output when using

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_critical_area_options -reporting on

in conjunction with

report_udfm_statistics

```
//  UDFM Fault Statistics
//
//  Type                  Block     Name   Instances  Port-Insts    Faults       TCA[ts]  DS[%]   FC[%]  TC[%]  TCA[%]
//  --------------------  --------  -----  ---------  ----------  --------  ------------  -----   -----  -----  ------
//  auto-inserted-faults  Instance  *              1           1        70          0.00  21.43   24.29  25.00    0.00
//  inter_cell_bridges    Instance  *              1           0    263158     251824.39  91.59   91.66  97.69   97.75
//  interconnect_bridges  Instance  *              1           0   9799895   42306688.00  82.03   83.80  89.08   85.22
//  interconnect_opens    Instance  *              1           1   1499930   33329146.00  74.67   96.83  97.91   96.32
//  intra_cell_defects    Module    *         197986        8975  12701231   80161624.00  93.17   95.88  97.34   97.39
//  undefined-cells       Cell      *           1151        1147     15338          0.00  49.84   81.80  92.91    0.00
//                                  Total     199141       10124  24279622  156049120.00  87.48   91.01  94.13   93.86
```

Note

Some columns are omitted for clarity.

Example 3

The following example shows the generated fault file (MTFI) when using

set_critical_area_options -reporting on

in conjunction with

write_faults

```
FaultInformation {
 version : 1;
 FaultType (UDFM) {
  FaultList {
   Format : Identifier, Class, TCA;
   UdfmType ("udfm_top") {
   Cell ("topcAND2A") {
    Instance ("gatesA/gate1") {
     "sa0_A", AU, 10.00;
     "sa0_B", DS, 10.00;
     "sa0_Z", DS, 10.00;
     "sa1_A", DS, 10.00;
     "sa1_B", DS, 10.00;
     "sa1_Z", DS, 10.00;
    }
    Instance ("gatesA/gate2") {
     "sa0_A", AU, 10.00;
     "sa0_B", DS, 10.00;
     "sa0_Z", DS, 10.00;
     "sa1_A", DS, 10.00;
     "sa1_B", DS, 10.00;
     "sa1_Z", DS, 10.00;
    }
...
```

Example 4

The following example shows the log file output when using

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_critical_area_options -reporting on

in conjunction with

report_faults

```
//    command: report_faults -bridge -net1 clk_half_L7_N0
//  Class  UdfmType              Fault        TCA     NetA                                   NetB
//  -----  --------------------  -----------  ------  --------------  -----------------  --------------
//  AU     interconnect_bridges  B1_B0_dom_A  522.83  u_edt_wrapper/  /edt_scan_in[38]   clk_half_L7_N0
//  AU     interconnect_bridges  B1_B1_dom_A  522.83  u_edt_wrapper/  /edt_scan_in[38]   clk_half_L7_N0
```

Note

The NetA column above is truncated for clarity.

Example 5

The following example shows the log file output when using

set_critical_area_options -reporting on

in conjunction with

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_patterns

```
//  Simulation performed for #gates = 395716  #faults = 223481
//  system mode = analysis    pattern source = internal patterns
//  ---------------------------------------------------------------------
//  #patterns  test      #faults  #faults    # eff.    # test    TCA test       process      RE/AU/AAB/EAB
//  simulated  coverage  in list  detected   patterns  patterns  coverage       CPU time
//  deterministic ATPG invoked with comb/seq abort limit = 200/60
//   ---       ------     ---      ---       ---       ---       ------       34.01 sec    53/8783/142/0
//   64        73.65%    55984    167420      64        64       74.16%      187.17 sec
//   ---       ------     ---      ---       ---       ---       ------      214.25 sec    228/8851/291/0
//   128       82.36%    35174     20548      64       128       83.50%      235.34 sec
//   ---       ------     ---      ---       ---       ---       ------      275.16 sec    708/9107/521/0
//   192       86.37%    25088      9292      64       192       87.18%      281.68 sec
//   ---       ------     ---      ---       ---       ---       ------      321.95 sec    1484/9482/825/0
//   256       88.57%    19167      4652      64       256       89.09%      326.51 sec
//   ---       ------     ---      ---       ---       ---       ------      356.21 sec    2226/9884/1081/0
//   320       89.97%    15152      2782      64       320       90.31%      361.23 sec
//   ---       ------     ---      ---       ---       ---       ------      395.24 sec    3066/10234/1300/0
//   384       90.95%    12278      1588      64       384       91.03%      398.27 sec
//   ---       ------     ---      ---       ---       ---       ------      428.03 sec    3981/10588/1563/0
//   448       91.79%     9642      1246      64       448       91.78%      432.07 sec
//   ---       ------     ---      ---       ---       ---       ------      464.63 sec    5031/11034/1879/0
//   512       92.57%     7034       969      64       512       92.42%      468.52 sec
//   ---       ------     ---      ---       ---       ---       ------      512.11 sec    6176/11492/2197/0
//   576       93.29%     4602       694      64       576       92.88%      515.24 sec
//   ---       ------     ---      ---       ---       ---       ------      522.09 sec    6221/11504/2213/0
//   640       93.46%     4141       402      63       639       93.02%      525.31 sec
//   ---       ------     ---      ---       ---       ---       ------      527.31 sec    6221/11504/2213/0
//   704       93.54%     3953       188      60       699       93.11%      529.74 sec
//   ---       ------     ---      ---       ---       ---       ------      531.58 sec    6221/11504/2213/0
//   768       93.61%     3745       208      60       759       93.17%      534.17 sec
//   ---       ------     ---      ---       ---       ---       ------      536.14 sec    6221/11504/2213/0
//   832       93.67%     3592       153      59       818       93.24%      538.44 sec
//   ---       ------     ---      ---       ---       ---       ------      538.65 sec    6221/11504/2213/0
//   895       93.72%     3422       170      57       875       93.31%      540.97 sec
//  73 faults were identified as detected by implication.
//   ---       ------     ---      ---       ---       ---       ------      552.70 sec    6221/11504/1666/0
//   953       93.83%     3158       222      34       909       93.39%      554.23 sec
//   ---       ------     ---      ---       ---       ---       ------      554.66 sec    6221/11504/1666/0
//   988       93.84%     3134        24       7       916       93.40%      556.55 sec
//   ---       ------     ---      ---       ---       ---       ------      559.59 sec    6221/11504/1664/0
//   992       93.85%     3125         9       3       919       93.40%      559.93 sec
//   ---       ------     ---      ---       ---       ---       ------      562.78 sec    6221/11504/1664/0
//   993       93.85%     3123         2       1       920       93.40%      563.04 sec
//   ---       ------     ---      ---       ---       ---       ------      565.28 sec    6221/11504/1664/0
//  EDT with 1-hot scan masking.
//   ---       ------     ---      ---       ---       ---       ------      567.36 sec    6221/11504/1664/0
```

Example 6

The following example shows the log file output when using

set_critical_area_options -reporting on

in conjunction with

identify_redundant_faults

```
//  ---------------------------------------------------------------------
//  Performing redundant fault identification for 10121 faults
//  ---------------------------------------------------------------------
//  deterministic ATPG invoked with abort limit = 200
//  # red.    # non-red.   # abort     # remn.   progress   test      TCA test    process
//  faults      faults      faults      faults               coverage  coverage    CPU time
//     155        7019         63        2884    71.50%    93.91%    93.46%       5.14 sec
//     353        9276        189         303    97.01%    93.99%    93.50%      10.34 sec
//     386        9530        205           0   100.00%    94.00%    93.51%      10.99 sec
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_current_design

Context: all contexts

Mode: setup, insertion

Specifies the top level of the design for all subsequent commands until reset by another run of this command.

Usage

set_current_design [*top_design* [-icl_module *icl_module*]] [-verbose] [-in_library *library*] [-show_elaboration_warnings] [-parameter_values *parameter_list*] [-no_icl_module_matching]

Description

When you issue this command without an argument, the tool automatically finds the top level of the design if only one exists. If the tool finds more than one top, the tool presents a list of the possible top levels; you must reissue the command and specify one of the top levels listed. Note, if you issue this command without an argument and a design is already elaborated, the same design is systematically re-elaborated.

If you want to elaborate a parameterized design using non-default parameters, use the -parameter_values option. If you elaborate a parameterized design without using that option, the tool uses the default parameter values. In such a case, parameterized views of modules are available to introspection only for parameterized modules instantiated below the current design.

If you want to read additional designs or libraries after setting a top module, you must again issue the set_current_design command to re-elaborate the design before you can operate on the design.

If you have operated on the design or have read a flat model, you cannot read additional designs or libraries, or set a new top design without either deleting the current settings using the delete_design command or using the “read_verilog -force” or “read_cell_library -force” command.

When you use the command:

```
set_current_design [get_module M1 -in_library L1]
```

you are able to specify the logical library in which the module M1 exists. When using a module_name as the argument to the set_current_design command, the -in_library option is used to specify in which library to look for module name *module_name*. When the top level is an architecture instead of a module, the get_module command cannot be used to refer to it. You must refer to it only by name. In this case, you must use the -in_library switch if the architecture is not in the default logical library.

In the dft and “patterns -ijtag” context, the set_current_design command is used to set the current design even if the design only has an ICL view. When set_current_design executes in

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

these contexts, if a loaded ICL module exists whose name matches the specified current design name, the tool sets the context to ijtag retargeting; if no loaded ICL module exists whose name matches the specified current design name, the tool enables the ICL extraction mode.

In the IJTAG retargeting context, if a design module is not already loaded, the tool creates a design module with the ports found in the ICL module. If a design module is already loaded, the tool verifies that the module contains all of the ports defined in the ICL module. If a filename *design_name.pdl* is found beside the file that contains the top level ICL module, the PDL file is automatically sourced as described in "dft_inserted_designs" on page 4820.

The tool writes out patterns that include the list of ports found within the design module as well as those in the ICL module. If you plan to use a test_setup procedure that interacts with ports that are in the design module but not described in the ICL module, you must load the design view and not let the tool create it from the ICL module.

Ports (and their attributes) are not deleted by the set_current_design command when their module is no longer in the elaboration tree. Instead, the tool stores this information, which becomes accessible when their module re-appears in the elaboration tree, for example, due to another issuance of the set_current_design command.

The set_current_design command issues a warning when port declarations use data types that are not visible to the tool. Use report_current_design to obtain more information about these ports.

Elaborated Designs and Importing Generated Clocks

When in the DFT context with no sub-context, added clocks are rule checked when set_dft_specification_requirements -memory_test or -logic_test are set to on.

When process_dft_specification is called, the rule-checked clock definitions are stored into the TSDB in the following location within the dft_inserted_designs directory:

```
design_name_design_id.dft_inserted_design_design/design_name.tcd
```

In successive DFT insertion passes when the set_current_design command is called in the DFT context with no sub-context, the clocks are automatically restored. They are also automatically restored if the set_current_design command is called in the patterns -ijtag context. In contrast, the clocks are not automatically restored in other contexts.

In the patterns -scan context, you use the import_scan_mode command followed by the import_clocks command to import the clocks. The import_scan_mode command only works if you used Tessent Scan for scan insertion, and the import_clocks command only works if you ran the DFT step described above that resulted in the clocks being stored in the TSDB.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Files Read During set_current_design

Prior to elaboration and given that the current design in memory is not an interface (blackbox):

- The interface file (.<*extension*>_*interface*) of all child blocks of the current design are loaded, unless the child module is already found in memory.

  The interfaces that were loaded are automatically removed if the module did not get elaborated and this can happen when the interface of its parent block has been loaded.

- When the interface file of a child block has been loaded, the corresponding Tessent Core Description file (*.tcd*) for the child module is also loaded.

There are some exceptions to the automatic loading of the interface and TCD files:

- When the interface_has_external_dependencies property is set in the *.tsdb_info* file of the child block. The interface_has_external_dependencies property is set to "1" if the port list of the interface has external dependencies such as typedef packages or System Verilog interfaces.

  Note

  If you want to load the child block, use the "read_design *child_block* -view interface" command. Otherwise, the tool reports an error because the child block is a black box.

- For the module specified as "set_current_design -icl_module *module*", which is used post-synthesis to reconcile the ICL extracted from the RTL and the gate netlist.

Prior to elaboration, the module matching search is augmented to automatically load some files from the TSDBs when needed:

- For ICL files (*.icl*) and TCD scan file (*.tcd_scan*)
  - All the paths to *.dft_inserted_design* sub-directories of all child blocks of the current design.
  - All the paths to all the *.instrument* sub-directories of all child blocks of the current design.
- For TCD Bscan files (*.tcd_bscan*)
  - All the paths to *.dft_inserted_design* sub-directories of all child blocks of the current design.

Important Considerations

For a specific <*design_name*>/<*design_id*> combination, it is an error to have more than one <*design_name*>_<*design_id*>*.dft_inserted_design* in any opened TSDB.

When you want to redo the DFT insertion for a specific <*design_name*>/<*design_id*> combination, the <*design_name*>_<*design_id*>*.dft_inserted_design* must be in the current

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

output directory (set_tsdb_output_directory, defaults to *tsdb_outdir*) to be overwritten. Otherwise, an error occurs since DFT insertion never writes outside of the current output directory.

When redoing DFT insertion of a child block for a specific <*design_id*>, it is recommended to remove any *.dft_inserted_design* for the same block that builds on the <*design_id*> being redone. After DFT insertion has been redone for child block, it is strongly recommended to redo all DFT insertions that build on this DFT insertion pass.

Loading a Design With the -no_hdl Option

If you load a design with the -no_hdl option, the tool loads the interface HDL using information stored in the source file directory to resolve any external dependencies:

```
read_design my_design -design_id my_id -no_hdl
set_current_design current_design_name
```

If there is no TSDB, the tool uses the ICL file previously loaded to find the interface HDL file. If the interface HDL file cannot be found, the tool creates the design from the ICL model. This is necessary for design interface modules that include port definitions with external dependencies on SystemVerilog interfaces or data types defined in packages, and enables you to perform testbench simulations.

If the TSDB is present but incomplete, set_current_design can fail because some files needed to parse the interface HDL file may be missing.

If the interface HDL file includes port definitions to types declared only in the $unit scope, you can read the HDL file explicitly using read_verilog with the -format sv2009 option:

```
read_verilog -format sv2009 tsdb_outdir/.../top.sv09_interface
set_current_design top
```

This is possible because all $unit types and their dependencies are cloned into the same HDL file.

Logic Synthesis and Port Name Mapping

Logic synthesis can flatten or bit-blast ports, causing post-synthesis names to differ from those in the initial RTL. This process is automatic and typically does not require user intervention. For more information, refer to "Automatic Design Mapping" in the *Tessent Shell User's Manual*.

Arguments

- *top_design*

  An optional string that specifies the name of the top-level module in the design. When the module name is provided, the set_current_design command looks for the module in the default library. If you want to make a module from a different library than the current

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

design, you can use the get_modules command to return the module and then pass the module object directly to set_current_design as follows:

```
set_current_design \
  [get_modules module_name -in_library library_name]
```

- -icl_module *icl_module*

  An optional switch and string pair that specifies the name of an already loaded ICL module to uniquify and associate with the current design being set. This switch must be used in the first insertion pass after synthesis when your RTL design is parameterized and multiple netlists are created for the design by your synthesis tool. It specifies the ICL module in the RTL view that maps to the top module of the netlist being elaborated.

  As part of the first DFT insertion pass for a netlist generated by synthesis, the ICL must be uniquified in order to reconcile the naming of the netlist top module with that of the top ICL module. The set_current_design command invoked as "set_current_design *design_name* -icl_module *icl_module_name*" performs this uniquification. It updates the name of the ICL module to match the name of the top module of the netlist (*design_name*). All references to the pre-uniquified module are updated with the new name in the *.icl*, *.tcd*, and *.tsdb_info* files in the *./dft_inserted_designs/<block_name>.dft_inserted_design* directory. In addition, the attribute tessent_original_module_name is added to the wrapper for the uniquified ICL module in the *.icl* file. The value of that attribute is the original name of the top module.

  Furthermore, when you perform the DFT insertion on the netlist at the parent block level, the ICL hierarchy in the *.icl* file in the *./dft_inserted_designs/ <parent_block_name>.dft_inserted_design* directory is updated to reflect the child block top modules in the gate view. The instance properties and the child ICL module names are all updated to reflect the child netlist names. The ChildBlockModules wrappers in the *.tcd* and *.tsdb_info* files are updated as well.

- -in_library *library*

  An optional switch and value pair that specifies to restrict the search to a specified library. The *library* value must be a single string corresponding to one of the logical library names defined with the set_logical_design_libraries command. When the -in_library option is not specified, the tool only looks for modules in the default library. Use this switch when the specified module or top level architecture name is not in the default library. You can only use this option in the rtl context.

- -verbose

  An optional switch that automatically reports each ICL and TCD file read by design module matching. The tool returns the following message for every file read:

  ```
  // Reading <format> file filename
  ```

  where <format> can be at least one of the following: verilog, icl, tcd_bscan, tcd_scan, tcd_fusebox, tcd_memory, tcd_memory_cluster, or tcd_sim_lib.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -show_elaboration_warnings

  An optional switch that displays details of netlist and library warnings.

- -parameter_values *parameter_list*

  An optional switch that specifies a Tcl list of parameter values for top_design when you use the command to specify a parameterized block with non-default parameters. This switch is useful when you have multiple instances of the same core, and each instance requires different parameter values. In particular, during bottom-up DFT insertion, where the tool inserts each core stand-alone before inserting the top of the chip, it is important that the tool parameterize each core in the same way as it is instantiated in the design.

  Specify the Tcl list as alternating parameter name and value pairs:

  ```
  set_current_design <top_design> -parameter_values \
      {paramName1 paramValue1 paramName2 paramValue2 ... }
  ```

  The parameter values must be simple integers or enums. See Example 5 and Example 6 for further information for integers, and Example 7 for enums.

  Restriction

  If you use this option, you must comply with the restrictions outlined in the following sections:

  - Parameter Overrides in the Instantiation of the Root Module of a Child Block
  - Generic Overrides in the Instantiation of the Root Design Entity of a Child Block

  This switch is available only in -rtl mode ("set_context dft -rtl"). For more information, refer to the set_context command.

- -no_icl_module_matching

  An optional switch that prevents the automatic ICL module matching that normally occurs at the end of design elaboration within some contexts, or when you enter some contexts with an elaborated design.

  You should use this switch when ports on design modules with associated ICL modules were renamed during synthesis or layout, *and* you do not need to rerun ICL extraction on this view of the design.

  If you are going to insert new IJTAG nodes in this view of the design, you must rerun ICL extraction. In this case, you must go back to the synthesis step and request to preserve the module boundary for all modules that have an associated ICL module. For more information about how to obtain the list of modules that must have their interface preserved to enable rerunning ICL extraction after synthesis, see the description of the tessent_get_preserve_instances proc under Synthesis Helper Procs in the *Tessent Shell User's Manual*.

  If you do not need to rerun ICL extraction, using this switch prevents the I1 DRC violation you would normally get when a port of a matched ICL module is not found on its associated design module. If you try to rerun ICL extraction after using this switch, the tool reports an

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

error because module matching was disabled and no instances have the instrument_name attribute set.

Examples

Example 1

The following example sets the top design to block_1:

```
set_current_design block_1
//  Note: Top design is 'block_1'.
```

Example 2

The following example finds the top level of the design (block_2) and sets it as the top. This use model only works in the no_rtl context and requires that all modules currently loaded are instantiated below a single module.

```
set_current_design
//  Note: Top design is 'block_2'.
```

Example 3

The following example loads a mixed-language design in the DFT context in RTL mode.

```
SETUP> set_context dft -rtl
# Specify library1 and work as the two active libraries
SETUP> set_logical_design_libraries -logical_library_list {library1 work} \
        -default_library work
SETUP> read_vhdl core*.vhdl -in_library library1 -format 93
# Set the current_design to core1 from library1
SETUP> set_current_design [get_modules core1 -in_library library1]
//  Note: Top design is 'core1'.
```

Example 4

In the following example the parameterized design top_core_pv1 has two netlists after synthesis:

- top_core_pv1_1
- top_core_pv1_2

You load the design top_core_pv1_1 in preparation for the first gate_level insertion pass. When the design is elaborated using the set_current_design command, you need to specify the name of the top ICL module which existed in the RTL view as the value of the -icl_module option. The ICL module 'top_core_pv1' is renamed to 'top_core_pv1_1' and the original name is recorded in the attribute tessent_original_module_name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
SETUP> set_context dft -no_rtl -design_id gate1
SETUP> open_tsdb top_core_pv1.tsdb
SETUP> read_verilog ./synth/top_core_pv1/gates.top_core_pv1_1.v
SETUP> read_design top_core_pv1 -design_id rtl -no_hdl
SETUP> set_current_design top_core_pv1_1 -icl_module top_core_pv1
```

Example 5

In the following example, the parameterized design top_core_pv1 has two parameters and two parameterized views. The first parameter, num_mems, affects DFT insertion; the second parameter, other_param, does not. The value of num_mems is the same for both views but is different from the default value. You can choose either view as the one to use for DFT insertion, and the tool synthesizes both views. Therefore, you must use the -icl_module option in the gate_level insertion flow, as described in Example 4.

Once the design top_core_pv1 loads in preparation for RTL DFT insertion, the set_current_design command creates a parameterized view of top_core_pv1:

```
SETUP> set_current_design top_core_pv1 \
    -parameter_values {num_mems 3 other_param 2}
```

The parameterized view created here is used in RTL DFT insertion.

The tool creates netlists for both views of top_core_pv1 (as modified by RTL DFT insertion) during synthesis. Follow the steps in Example 4 to complete the gate level insertion flow.

Example 6

You can override complex signal parameters. Given the following code:

```
interface vec #(parameter WIDTH = 1);
  logic [WIDTH-1:0] v;
  modport in (input v);
  modport out (output v);
endinterface
module top (input logic clk,
           input logic resetn,
           vec.in d1,
           vec.in d2);
endmodule
```

You can override the parameters of ports d1 and d2 with the set_current_design command:

```
SETUP> set_current_design top -parameter_values {d1.WIDTH 4 d2.WIDTH 8}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 7

You can override enum parameters when they are not locally defined within a module. Given the following code:

```
package pkg1;
   typedef enum {red, green, blue} colors_t;
endpackage

typedef enum {pink, yellow, cyan} topcolor_t;
module top (input logic clk,
            input logic resetn);
   parameter pkg1::colors_t my_pkg_enum = pkg1::blue;
   parameter topcolor_t my_enum = cyan
endmodule
```

You can override the my_pkg_enum parameter with the set_current_design command:

SETUP> set_current_design top -parameter_values {my_pkg_enum pkg1::green}

You cannot override the my_enum parameter because it is defined within the top module. For example, the following set_current_design command produces an error:

SETUP> set_current_design top -parameter_values {my_enum yellow}

```
Error: Not able to create a parameterized view for module/entity 'top'.
Overwriting enum parameter 'my_enum', where enum is defined within a
module, is not supported.
```

Related Topics

delete_cell_library

delete_design

read_cell_library

read_flat_model

read_verilog

set_cell_library_options

set_design_include_directories

set_design_macros

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_current_edt_block

Context: dft -edt, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Directs the tool to apply certain subsequent commands to a particular EDT block only.

Usage

set_current_edt_block ***block_name***

Description

The set_current_edt_block command restricts the applicability of certain subsequent commands to the EDT block having the specified ***block_name***. This is referred to as setting or changing the EDT context and the EDT block on which context-sensitive commands currently operate is referred to as the current EDT block. The current EDT block remains the sole target of the context-sensitive commands until you delete its definition using the delete_edt_blocks command, or you change the context to a different EDT block using the add_edt_blocks command or another set_current_edt_block command.

Use this command when you want to define or redefine the attributes of a specific EDT block using context-sensitive commands. The add_scan_chains command, EDT-specific commands that define attributes for a given block, and some reporting commands are presently the only context-sensitive commands. All other commands (set_system_mode, create_patterns, and report_statistics, for example) apply to the entire netlist.

If multiple EDT blocks are defined and none of them is specified as the current EDT block, subsequent context-sensitive commands that define attributes for a given block ("set_edt_options -channels" for example) returns an error. This can happen if you add multiple EDT blocks, then delete the one that is the current EDT block.

For more information about EDT blocks or the modular Tessent TestKompress flow, refer to the "Modular Compressed ATPG" chapter of the *Tessent TestKompress User's Manual*.

Arguments

- ***block_name***

  A required string that specifies the name of the EDT block you wish to set as the current EDT block for subsequent context-sensitive commands. The ***block_name*** must be one you defined previously using the add_edt_blocks command. To view a list of the current defined block names, enter "report_edt_blocks".

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example lists current user-defined block names, then changes the EDT context to the EDT block named “my_core1_block”:

report_edt_blocks

```
//  my_core1_block
//  my_core2_block
//  my_core3_block
//  my_core4_block   (current block)
```

set_current_edt_block my_core1_block
report_edt_blocks

```
//  my_core1_block   (current block)
//  my_core2_block
//  my_core3_block
//  my_core4_block
```

Example 2

The following example uses the set_current_edt_block command to force the context-sensitive report_edt_configuration command to report on specific EDT blocks:

set_current_edt_block my_core2_block
report_edt_configuration

```
//
//  EDT block "my_core2_block"
//  -------------------------
//  IP version:                    3
//  External scan channels:        3
//  Decompressor size:             10
//  Clocking:                      edge-sensitive
//
//  Note: Only current EDT block "my_core2_block" is reported. Use "report
//  edt configuration -all_blocks" to report the configurations of all
//  blocks.
```

set_current_edt_block my_core4_block
report_edt_configuration

```
//
//  EDT block "my_core4_block"
//  -------------------------
//  IP version:                    3
//  External scan channels:        1
//  Decompressor size:             8
//  Clocking:                      edge-sensitive
//
//  Note: Only current EDT block "my_core4_block" is reported. Use "report
//  edt configuration -all_blocks" to report the configurations of all
//  blocks.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

add_edt_blocks

delete_edt_blocks

report_edt_blocks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_current_edt_configuration

Context: dft -edt

Mode: setup

Prerequisites: One or more compression configurations must be defined with the add_edt_configurations command.

Sets a particular compression configuration for the tool to use within the current design block.

Usage

set_current_edt_configuration ***config_name***

Description

The set_current_edt_configuration command specifies a compression configuration for the current design block. This is necessary so that the tool can apply subsequent commands to define or redefine the attributes of the specific compression configuration.

For more information about compression configurations, see the "Defining Dual Compression Configurations" section of the *Tessent TestKompress User's Manual*.

Arguments

- ***config_name***

  A required string that specifies the name of a compression configuration. The ***config_name*** is the name specified when the compression configurations were defined. Use the report_edt_configurations command to display a list of the currently defined configuration names.

Examples

The following example sets the current EDT block, my_core1_block, and the compression configuration, system_test. Then, it redefines the input and output channels and reports the new configurations.

set_current_edt_block my_core1_block
report_edt_configuration

```
// my_core1_block
// IP version: 4
// Bypass logic: On
// Lockup cells: On
// Clocking: edge-sensitive
// Input channels: [manufacturing_test : 10] [system_test : 4]
// Output channels:[manufacturing_test : 5] [system_test : 2]
// Bypass chains: 5
```

set_current_edt_configuration system_test
set_edt_options -input_channels 3 -output_channels 3
report_edt_configuration

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// my_core1_block
// IP version: 4
// Bypass logic: On
// Lockup cells: On
// Clocking: edge-sensitive
// Input channels: [manufacturing_test : 10][system_test : 3]
// Output channels: [manufacturing_test : 5][system_test : 3]
// Bypass chains: 5
```

Related Topics

add_edt_configurations

delete_edt_configurations

report_edt_configurations

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_current_mode

Context: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting, patterns -failure_mapping

Mode: setup

Specifies the current test mode in pattern retargeting and core mapping flows.

Usage

For -scan and -scan_diagnosis:

set_current_mode [*mode_name*] [-type *mode_type*]

For -scan_retargeting and -failure_mapping:

set_current_mode ***mode_name***

Description

The set_current_mode command specifies the current test mode in pattern retargeting and core mapping flows. The different types of test modes (unwrapped, internal, external, and retargeting) imply different operating behaviors of the scan chains. The command specifies that the design's scan configuration is set up for a particular test mode, mainly for the purpose of generating retargetable patterns for a wrapped core, but you can also use the command in the "core mapping for ATPG" process. Depending on the mode type, the ATPG tool may apply certain constraints automatically (as described under "-type" below). For all mode types, the mode name and mode type are captured in the Tessent Core Description (TCD) file, which is used for pattern retargeting or core mapping.

In addition to the three test modes described under "-type" below, there is a fourth test mode called "retargeting," which you cannot set with the -type switch and is the only test mode available in the "patterns -scan_retargeting" and "patterns -failure_mapping" contexts. The "retargeting" test mode provides access to lower-level cores for the purpose of retargeting patterns. Because "retargeting" is the only available test mode, it is not necessary to use the set_current_mode command in these contexts. The only situation in which you need to use the set_current_mode command is when you are in the "patterns -scan" or "patterns -scan_diagnosis" contexts and you want to do pattern retargeting.

If you have explicitly set the type and then use set_current_mode, without specifying -type, the tool leaves the type as it was set, rather than reset it to the context default.

For more information about the mapping process and TCD files, refer to "Core Mapping for ATPG Process Overview" in the *Tessent Scan and ATPG User's Manual*. For more information about pattern retargeting, refer to "Scan Pattern Retargeting" in the *Tessent Scan and ATPG User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- *mode_name*

  An optional or required literal that specifies a user-defined name for the mode. If you don't specify this argument, the mode_name argument defaults to the value of mode_type. The mode_name is optional for the -scan and -scan_diagnosis sub-contexts and is required for the -scan_retargeting and -failure_mapping sub-contexts.

- -type *mode_type* **("patterns -scan" and "patterns -scan_diagnosis" only)**

  An optional switch that specifies the test mode type for the tool when generating retargetable patterns. Valid options are:

  unwrapped — Specifies the current default test mode in which ATPG runs in the "patterns -scan" context. This mode does not support core isolation or hierarchical test, so use this mode when generating patterns for designs that do not contain wrapper chains.

  internal — Specifies the internal test mode of a wrapped core in which the tool generates patterns while the core is isolated from the rest of the design. Due to this isolation, those patterns are independent of circuitry outside of the core's wrapper chain and can therefore later be retargeted from the core level to a higher level of the design. When you specify this option, the tool automatically adds CX constraints to all primary inputs (including Scan or EDT channel inputs) and masks all primary outputs during design rule checking and pattern generation. However, exceptions include primary inputs already constrained to 1 or 0, and clock inputs. For more information, refer to "Core-Level Pattern Generation" in the *Tessent Scan and ATPG User's Manual*.

  When you run ATPG in the internal type mode, primary inputs (PIs) and primary outputs (POs) are not valid fault sites when the core is embedded at the next level up, which creates inconsistencies between fault accounting at the core level and those at the top. The tool behavior is to not fault PIs and POs in this situation.

  external— Specifies the external test mode of a core in which the tool generates patterns for the interconnect circuitry outside of the wrapper chain. You use the external mode after instantiating the core or a graybox representation in the design. For more information, refer to "Generating Patterns in External Mode" in the *Tessent Scan and ATPG User's Manual*. Note that using external mode does not change constraints or masking of the core pins (as with internal mode), so external mode is identical to unwrapped mode with regard to the tool's ATPG functionality; however, unlike unwrapped mode, external mode implies that you have instantiated wrapped cores.

Examples

This example sets the mode name and mode type to "internal".

```
set_context patterns -scan
set_current_mode -type internal
```

Related Topics

get_current_mode

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_current_physical_block

Context: dft -edt, dft -scan, dft -test_points, patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: setup, analysis

Set the current physical block where all subsequent load/unload and capture timing settings are applied.

Usage

set_current_physical_block
[-icl_module *icl_module_name* | -module *module_name* | -current_design]

Description

The set_current_physical_block command specifies the current physical block where the tool applies subsequent load/unload and capture timing settings. The default value is the current design.

Arguments

- -icl_module *icl_module_name*

  A switch and string pair that specifies the name of a valid ICL module that will be used by the tool as the current physical block.

- -module *module_name*

  A switch and string pair that specifies the name of a module that will be used by the tool as the current physical block.

- -current_design

  A switch that specifies the using the current design as the current physical block. This is the default.

Examples

The following example sets the current physical block to an ICL module named “xtea”.

```
set_current_physical_block -icl_module xtea
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_current_silicon_insight_setup

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Sets the currently selected setup in the Tessent SiliconInsight Setup Specification.

Usage

set_current_silicon_insight_setup ***setup_type***

Description

Each adaptor supports offline mode as indicated by the "_offline" suffix. Use this mode for testing that the patterns run well with the adaptor.

This command only affects the current session. If you want to save a change to the setup specification so that you can read it in using the read_config_data command in a subsequent session, you must use the write_config_data command.

Arguments

- ***setup_type***

  A required string that specifies one of the supported Tessent SiliconInsight setup specification types. Options are:

  NoTester(batch_cdp)

  Sid(jtagkey)

  Sid(jtagkey_offline)

  Sid(flyswatter2)

  Sid(flyswatter2_offline)

  Sid(olimex_arm_usb_ocd)

  Sid(olimex_arm_usb_ocd_offline)

  Sid(olimex_arm_usb_ocd_h)

  Sid(olimex_arm_usb_ocd_h_offline)

  Sid(opalKellyXem6310)

  Sid(opalKellyXem6310_offline)

  Sid(signalyzer)

  Sid(signalyzer_offline)

  Sid(signalyzerSHA40)

  Sid(signalyzerSHA40_offline)

  Sid(signalyzer40)

  Sid(signalyzer40_offline)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Sid(signalyzerH4)

Sid(signalyzerH4_offline)

Sid(signalyzerSP)

Sid(signalyzerSP_offline)

Sid(simdut)

Ate(ate1)

Examples

set_current_silicon_insight_setup Sid(simdut)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_current_simulation_context

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist.

Sets the current simulation context.

Usage

Context: all contexts

set_current_simulation_context ***context_name***

Description

The set_current_simulation_context command sets the simulation context for later analysis and introspection. Table 8-6 lists the predefined simulation contexts for all contexts except dft -scan and dft -test_points. Table 8-7 lists the predefined simulation contexts that are only available for dft -scan and dft -test_points contexts.

Table 8-6. Predefined Simulation Contexts For All Contexts Except dft -scan and dft -test_points

Simulation Context	Description
stable_after_setup	This context specifies the values that are constant in the period following the end of the test_setup procedure and the beginning of the test_end procedure. For scan patterns, the value is constant throughout scan and capture cycles. For IJTAG patterns, the value is constant throughout any IJTAG operations.
stable_capture	This context specifies the values that are constant throughout the capture procedures.
stable_load_unload	This context specifies the values that are constant throughout the load_unload cycles.
stable_shift	This context specifies the values that are constant throughout the shift cycles.
tie_value	This context specifies the values that are constant due to tied nodes in the design. For example, a net assigned to 1'b0 has a tie value of 0. Primary inputs constrained with the “add_input_constraints -ct0 or -ct1” switches are also considered tied and are reported as constant in this simulation context.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-7. Predefined Simulation Contexts For dft -scan and dft -test_points Contexts Only

Simulation Context	Description
srules_all_clocks_off	Use this context to view the values that were present at the start of the S-rules analysis.
srules_all_clocks_x_for_capture	Use this context to view an S-rules specific background that has all clocks set to X. This can be used to debug how shift clocks are assigned to each scan element.
srules_s1_background	Use this context to view the exact values that were used to perform the S1 analysis.

Arguments

- ***context_name***

  A required literal that specifies the simulation context to make current. You can specify any of the simulation contexts listed in Table 8-6, Table 8-7 or any user-defined simulation contexts that have been added. By default, the simulation context is stable_after_setup.

Examples

Example 1

The following example sets the current simulation context to stable_shift:

```
> set_current_simulation_context stable_shift
```

Related Topics

get_current_simulation_context

get_simulation_context_list

report_simulation_contexts

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_dedicated_wrapper_cell_options

Context: dft -scan

Mode: setup, analysis

Controls how the tool infers the dedicated wrapper cells (DWCs) for primary IOs.

Usage

set_dedicated_wrapper_cell_options [**on** | **off** | **auto**]
[-ports *port_spec*]
[-input_module_name *module_spec*]
[-output_module_name *module_spec*]
[-clock_source *port_pin_spec*]
[-below_instances *instance_spec*]
[-isolate_clusters {on | off}]

Description

The set_dedicated_wrapper_cell_options command controls how the tool infers DWCs on all or the specified primary IOs. It defines how the DWCs on each port of the current design are handled during wrapper analysis. It is an incremental command and you can use it multiple times to generate a desired superset of options.

Note

During wrapper analysis, the tool checks whether a PI is driving the Set or Reset ports of a flip-flop encountered during the forward tracing, and if so, it infers a dedicated wrapper cell on this PI. For more information on this tool behavior and how you can influence it, see "Identification of Clock and Reset Ports" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- **on** | **off** | **auto**

  Ports that are excluded from wrapper analysis are not affected by this command. If excluded ports are specified with the -ports switch or if the -ports switch is not specified and there are excluded ports in the design you see the following warning:

```
// Warning: Ignoring excluded port(s) <port1>, <port2>, <port3> …
//          To include these ports in wrapper analysis please use:
//          set_wrapper_analysis_options -clear_excluded_ports
```

  An optional switch that specifies whether to auto infer the dedicated wrapper cells on all or the specified primary IOs, or to have the dedicated wrapper cells added to all or the specified primary IOs, or to have all or the specified primary IOs excluded from having the dedicated wrapper cells added to them:

  Undriven outputs receive dedicated wrapper cells based on the configurations of "set_wrapper_analysis_options -register_undriven_output on|off" and are not affected by global on|off|auto settings. However, specifying a specific undriven port (or group of

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

undriven ports) with "-ports {…}" overrides the -register_undriven_output settings for that/those specific ports(s) as seen in Example 1, following.

- **on** — specifies to add a dedicated wrapper cell for the given ports. The "on" state is used to automatically register all primary IOs specified with the -ports switch (including undriven outputs), or to register all primary IOs (with the exception of undriven outputs) when the -ports specification is not provided.
- **off** — specifies to force the ports to be covered by shared wrapper cells even if it requires exceeding the thresholds set by the following switches in the set_wrapper_analysis_options command:
  - -output_fanin_flop_threshold
  - -input_fanout_flop_threshold
  - -output_fanin_libcell_levels_threshold
  - -input_fanout_libcell_levels_threshold
  - -non_scan_depth_threshold

  This applies to the primary IOs specified with the -ports switch (including undriven outputs), or all primary IOs (with the exception of undriven outputs) when the -ports specification is not provided.

  In addition, any previous mapping of clock source to wrapper cells are cleared for the given ports (or for all ports if the -ports switch is not specified).
- **auto** — specifies that dedicated wrapper cells are inferred and determined by the tool during the wrapper cells analysis for the primary IOs specified with the -ports switch or all primary IOs when the -ports specification is not provided. This is the default.

- -ports *port_spec*

  An optional switch and string that specifies the ports for which the current setting applies to. When this switch is not specified, the dedicated wrapper cell inferring setting is applicable to all primary Input and Output ports.

  port_spec can be a Tcl list of one or more ports, or a collection of one or more port objects.

- -input_module_name *module_spec*

  An optional switch and string pair that specifies a library model to use to instantiate a dedicated Input wrapper cell.

  The *module_spec* string can be the name of a library model or a collection containing a single library model object.

  The tool implements dedicated wrapper cells with capture_window_behavior according to the structure of the library model. The "set_wrapper_analysis_options -capture_window_behavior" command setting does not affect dedicated wrapper cells that the tool implements using the library model.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -output_module_name *module_spec*

  An optional switch and string pair that specifies a library model to use to instantiate a dedicated Output wrapper cell.

  The *module_spec* string can be the name of a library model or a collection containing a single library model object.

  The tool implements dedicated wrapper cells with capture_window_behavior according to the structure of the library model. The "set_wrapper_analysis_options -capture_window_behavior" command setting does not affect dedicated wrapper cells that the tool implements using the library model.

- -clock_source *port_pin_spec*

  An optional switch and string pair that specifies an internal pin or top-level port to use as the clock source for the inserted dedicated wrapper cells.

  The *port_pin_spec* string can be the name of a pin, port, or a collection containing a single pin or port object.

  The tool applies this clock source to the dedicated wrapper cells of the specified primary IOs when the -ports switch is specified or to the dedicated wrapper cells of all primary IOs when the -ports switch is not specified.

  The tool ignores this switch when combined with "off", and it reports the following error message:

  ```
  // Error: -clock_source has no effect when dedicated wrapper cells
  // are off (disabled).
  ```

  Note

  For more information, see Clock Selection for Dedicated Wrapper Cells in the *Tessent Scan and ATPG User's Manual*.

- -below_instances *instance_spec*

  An optional switch and string pair that specifies the instances below which the tool can place dedicated wrapper cells. By default, the dedicated wrapper cell placement is at, or as close as possible, to primary input and output ports. Use this switch to optionally move the dedicated wrapper cells further down the design hierarchy.

  If you call this command more than once for a specific port or without the -ports switch, the most recent call overrides previous calls. Therefore, the tool removes any instances previously specified for that port unless you also specify them in the latest call.

  If you do not specify any instances for a specific port, the tool places the DWCs according to the global default. If the global default is also empty, the tool places the dedicated wrapper under the top level or as close as possible, given other constraints.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The tool ignores this switch when combined with "off," and it reports the following error message:

```
// Error: The switch -below_instances has no effect when the
dedicated wrapper cells option is set to off (disabled).
```

Also, when run without the -ports switch, running "set_dedicated_wrapper_cells OFF" resets all settings of instances below which the tool may place dedicated wrapper cells, including the default setting and specific settings for all ports. When you specify the -ports switch, the tool resets the settings only for the specified ports.

Restriction

You cannot place DWCs below hard or blackbox modules, and we do not recommend you place DWCs below non-scannable instances. Therefore, this switch ignores non-scannable instances, blackbox instances, and instances of hard modules. The tool reports an appropriate warning message. If no valid instances remain after removing these instances, the tool reports the following error message:

```
// Error: none of the instances specified with '-below_instances'
are valid for placement of dedicated wrapper cells.
```

instance_spec — A Tcl list of one or more instances or a collection of one or more instance objects. The tool ignores non-scannable instances, blackbox instances, and instances of hard modules.

- -isolate_clusters {on | off}

  An optional switch and literal pair that specifies dedicated wrapper cells (DWCs) per port per "cluster_name" attribute. This could lead to multiple DWCs per port. The tool treats state elements with no specified cluster names as a separate cluster. The default is off.

  The tool ignores this switch when combined with "off," and it reports the following error message:

```
// Error: The switch -isolate_clusters has no effect when the
dedicated wrapper cells option is set to off (disabled).
```

  Also, running "set_dedicated_wrapper_cells OFF" resets specific settings for all ports when run without the -ports switch. When you specify the -ports switch, the tool resets the settings only for the specified ports.

Examples

Example 1

In this example, all driven Primary IOs receive dedicated wrapper cells. Undriven IOs do not receive dedicated wrapper cells:

set_dedicated_wrapper_cell_options on

Example 2

In this example, all Primary IOs receive dedicated wrapper cells, including undriven IOs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_dedicated_wrapper_cell_options on
set_wrapper_analysis_options -register_undriven_output on
```

Example 3

In the following example:

Ports A, B, C, D, excluded_1 are driven.

Ports undriven_1, undriven_2 are undriven.

Ports A, D, undriven_1 receive dedicated wrapper cells.

Ports B, C, undriven_2, excluded_1 do not receive dedicated wrapper cells.

```
set_wrapper_analysis_options -exclude_ports excluded_1
set_dedicated_wrapper_cell_options on
set_dedicated_wrapper_cell_options off -ports {A B C}
set_dedicated_wrapper_cell_options on -ports {A undriven_1}
```

Example 4

The following example shows how you can place the dedicated wrapper cells that drive set and reset ports at the top level, while the tool places all other dedicated wrapper cells inside an instance (not top) using the -below_instances switch. The tool can only place dedicated wrapper cells that drive ports with the “data_a” prefix inside the instance and sub-instances of “blockA/subblockA”.

```
// Dedicated wrapper cells for all ports must be placed inside some
// instance (they may not be placed in the top level).
set_dedicated_wrapper_cell_options -below_instances [get_instances]

// Disable this restriction for dedicated wrapper cells that drive the “set” and “reset” ports.
// (They may be placed in the top level)
set_dedicated_wrapper_cell_options OFF -ports {set reset}
set_dedicated_wrapper_cell_options AUTO -ports {set reset}

// Set even harder limitations on dedicated wrapper cells associated with the “data_a” prefix,
// they must be placed inside the sub-block “subblockA” that is inside the instance of
// “blockA”
set_dedicated_wrapper_cell_options –ports {data_a*} -below_instances {blockA/subblockA}
```

Related Topics

set_wrapper_analysis_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_default_ssn_datapath_configuration

Context: all contexts

Mode: setup

Adds an override setting to ICL ports, ICL pins, or ICL ScanRegisters, to override the default SSN datapath configuration.

Usage

set_default_ssn_datapath_configuration [-instance *design_instance*]
-target *icl_pin_port_or_reg* -value *pdl_value*

Description

The tool requires the SSN datapaths to be correctly configured during system mode transition from setup to analysis to perform rule checks to derive certain properties of the SSN. A valid SSN datapath configuration consists of a suitable configuration of the multiplexers on the datapath and a suitable configuration of the multiplexers on the SSN clock paths if there are any.

The default state of the ICL network is derived from ResetValue and DefaultLoadValue specifications in the ICL and from setting the DFT signal ssn_en to 1. However, the default state of the ICL network may not constitute a valid SSN datapath configuration. If several instances of the same core must be configured differently, adjusting the ResetValue and DefaultLoadValue specifications does not solve the problem.

The command set_default_ssn_datapath_configuration addresses this specific problem. The assignments override the default state of the ICL network during the SSN-related processing steps in the system mode transition that require a complete SSN datapath configuration.

This command does not require the ICL to be extracted and elaborated because the command is supposed to be usable in a flow that incorporates ICL extraction. The tool does not check arguments when running this command. Arguments are checked later during system mode transition when the complete ICL is available.

Note

The tool does not generate any patterns to obtain the specified state of the ICL network. If necessary, you must do one of the following:

- Perform appropriate adaptations of the test_setup procedure
- Introduce suitable test steps to align the actual state of the ICL network with the assumed state specified using set_default_ssn_datapath_configuration

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- -instance *design_instance*

  An optional switch and value pair that specifies a design instance associated with the ICL instance that is the starting point for lookup of the target. This enables you to specify ICL elements without knowing the full ICL instance path, such as prior to invocation of extract_icl, when the hierarchical ICL model has not yet been created.

- **-target *icl_pin_port_or_reg***

  A required switch and value pair that specifies a primary ICL input port, an ICL pin, or an ICL ScanRegister. The icl_pin_port_or_reg parameter can also specify a sub-range. Collections are not valid. The tool can only use ICL pins if primary input ports or ScanRegisters ultimately drive them. The target is relative to the ICL instance associated with the design instance specified in the -instance argument. If you do not specify -instance, the target specification is relative to the current design.

- **-value *pdl_value***

  A required switch and value pair that sets the specified PDL value on the ICL target.

Examples

The following example sets two PDL values of two ICL targets to override the default SSN datapath configuration.

```
SETUP> set_default_ssn_datapath_configuration -target cfginst.din -value 0b1101xx10
SETUP> set_default_ssn_datapath_configuration -target ssnMuxCtrl -value Upper
```

The following example instructs the tool to apply the ICL enumeration value “Upper” to the ICL element “ssnMuxCtrl” in an ICL instance that is expected to be associated with the design instance “tdr_inst”.

```
SETUP> set_default_ssn_datapath_configuration -instance tdr_inst -target ssnMuxCtrl \
       -value Upper
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_defaults_value

Context: dft, patterns, unspecified

Mode: setup, analysis, insertion

Sets the default value of a property in one of the DefaultsSpecification (policy) wrappers.

Usage

set_defaults_value ***property_path_name*** {*value* | -unspecify} [-policy *policy*]

Description

This command has access to and sets default values for the properties of the sub-wrappers in the top DefaultsSpecification(<policy>) to match your preferences and requirements. The DefaultsSpecification(<policy>) sub-wrappers are DftSpecification and PatternsSpecification.

The set_defaults_value command automatically manages the priority of the <policy> and sets the value of the highest priority by default. The policies, in order of highest priority to lowest are: user, group, and company.

Note

The values set using this command impact the default values used by the create_dft_specification command. They do not impact existing dft specifications.

Use the command "report_config_syntax DefaultsSpecification" to see all the available properties you can set. You can use the -level 1 and -wrapper_only switches to see the next level of sub wrappers and add the wrapper name you care about to the path to limit the size of the report as follows:

```
> report_config_syntax DefaultsSpecification/DftSpecification -level 1 \
  -wrappers_only
DefaultsSpecification(<policy>) { // legal: company group user
  DftSpecification {
    IjtagNetwork {
    }
    BoundaryScan {
    }
    MemoryBisr {
    }
    MemoryBist {
    }
  }
}

> report_config_syntax DefaultsSpecification -level 1 -wrappers_only
DefaultsSpecification(<policy>) { // legal: company group user
  PatternsSpecification {
  }
  DftSpecification {
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
> report_config_syntax DefaultsSpecification/DftSpecification -level 1 \
  -wrappers_only
DefaultsSpecification(<policy>) { // legal: company group user
  DftSpecification {
    IjtagNetwork {
    }
    BoundaryScan {
    }
    MemoryBisr {
    }
    MemoryBist {
    }
  }
}

> report_config_syntax DefaultsSpecification/DftSpecification/\
  IjtagNetwork -level 1 -wrappers_only
DefaultsSpecification(<policy>) { // legal: company group user
  DftSpecification {
    IjtagNetwork {
      ImplementationOptions {
      }
      HostScanInterface {
      }
    }
  }
}

SETUP> report_config_syntax DefaultsSpecification/DftSpecification/\
       IjtagNetwork/ImplementationOptions -level 1
DefaultsSpecification(<policy>) { // legal: company group user
  DftSpecification {
    IjtagNetwork {
      ImplementationOptions {
        scan_path_retiming : <string> ; // legal  : (latch) flop
      }
    }
  }
}
```

Arguments

- ***property_path_name***

  A required string that specifies a configuration pathname *without* the DefaultsSpecification prefix. For the DftSpecification, this path starts with “DftSpecification/”prefix. For the PatternsSpecification, this path starts with “PatternsSpecification/”. For example:

  ```
  set_defaults_value \
  DftSpecification/MemoryBist/ControllerOptions/use_multicycle_paths off
  ```

- *value*

  An optional value that specifies the value of the specified property.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -unspecify

  An optional value that unspecifies the property instead of setting its value. Use this option to remove a property setting for a policy.

- -policy *policy*

  An optional switch and value pair that specifies for which DefaultsSpecification (policy) wrapper the property is to be set or unspecified. The available policies, in order from highest priority to lowest, are: user, group, and company. When this switch is not specified, the user policy is used.

Examples

Example 1

The following example sets the rtl_extension property in the company DftSpecification policy and unspecifies the same property in the user policy.

```
set_defaults_value DftSpecification/rtl_extension vb -policy company
```

Unspecify the rtl_extension property in the DftSpecification user policy.

```
set_defaults_value DftSpecification/rtl_extension unspecify -policy user
```

Because the user policy for this property was just unspecified and assuming the group policy for this property was not set, it returns vb:

```
get_defaults_value DftSpecification/rtl_extension
```

Example 2

This example shows using set_defaults_value and get_defaults_value that, when automatically resolving the priority, the default value of the property being set is accessed.

```
get_defaults_value PatternsSpecification/MaufacturingOptions/manufacturing_patterns_formats
```

The command returns STIL, the default, because the property, in this case, is not set for any of the PatternsSpecification priorities.

This command sets the value of the “group” policy for this property to Wgl:

```
set_defaults_value PatternsSpecification/ManufacturingOptions/manufacturing_patterns_formats Wgl \
 -policy group
```

This command returns the default value for the PatternsSpecification/ManufacturingOptions/manufacturing_patterns_formats property:

```
get_defaults_value PatternsSpecification/ManufacturingOptions/manufacturing_patterns_formats
```

The command returns wgl because only the PatternsSpecification/ManufacturingOptions/manufacturing_patterns_formats group priority property has been set. The get_defaults_value command first looks for a value for the property in the user priority and if not set, it looks in the group priority and finds, in this case, wg1.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This command demonstrates that the tool uses the default value for the property when it has not been set:

```
get_defaults_value PatternsSpecification/ManufacturingOptions/manufacturing_patterns_formats \
-policy company
```

This returns STIL, the default for the “company” policy,

Related Topics

get_defaults_value

create_dft_specification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_design_include_directories

Context: unspecified, all contexts

Mode: setup

Specifies one or more directory paths that contain include files for use by subsequent read_verilog commands.

Usage

set_design_include_directories *include_dir* … | [-Clear]

Description

If you run this command multiple times, the last set of include directories is used. When you use this command without arguments, the command reports the list of include directories.

Each include_directory must be relative to the current tool invocation directory or an absolute directory pathname. Include files are searched for in the following order of precedence:

1. Absolute pathnames specified by `include directives in the Verilog design.
2. Directories specified with the command.

   For relative `include file targets, directories are searched in the order they are listed, left to right, in the command. If identically named files exist in multiple directories, the first file found is used and the others are ignored.
3. Directory where the Verilog design file is located.
4. Directory in which you invoked the tool.

Arguments

- *include_dir*

  A string that specifies one or more directory paths.
- -Clear

  An optional switch that removes the current list of include directories.

Examples

The following commands specify that the tool search for `include files contained in ../dir1 when loading the top.v module:

```
set_design_include_directories ../dir1
read_verilog top.v
```

Related Topics

delete_design

read_cell_library

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_flat_model

read_verilog

set_current_design

set_design_macros

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_design_level

Context: dft, patterns

Mode: setup, insertion

Prerequisites: You can set the level can only after the design is elaborated.

Specifies the level of the current design.

Usage

set_design_level {**chip** | **physical_block** | **sub_block** | **instrument_block**}

Description

You must specify the design level with this command when you are in the “dft -rtl” or “dft -no_rtl” contexts and want to change to analysis system mode. There is no default setting, so you must specify the appropriate level because this setting affects the create_dft_specification and process_dft_specification commands, and some design rule checks that the tool performs when changing to the analysis system mode.

If you issue the extract_icl command immediately after having issued the process_dft_specification command, the tool uses the specified design level during extract_icl to set the tessent_is_physical_module attribute to true on the top ICL module. If you did not run the extract_icl command in the same tool session as the dft step, you must set the design level prior to issuing the extract_icl command.

When to Use the physical_block Level Versus the sub_block Level

When using the standard Tessent Shell workflows, use the physical_block and sub_block levels for the child block, and chip level for the top.

Note

The instrument_block level is for a special manual assembly flow described in the next section and illustrated in Example 2.

Understanding the difference between a physical block and a sub-block is critical. A physical block is synthesized and laid out as a separate entity. When a physical block is instantiated below another physical block, the netlist view of the child physical block is not loaded into the synthesis or layout tool when synthesizing the parent physical block. Instead, a timing model for the child physical block is used. This means the SDC created for the parent physical block can only refer to pins of the child physical block. It is not possible to reference any elements within the child physical block. You cannot on the other hand refer to pins of a sub-block using -to or -from when synthesizing it within its parent physical block. Instead you refer to the elements inside the sub-block. Another major difference is the fact that the netlist of the sub-block typically only exists as part of the parent physical netlist. You do scan insertion from the physical block boundary and insert the scan chains into the sub-block at the same time. If the lower block is a physical block, you insert the chains inside the lower physical block first and then you connect to those scan chains when inserting scan chains in the parent physical block.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In the RTL DFT flow, the sub-block is treated pretty much the same way as if it was a physical block. If you want to use the read_def command to extract the placement information of the memories to build one memoryBIST controller per physical cluster, you have to use the "-instance_of_current_design" switch in the read_def command if you are inserting memoryBIST in a sub-block. You only can get a DEF file for a physical block. By definition, you cannot get a DEF file for a sub-block because it is synthesized within the parent physical block. You use the DEF file of the parent physical block and use the "-instance_of_current_design" switch to specify what is the instance path of your sub-block within the parent physical block DEF file.

During signoff simulation, the simulation testbenches are separated per physical block to optimize performance during simulation. See the create_patterns_specification command description and the simulate_instruments_in_lower_physical_instances property in the DefaultsSpecification/PatternsSpecification/SignOffOptions wrapper for more information about the signoff simulation. If you have sub-block modules, they are simulated as part of their parent physical block because by the time you get to gate-level simulation, you only have a netlist and an SDF file per physical block as the sub-blocks are synthesized within their parent physical block.

When you use read_design to load a sub-block inside the tool, the is_hard_module Module-level attribute is automatically inferred to "true" on the sub-block modules if they are not the current_design and the -rtl context is used. When you insert DFT elements inside the current design, you can only make connections to the pins of the sub-block modules. If you need to add something inside the sub-block, you have to make the sub-block module the current design and do the additions at that level. You also extract the ICL file for the sub-block at that level. Remember that ICL extraction in the parent physical region is performed by re-using the ICL description of the sub-block and tracing outward from the pins of the sub-block instances. If edits were done in the sub-block module from the parent level, the changes would not be reflected in the extracted ICL. When you reach the gate-level DFT flow, as was described above, the scan insertion is performed from the parent physical block level and the scan chains are inserted inside the sub-block at the same time as in the parent physical block. Inserting scan chains and test points in the netlist has no impact on the ICL network. You do not need to re-run ICL extraction post-synthesis unless you are inserting IJTAG nodes after synthesis. Just like the rest of your functional logic, the equivalence between the pre- and post-synthesis views is verified using formal verification tools. The ICL network is functional logic to formal verification tools.

When to Use the instrument_block Level

If you want to avoid having Tessent Shell modify your golden RTL, a flow that is commonly used is to manually insert a DFT module into your design block and manually make the connections to and from this special DFT module. Such a flow is illustrated in Example 2. The Tessent DFT flow is performed on the DFT module by making it the current design and inserting the IjtagNetwork, the EDT controllers, the SSN datapaths, and the OCC controllers within it. The DFT module has the clock, IJTAG network and SSN ports pre-existing and manually pre-connected in the parent module. The Tessent Shell flow simply populates the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

content of the DFT module. You must specify the design level as instrument_block for this module.

When you reach the parent level of this block, you want to perform the rest of the flow with all its automation as if you had inserted the DFT elements from that level following the normal flow. This happens automatically when the design level of the DFT module was specified as instrument_block. The DFT signals added within the DFT module are promoted to exist in the parent level and the design source dictionary (get_design_sources -file_dictionary) of the parent block includes the files containing the modules of the DFT elements inserted into the DFT block. Once you have run ICL extraction on the parent module, the DFT elements on the child DFT block are all promoted into the parent block's TSDB as if you had inserted them at that level to start with. See Example 2 for an illustration of this flow.

Note

The instrument_block flow is not designed for and is not recommended for Memory BIST. You may be able to use it, but it involves complicated manual work that often outweighs the benefits.

- You can instantiate all your memories in the instrument_block and manually route all function connections to and from the memories through that block.
- Alternatively, you may have placeholder memory modules in the instrument_block with only the BIST multiplexer. They route every functional connection to and from the memories through those placeholder memory modules.
- The memory BIST insertion is then applied normally to the real or placeholder memories in the instrument blocks.
- This manual preparation is error-prone and needs adjustment each time a memory is changed, so it is not recommended.

Arguments

- **chip | physical_block | sub_block | instrument_block**

  A required literal that specifies the design level:

  chip — The design is intended to be used at the chip level. This implies that all ports have a pad buffer and a TAP is inserted while running the create_dft_specification command. A chip is, by definition, also a physical block. It is possible to assemble a chip that instantiates one or more chips. You can use "top" as a synonym of "chip".

  physical_block — The design is intended to be used for blocks that are synthesized and laid out as independent blocks. There are several implications when defining a module as a physical block versus a sub_block. Those are described above. Take care to specify the correct value.

  sub_block — The design is intended to be used for blocks that are synthesized and laid out as part of a parent physical block. There are several implication when defining a

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

module as a physical block vs a sub-block. Those are described above. Make sure to specify the right value.

instrument_block — The design is a special empty module into which the DFT elements are inserted. The special module is manually instantiated into its parent block and its pins are manually connected inside the parent block. See the When to Use the instrument_block Level section for more information about this special flow.

Note

There is no default setting, so you must specify one of these design levels.

Examples

Example 1

The following example sets the level of the current design to physical_block.

```
set_design_level physical_block
```

Example 2

The following example illustrates the usage of the instrument_block design level, which is described in the When to Use the instrument_block Level section.

Begin with an empty DFT module that contains the port definition for IJTAG and SSN, and an input and output port for each functional clock. Create output ports for the all_test and async_set_reset_dynamic_disable DFT signals, which you can use to control clock multiplexers and disable your asynchronous set and reset signals during scan shifting. For descriptions of these DFT signals, refer to Table 3-10 on page 201 and Table 3-13 on page 219 in the add_dft_signals command reference page.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
module gps_baseband_dft_box (
  clk, clk_occ,
  ijtag_tck, ijtag_si, ijtag_ce, ijtag_se, ijtag_ue,
  ijtag_sel, ijtag_reset, ijtag_so,
  ssn_bus_clock, ssn_bus_data_in, ssn_bus_data_out,
  async_set_reset_dynamic_disable, all_test);

input        clk;
output       clk_occ;
input        ijtag_tck, ijtag_si, ijtag_ce, ijtag_se, ijtag_ue,
             ijtag_sel, ijtag_reset;
output       ijtag_so;
input        ssn_bus_clock;
input  [1:0] ssn_bus_data_in;
output [1:0] ssn_bus_data_out;
output       async_set_reset_dynamic_disable, all_test;

//Drive the DFT signal to their off values
assign       async_set_reset_dynamic_disable = 1'b0;
assign       all_test = 1'b0;
// Add feedthrough for ssn bus
assign       ssn_bus_data_out = ssn_bus_data_in;
// Add feedthrough for ijtag chain
assign       ijtag_so = ijtag_si;
// Add feedthrough for functional clock that will be equipped with OCCs
assign       clk_occ = clk;

endmodule
```

You manually instantiate and connect this DFT module in your parent module. Modify your RTL to use clk_occ wherever it used clk, so that the clock of all your scannable elements is controlled by the OCC that will be inserted into the block.

```
gps_baseband_dft_box gps_baseband_dft_box (
  .clk(clk), .clk_occ(clk_occ),
  .ijtag_tck(ijtag_tck), .ijtag_si(ijtag_si), .ijtag_ce(ijtag_ce),
  .ijtag_se(ijtag_se), .ijtag_ue(ijtag_ue),
  .ijtag_sel(ijtag_sel), .ijtag_reset(ijtag_reset), .ijtag_so(ijtag_so),
  .ssn_bus_clock(ssn_bus_clock), .ssn_bus_data_in(ssn_bus_data_in),
  .ssn_bus_data_out(ssn_bus_data_out));
```

Perform the DFT insertion flow within the DFT module as normal, with those two differences:

```
# Specify the design level to instrument_block
set_design_level instrument_block
# Use add_dft_control_points to connect the DFT signals to the output
# ports
add_dft_control_points async_set_reset_dynamic_disable \
           -type dynamic_dft_control \
           -dft_signal_source_name async_set_reset_dynamic_disable
add_dft_control_points all_test -dft_signal_source_name all_test
```

At the next level, read your design normally then load the DFT module using the read_design command. Specify your clocks and run the check_design_rules command to validate and store them. Then, call the extract_icl command. At that point, it is as if you had inserted the DFT

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

elements from this level using the normal flow. After ICL is extracted, you can go to the DFT context and verify that the RTL has controllable clock and reset signals. DRC violation errors indicate if the RTL is not clean, which you must fix manually.

```
set_context dft -rtl -design_id rtl1

# Read in the files making up this block
read_verilog -f gps_baseband.f

#Overwrite the dft box module with the DFT inserted version
read_design gps_baseband_dft_box

# Elaborate the design and set its design level
set_current_design gps_baseband
set_design_level physical_block

# Define your clocks
add_clocks clk -period 3ns

# Call check_design_rules to store the clock definition
check_design_rules

# Extract IJTAG network and create ICL file for the design
extract_icl -create_ijtag_graybox on -write_in_tsdb on

report_dft_signals

# Creates a Synthesis script of ALL the RTL(original and newly created) to
#    be used in DC Shell in Step 2
write_design_import_script gps_baseband.dc_design_load_script -
use_relative_path_to ../2.synthesize_rtl -replace

set_context dft

#Check that it passes the pre-DFT DRC rules
set_dft_specification_requirements -logic_test on
set_drc_handling dft_c9 -auto_fix off

check_design_rules

# Generate patterns.
set spec [create_pattern_specification]
process_pattern_specification

# Run Simulation
set_simulation_library_sources -v ../../../library/standard_cells/
verilog/adk.v
run_testbench_simulations
```

Related Topics

get_design_level

set_current_design

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_design_macros

Context: unspecified, all contexts

Mode: setup

Specifies one or more defined macros for use by subsequent commands.

Usage

set_design_macros *macro_name*[=*macro_value*] … | [[*macro_name* ...] -Clear]

Description

The set_design_macros command gives you the convenience of using macros rather than having to edit the netlist to explicitly add the `define statements to control which parts of `ifdef statements are compiled.

If you run the command multiple times, the tool uses the last set of macros. When you use the command without arguments, the tool reports the list of defined macros, including the default macros TESSENT, MGC_TESSENT, and SYNTHESIS.

Arguments

- *macro_name*[=*macro_value*]

  An optional string that specifies one or more macros.

- [[*macro_name* ...] -Clear]

  An optional string and switch that specifies macros to be cleared. If you specify names of macros before the -clear switch, only those macros are cleared.

  Note

  If you use the -clear switch without a list of named macros, only user-defined macros are cleared. Predefined macros must be named in *macro_name* if you want to clear them.

Examples

Example 1

The following commands specify that the tool use a custom macro named macro_1 set to a value of 20 when loading the top.v module:

```
set_design_macros macro_1=20
read_verilog top.v
```

Example 2

The following example clears one of the predefined macros:

```
set_design_macros SYNTHESIS -clear
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

delete_design

read_cell_library

read_flat_model

read_verilog

set_current_design

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_design_sources

Context: unspecified, all contexts

Mode: setup

Specifies where to search for module definitions during module matching.

Usage

set_design_sources {[-format verilog] {[-V *file_path* ...] [-Y *dir_path* ... -extensions *ext* ...] [-append]} | { [-clear] } }

set_design_sources -format {icl | tcd_bscan | tcd_scan | tcd_fusebox | tcd_memory | tcd_memory_cluster} {[-search_design_load_path {on | off}] [-Y *dir_path* ... -extensions *ext* ...] [-append]} [-clear]

set_design_sources { -format {icl | tcd_bscan | tcd_scan | tcd_fusebox | tcd_memory | tcd_memory_cluster} [-clear]}

Description

Specifies where the tool should look for the definition of undefined modules in the list of files specified by the read_verilog command when -format is specified as or defaults to “verilog”. It specifies where to search for files during module matching for the other formats.

For more information about the module matching process, see the description of the set_module_matching_options command. If the tcd_scan, tcd_memory, tcd_memory_cluster, tcd_bscan, or tcd_fusebox wrappers are located in files with names that do not match the module_name of the Core wrapper, or if they do not have unique extensions for each format, use the read_core_descriptions command to load them in explicitly instead of trying to have them auto-loaded.

The -v, -y, and -extensions switches have similar behavior to standard Verilog compilers such as Questa SIM.

Note

You must issue set_design_sources commands after issuing any read_design command to avoid overwriting your set_design_sources commands.

When you issue the command multiple times, the tool uses only the last set of -v/-y/-extension lists. You can set the -y/-v/-extension list independently of the -search_design_load_path option, but the -y/-v/-extension forms a set for each format.

Table 8-8 shows the default extensions for the various input formats.

Table 8-8. Default Extensions for set_design_sources

Format	Default Extension
icl	icl icl.gz

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-8. Default Extensions for set_design_sources (cont.)

Format	Default Extension
tcd_bscan	tcd_bscan lvbscan tcd_bscan.gz lvbscan.gz
tcd_scan	tcd_scan tcd_scan.gz
tcd_fusebox	tcd_fusebox tcd_fusebox.gz
tcd_memory	tcd_mem_lib tcd_mem_lib.gz
tcd_memory_cluster	tcd_mem_cluster_lib tcd_mem_cluster_lib.gz

Arguments

- -format verilog | icl | tcd_bscan | tcd_scan | tcd_fusebox | tcd_memory | tcd_memory_cluster

  An optional switch that specifies which format the setting applies to. When unspecified, the format verilog is assumed.

  See the description of the set_module_matching_options command for more information about the module matching process that is done using the design sources for formats other than verilog.

- -search_design_load_path on| off

  An optional switch and literal pair that specifies whether the directory into which Verilog and VHDL design modules are loaded or searched are automatically used as valid search directories for the specified format. In cases where the tool adds the Verilog path to the search path, the tool also adds the paths used for the cell libraries. This option is on by default.

- -V *file_path*

  An optional repeatable switch that specifies one or more files in which to search for definitions of undefined Verilog modules. You can specify multiple -v options, multiple -y options, or both. The order of the -v/-y files and directories specified in the command determines the search order during parsing.

- -Y *dir_path* ... -extensions *ext* ...

  An optional repeatable switch that specifies one or more directories in which to search for definitions of undefined modules when -format is defined as or defaults to “verilog”. Any file inside the -y directory should be named as *module_name.extension* in order to be parsed. You can specify multiple -v options, multiple -y options, or both. The order of the -v/-y files and directories specified in the command determines the search order during parsing.

  When you use the -y switch one or many times, you are required to also use the -extensions switch once to instruct the tool to check only the files with the specified extensions. The value is a Tcl list of one or more extension names. The tool supports compressed files by expanding the extension name or list of extension names to include those names with a .gz suffix. That is, if you specify “-extensions v”, the tool expands it to “-extensions v v.gz ”. You can use this switch only once per command execution.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For formats other than Verilog, the directories are searched during the module matching process. See the description of the set_module_matching_options command for more information about the module matching process that is done using the design sources for formats other than Verilog.

- -append

  The -append adds the new settings of set_design_sources to the previous settings of previous set_design_sources.

- -clear

  An optional switch that clears the list of files and directories previously specified with the -v or -y switches. The -clear switch operates on only one format per command execution. If you do not use the -format switch, the -clear switch operates on Verilog files.

Examples

Example 1

To illustrate how the switches work, consider the following example:

top.v	f1.v	directory d1
module top;	module foo();	foo.v bar.v foo.sv zar.v
foo i();	endmodule	
endmodule		
	module bar();	
	endmodule	

When you issue these commands:

```
report_design_sources
set_design_sources -v f1.v
report_design_sources
read_verilog top.v
```

then the module foo in file f1.v is parsed, but not the module bar in f1.v.

When you issue these commands:

```
report_design_sources
set_design_sources -y d1 -extensions v
report_design_sources
read_verilog top.v
```

then the file foo.v is parsed, but not the other files in the d1 directory. Note that file foo.v is parsed but not foo.sv because the specified -extensions is v.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In the previous example, when you issue the following commands:

```
report_design_sources
set_design_sources -y d1 -extensions v -v f1.v
report_design_sources
read_verilog top.v
```

then foo.v in d1 directory is parsed. If the commands are:

```
report_design_sources
set_design_sources -v f1.v -y d1 -extensions v
report_design_sources
```

then module foo in f1.v is parsed.

Example 2

In the following example, the set_design_sources command specifies to look for undefined module definitions in the dir1, dir2, and dir3 directories and to check only those files with a v_ts or v extension.

```
set_design_sources -y {dir1 dir2 dir3} -extensions {v_ts v}
```

Related Topics

delete_design

get_design_sources

read_cell_library

read_icl

read_verilog

report_design_sources

set_current_design

set_design_include_directories

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_dft_cell_selection_mapping

Context: dft

Mode: all

Maps a library cell to one or more power domains.

Usage

set_dft_cell_selection_mapping [*dft_cell_selection_name*]
-power_domain_names *power_domain_name_list*

Description

The set_dft_cell_selection_mapping command specifies a library cell selection to associate with one or more power domains.

Tip

If you generate multiple Tessent IPs (such as an EDT, an OCC, and a SIB) in the same power domain at the top level of your design and want the tool to insert logic using the cell library you specify with the set_dft_cell_selection_mapping command, put all the IPs in the same block.

Arguments

- dft_cell_selection_name

  An optional string that specifies the name of the dft cell selection to map to the specified power domains. If you do not specify a dft_cell_selection_name, the default dft_cell_selection is used.

- -power_domain_names *power_domain_name_list*

  A required switch and list of power domain name attributes that specifies the power domains to be associated with the specified dft_cell_selection.

Related Topics

get_dft_cell_selection_mapping

report_dft_cell_selection_mapping

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_dft_enable_options

Context: dft -logic_bist, patterns -scan, patterns -scan_diagnosis

Mode: setup

Enables or disables static control signals implemented using an IJTAG TDR inside the LogicBIST controller.

Usage

set_dft_enable_options **-type** ***static_enable_signal***
-pin_name ***pin_name*** [-value {on | off}]

Description

In the hybrid TK/LBIST flow, the tool uses this command during fault simulation to specify the static enable signals inside the LogicBIST controller, and during the IP creation phase to specify the creation of static TDR bits inside the LogicBIST controller that control the static enable signals.

Prior to fault simulation, the tool initializes static control DFT signals created during IP insertion to the values required for fault simulation. Refer to "Initial Static DFT Signal Values" in the *Hybrid TK/LBIST Flow User's Manual* for details.

Note

The following considerations are important when using this command:

- Turning off the mcp_bounding_en signal is only supported under certain circumstances as described in "Initial Static DFT Signal Values."
- You cannot turn off X-bounding for LogicBIST because it can result in Xs being captured in the scan cells.
- Turning off the control or observe points enable signal can have an impact on coverage.

Arguments

- **-type** ***static_enable_signal***

  A required switch and literal pair that specifies the static enable signal type. static_enable_signal can have one of the following values:

  xbounding_en

  control_point_en

  clock_gater_control_point_en

  observe_point_en

  mcp_bounding_en

  test_en

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

capture_per_cycle_static_en

- **-pin_name** ***pin_name***

  A required switch and string pair that specifies the name of the pin to enable or disable.

- -value {on | off}

  An optional switch and literal pair that turns on or off the action for the specified pin. This switch is not used during IP creation.

Examples

The following example turns off observe points for transition fault simulation.

```
set_dft_enable_options -type control_point\
   -pin_name /my_core_lbist_i/my_core_lbist_ctrl_signals_i/control_point_en -value on

set_dft_enable_options -type observe_point \
   -pin_name /my_core_lbist_i/my_core_lbist_ctrl_signals_i/observe_point_en -value off

// Warning: Disabling 'observe_point' enable signal may have an impact
// on coverage.
```

The following example generates TDR bits inside the LogicBIST controller and uses them to drive design pins during IP creation:

```
set_dft_enable_options -type xbounding_en -pin_name xbnd_en
set_dft_enable_options -type control_point_en \
      -pin_name [get_pins {core_i*/cp_en core_i*/obs_en}]
                              # A single TDR bit controls both the control point and
                              # observe point enable pins.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_dft_signals_options

Context: dft -rtl, dft -no_rtl

Mode: setup

Specifies certain implementation details for DFT signals.

Usage

set_dft_signals_options **-disable_for_functional_safety** {**off** | **static**}

Description

When creating functional mode fault-tolerant circuitry for test data registers that drive IJTAG static signals, the tool uses the set_dft_signals_options command to infer the dft_signal_disable DFT signal. The tool uses this signal in the Hardware Fault Tolerant (HFT) module. This RTL module provides gating of specified static DFT signals and Single-Event Upset (SEU) monitoring.

Refer to "Functional Mode Fault Tolerance for Static IJTAG Signals" in the *Tessent Shell User's Manual* for complete information.

Arguments

- **-disable_for_functional_safety** {**off** | **static**}

  A required switch and literal pair that turns on (static) or off (off) the creation of gating and monitoring logic for static DFT signals.

  - -disable_for_functional_safety off: By default, no gating or monitoring is implemented. The Test Data Registers fan out directly to functional logic.
  - -disable_for_functional_safety static: The tool gates off static DFT signals designed to interact with functional logic created with the add_dft_signals -create_with_tdr command in the functional mode of design operation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example shows how you can add DFT logic with gating logic:

```
# put gating DFT signal on custom logic
add_dft_signals dft_signal_disable \
    -source_node {fu_sa_inst/fusa_disable_out}

set_dft_signals_options -disable_for_functional_safety static

add_dft_signals capture_per_cycle_static_en \
    -create_with_tdr \
    -disable_for_functional_safety on

#do not gate-off OP_EN
add_dft_signals observe_test_point_en \
    -create_with_tdr \
    -disable_for_functional_safety off

check_design_rules
create_dft_specification

process_dft_specification

# Command to get location of alarm node
get_icl_pins -filter \
    {tessent_dft_function=="functional_mode_alarm"} \
    -of_instance [get_icl_instance -filter \
    {tessent_instrument_type=="mentor::ijtag_node"}]
```

Related Topics

get_dft_signals_option

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_dft_specification_requirements

Context: dft

Mode: setup

Specifies design rule requirements and instrument types.

Usage

set_dft_specification_requirements [-memory_test {on | off}] [-memory_bist {auto | off}]
[-memory_bisr_chains {auto | on | off}]
[-memory_bisr_controller {auto | on | off}]
[-memory_bisr_host_list *host_list_dictionary*]
[-bisr_segment_order_file *file_path*]
[-design_type {standard | tile}]
[-boundary_scan {on | off}]
[-ac_boundary_scan {auto | off}]
[-bsdl_extraction {on | off }]
[-bonding_configuration *BSDL_file_suffix*]
[-logic_test {on | off}] [-add_test_clock_on_ssh {on | off}]
[-tck_injection {on | off | internal_sources_only}]
[-host_scan_interface_type {auto | ijtag | tap | tpsp}]

Description

Specifies the requirements to check during the check_design_rules command and the instrument types to include into the DftSpecification when the create_dft_specification command runs.

Note

The command arguments are not cumulative with multiple calls of the command. Only the most recent command invocation takes effect. The tool resets the DftSpecification settings to the default settings after returning to system mode setup from another system mode.

You typically only need to use the "-memory_test on" option when you want to implement memory test at a given level. When memory_test is on, memory_bist, memory_bisr_chains, and memory_bisr_controller default to auto; otherwise, they default to off. When boundary_scan is on, ac_boundary_scan defaults to auto; otherwise, it defaults to off.

These defaults mean that if the current level either instantiates blocks that contain memory BIST or itself contains memories not already connected to a memory BIST controller, the tool runs the memory BIST pre-DFT DRC rule (Pre-DFT Clock Rules (DFT_C Rules) -DFT_C5) when the check_design_rules command runs. The required specification is added to the DftSpecification when the create_dft_specification command runs.

Licensing

The command arguments you choose cause a license checkout for the selected products.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- -memory_test on | off

  An optional switch and literal pair that specifies whether memory test is to be design rule checked and implemented. Specifying "-memory_test on" is a short cut to setting the three options -memory_bist, -memory_bisr_chains, and -memory_bisr_controller to auto.

  Tip

  Use this switch in conjunction with the -logic_test on | off switch in pre-DFT rule checks to detect potential DFT_C6 DRC violations.

- -memory_bist auto | off

  An optional switch and literal pair that specifies whether memory BIST clocking and, potentially, memory BIST controllers are to be considered. When set to auto, if the current design instantiates blocks containing memory BIST or itself contains memories not already connected to a memory BIST controller, the tool runs the memory BIST pre-DFT DRC rules (Pre-DFT Clock Rules (DFT_C Rules) -DFT_C5) during the check_design_rules command and adds the required specification to the DftSpecification when the create_dft_specification command runs.

- -memory_bisr_chains auto | on | off

  An optional switch and literal pair that specifies whether memory BISR chains are to be inserted or connected in the current level. When set to auto, if the current design instantiates blocks containing memory BISR chains or itself contains repairable memories not already connected to a memory BISR register, the bisr_order_file is validated or created during the check_design_rules command and the required specification is added to the DftSpecification when the create_dft_specification command runs. The bisr_order_file is created taking into account physical placement when a DEF file is read in using the read_def command before running the check_design_rules command. You can also partition the BISR chains into power domains if you read in a UPF or CPF file using the read_upf or read_cpf command before running the check_design_rules command.

- -memory_bisr_controller auto | on | off

  An optional switch and literal pair that specifies whether a memory BISR controller is to be inserted in the current design level. When set to auto, a controller is inserted based on the DefaultsSpecification/DftSpecification/MemoryBisr argument setting relevant to the current design level. The default is ON for the chip level and OFF for both the sub block and physical block levels. You can force the insertion of a controller at the current design level by setting the -memory_bisr_controller option to on.

- -memory_bisr_host_list *host_list_dictionary*

  An optional switch and value pair that specifies the power domain group labels and the secondary host scan chain interfaces for each power domain group. The *host_list_dictionary* value is a Tcl dictionary composed of a power domain group label key followed by a sub-dictionary value listing the secondary host scan chain interfaces for that power domain

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

group. The following shows an example *host_list_dictionary* value that specifies the secondary host scan chain interfaces for two power domain groups, pdgA and pdgB:

```
-memory_bisr_host_list {pdgA {top_1 top_2} pdgB {top bottom}}
```

- DEF File — When you have a DEF file, the tool ignores the ordering of the specified secondary host chain interfaces. Instead, the tool places the secondary host chain interfaces before, after, or between the RAMs on the same power domain group to minimize routing of the scan chain.
- No DEF File — The tool uses the ordering of the secondary host chain interfaces when you do not have a DEF file to provide the position of the bisr_from_si ports. In this case, the tool places the secondary host chain interfaces in the order you specify in the list after the RAMs belonging to the same power domain group.

See Example 3 of the MemoryBISR/SecondaryHostChainInterface reference section for a usage example of this switch with the resulting bisr_order_file.

- -bisr_segment_order_file *file_path*

  An optional switch and value pair that references a MemoryBisr bisr_segment_order_file following the format described in Figure 13-72 on page 4264. By default, a BISR segment order file is created when the check_design_rules command is run if repairable memories, child blocks with BISR chains, or both are present in the current design. If you read a DEF file using the read_def command, this file is automatically created with an ordering extracted from the DEF file in order to minimize routing. If you did not provide a DEF file and want to change the tool default BISR segment order, you can edit and manually order the BISR segment order file and then reference the file using this option. The specified BISR Segment Order File applies to all memories in the design that are processed by the next create_dft_specification command.

  Note

  This option replaces the deprecated -bisr_segment_order_file argument for the set_memory_instance_options command. The set_memory_instance_options -bisr_segment_order argument will become obsolete in a future release.

- -design_type <u>standard</u> | tile

  An optional switch and literal pair that specifies whether the current design is implemented as a block within a tile-based design flow or as a standard design block. The default is a standard design implementation. When you specify tile and the current design contains BISR chains, the tool adds an extra pipeline register at the end of all BISR chains. This pipeline register does not contain the negative edge retiming flip-flop, which simplifies the timing closure of the BISR chains at the parent level of the design. Also, during IJTAG create_dft_specification, the tool places the SIB ring (STI and SRI SIBs) below a “tile collector” SIB, which has its retiming element removed for the same reasons as the BISR chains.

  - For a detailed example of the effect of this switch on the IJTAG network, see Example 3 of IjtagNetwork. Also see the “create_dft_specification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

-tile_ijtag_host_list" command option to see how to create an IJTAG host for the adjacent tiles.

  - Example 3 of MemoryBISR/SecondaryHostChainInterface shows a detailed example of the effect of this switch on the BISR chains.
  - To see the effect of this switch on the embedded boundary scan chains, refer to the C1 and C2 parts of Example 1 in EmbeddedBoundaryScan/HostBScanInterface.
  - The Examples section of the process_top_module_connections command and the TopModuleConnections wrapper show detailed examples on how to create the DFT ports on the tiles and create connections between them at the top level.
  - For a detailed example on how to distribute the SSN network in a tiling design, see Example 3 of the SSN wrapper section.

- -boundary_scan on | off

  An optional switch and literal pair that specifies whether boundary scan is to be design rule checked and implemented. When you set -boundary_scan to on, the -ac_boundary_scan option is automatically set to auto.

  If the design level is not "chip", you can use the set_boundary_scan_port_options command to define the list of ports for which boundary scan is needed.

- -ac_boundary_scan auto | off

  An optional switch and literal pair that specifies whether AC boundary scan is to be inserted. When set to auto, AC boundary scan is inserted as soon as a pad with AC functions is present. When set to off, AC boundary scan (IEEE 1149.6) is not implemented and only IEEE 1149.1 cells are inserted.

- -bsdl_extraction on | off

  An optional switch and literal pair that specifies whether to extract the BSDL file. You cannot use this switch with any other switch except -bonding_configuration.

  See "Extracting the BSDL File" in the *Tessent BoundaryScan User's Manual* for more information.

- -bonding_configuration *BSDL_file_suffix*

  An optional switch and string pair that sets a suffix for the extracted BSDL file. This is useful if, for example, you need to maintain more than one BSDL file for multiple bonding configurations.

- -logic_test on | off

  An optional switch and literal pair that specifies whether logic test is to be design rule checked and implemented. When set to on, the DFT_C2, DFT_C3, DFT_C6, DFT_C8, and DFT_C9 DRC rules are run and the AsyncSetReset auto corrections implemented. Specifying add_dft_control_points -type async_set_reset has no impact unless the DFT_C9 rule is run and the active polarity of the asynchronous set or reset sources extracted by the DRC.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Setting the -logic_test switch to "on" checks any child modules with traceable pre-existing scan chains.

Whenever the current design includes a child physical block with an active scan mode configured with IJTAG, you must first do a DFT insertion pass to connect the IJTAG network to the child block so that the IJTAG network is available to configure the child block in the scan mode that is being traced. Tracing the scan chains in the child block is required to determine the scannable from the non-scannable flip-flops within those blocks during the above-listed DFT_Cx DRCs. During the first DFT insertion pass, you also insert the non-logic test instruments, such as Boundary Scan and Memory BIST, if you need them. See "Tessent Shell Flow for Flat Designs" and "Tessent Shell Flow for Hierarchical Designs" in the *Tessent Shell User's Manual* for more details about the first and second DFT insertion passes.

Tip

Consider using this switch in conjunction with the -memory_test on | off switch in pre-DFT rule checks to detect potential DFT_C6 clock DRC violations. This switch also enables other pre-DFT logic test DRCs that detect controllability issues, such as DFT_C9 for sets and resets.

- -add_test_clock_on_ssh on | off

  An optional switch and literal pair that enables inferring the test_clock signal when you are using SSN and you have child blocks with a test_clock that reaches the boundary of the core. This enables equipping the ScanHost that hosts the dynamic DFT signal to have a test_clock output port by default.

- -tck_injection on | off | internal_sources_only

  An optional switch and literal pair that specifies whether TCK muxing is to be inserted in the current level. When enabled, it identifies clock locations where TCK should be injected to drive child block ports and memories that have use_in_memory_bist_dft_specification set to "on", or "auto" that resolve to "on" in analysis mode. These locations are added to the MemoryBist wrapper of the DftSpecification when the create_dft_specification command is run. Specifying this option to a value other than off requires the -memory_bist option to be set to auto.

  When set to off, the pre-DFT DRC rules DFT_C11, DFT_C12 and DFT_C13 are not run. This value is typically used if you intend to insert OCC in a later insertion pass, or if you do not intend to run memory BIST TCK mode.

  When set to on, the pre-DFT DRC rules DFT_C11, DFT_C12 and DFT_C13 are run and identify all asynchronous, generated and branch clocks in the direct fan-in of memories and blocks for TCK injection. This value is typically used for chip design levels in which you wish to run memory BIST TCK mode by using the tck_clock_only Patterns property.

  When set to internal_sources_only, the pre-DFT DRC rules DFT_C11, DFT_C12 and DFT_C13 are run and identify all generated and branch clocks in the direct fan-in of

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

memories and blocks for TCK injection. This value is typically used for sub_block and physical_block design levels in which you wish to run memory BIST TCK mode though use of the tck_clock_only Patterns property.

Clocks that already have TCK injection are not identified for additional muxing if it is detected that it already exists in the DFT_C11 analysis, even if this option is specified with the on or internal_sources_only values. Existing TCK injection points are only identified if they are controlled by the tck_select DFT signal.

Note

This option should not be used if you intend to add OCCs to your design. The OCCs implement the TCK muxing if the ijtag_host_interface property in the OCC wrapper is not set to "none".

- -host_scan_interface_type {auto | ijtag | tap | tpsp}

  An optional switch and literal pair that specifies the desired host scan interface type. The default is auto.

  When set to auto, the command instructs the create_dft_specification command to populate the IjtagNetwork wrapper with an IJTAG host scan interface when the current design level is sub_block or physical_block, or with a TAP host scan interface when the current design level is chip.

  When set to ijtag, it forces the use of an IJTAG network interface in the DftSpecification even when the current design level is set to chip.

  When set to tap, it forces the use of a TAP host scan interface in the DftSpecification and adds a Tap wrapper, regardless of the current design level.

  When set to tpsp, it forces the use of a Two-Pin Serial Port interface and adds a TwoPinSerialPort wrapper to host a Tap wrapper to the IjtagNetwork DftSpecification.

Examples

The following example specifies that memory test is to be considered in the current DFT insertion pass. Notice that specifying memory test is equivalent to setting the other three related options to auto.

```
set_dft_specification_requirements -memory_test on
```

is equivalent to

```
set_dft_specification_requirements -memory_bist auto \
                                   -memory_bisr_chains auto \
                                   -memory_bisr_controller auto
```

The following example only sets memory BIST to auto which means that even if the current design contains repairable memories, memory BISR chains are not inserted. You can use the "set_memory_instance_options -use_in_memory_bisr_dft_specification" command to control this on a per memory basis.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_dft_specification_requirements -memory_bist auto
```

Related Topics

get_dft_specification_requirement

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_diagnosis_options

Context: patterns -scan_diagnosis

Mode: setup, analysis

Specifies the type of diagnosis performed by the diagnose_failures command.

Usage

set_diagnosis_options
[-Mode { **Auto** | **Scan** | **Chain** | **Iddq** }]
[-Verify_patterns { **ON** [**Noverbose** | **Verbose**] | **OFf** }]
[-Report { **Default** | ***percentage*** }]
[-AT_speed { **ON** | **OFf** }]
[-MAx_suspects { **Default** | ***limit*** }]
[-Time_limit { **OFf** | ***seconds*** }]
[-WALL_time_limit { **OFf** | ***seconds*** }]
[-Pattern_sampling { ***failing_pattern_limit*** | **OFf** } { ***passing_pattern_limit*** | **OFf** }]
[-FAILurefile_mismatch_verbosity { **Default** | ***number*** | **All** }]
[-BRidge_analysis { **ON** | **OFf** }]
[-CHEck_last_shift_value { **ON** | **OFf** }]
[-X2B_mismatch { **Ignore** | **Warning** [**Noverbose** | **Verbose**] | **Error** }]
[-B2X_mismatch { **Ignore** | **Warning** [**Noverbose** | **Verbose**] | **Error** }]
[-CELl_internal_analysis { **ON** | **OFf** }]
[-MAX_Faulty_chains ***failed_chain_threshold***]
[-CHAIN_DIAgnosis_result { **CEll** | **CHain** }]
[-CHAIN_LIB_internal_pathname { **OFf** | **ON** }]
[-ABORT_DIAGNOSE_COMPOUND_faults { **ON** | **OFf** }]
[-ABORT_DIAGNOSE_MANY_faulty_chains { **ON** | **OFf** }]
[-ABORT_DIAGNOSE_MINIMUM_Chain_failing_probability ***number***]
[-ABORT_DIAGNOSE_MINIMUM_Scan_pattern_failing_probability ***number***]
[-REPORT_UNSCORED_CHAIN_suspects { **OFf** | **ON** }]
[-IGNORE_TOOL_version { **OFf** | **ON** }]
[-INCLude_fail_signatures_size { ***maximum_rows_per_table*** | **MAX** }]
[-INCLUDE_BRIDGE_to_power { **ON** | **OFf** }]
[-GRoss_delay { **ON** | **OFf** }]
[-COMPOUND_Hold_time_fault_diagnosis { **ON** | **OFf** }]
[-INCLUDE_DFM_rules { **ON** | **OFf** }]
[-INCLUDE_RCD_constants { **ON** | **OFf** }]
[-EXPected_value { **ON** | **OFf** }]
[-INTERNAL_FAiling_cells { **ON** | **OFf** }]
[-CYcle_offset *offset*]
[-DYNAMIC_Partition master]
[-MISSing_rcd_action { **Ignore** | **Warn** | **Error** }]
[-CELL_PORT_BRIDGE_analysis { Auto | ON | OFf }]
[-CELL_FAULTS *udfm_pathname* [-RCAD ON]]
[-LANDMARK_polygon_limit ***integer***]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

[-INCLUDE_CORE_INSTANce_name { **ON** | **OFf** }]
[-MBFF_TAG_SCI_TEMPlate ***template_string***]
[-MBFF_TAG_SCO_TEMPlate ***template_string***]
[-CHAINDIAG_PATTERN_SAMpling { ***integer*** | **OFf** } { ***integer*** | **OFf** }]
[-CORE_INSTANCE_ERROR { **ON** | **OFF** }]
[-INCLUDE_SRC_SINK_cells_in_marker { **ON** | **OFF** }]
[-ENCRYPT { **OFf** | **ON** }]
[-ENCRYPT_PINnames { **OFf** | **ON** }]
[-REPORT_PIN_LOCations { **ON** | **OFf** }]
[-ADD_SPICE_LAYERS_TO_DIAGNOSIS_REPORT { **ON** | **OFf** }]

Description

The set_diagnosis_options command specifies global settings that determine the behavior of subsequent diagnose_failures commands. These global settings remain in effect until another set_diagnosis_options command is issued.

The set_diagnosis_options command is optional, as there are invocation defaults for all the settings. If no arguments are specified, a list of the current settings displays.

Arguments

- -Mode { **Auto** | **Scan** | **Chain** | **Iddq** }

  A switch and literal pair that determines what type of diagnosis is performed. The following options are available:

  Auto — A literal that enables Tessent Diagnosis to determine what type of diagnosis to run based on the contents of the failure file. If the failure file contains a *chain test* keyword and chain test failures, a chain diagnosis is performed. If the failure file contains scan test failures, a scan diagnosis is performed.

  If the failures include a padding cycle of a scan chain, a chain diagnosis is run.

  If chain failures are not preceded by a *chain test* keyword, use the -faulty_chain switch with the diagnose_failures command to run chain diagnosis.

  If the failure file contains both chain failures and scan failures, you must use the *scan test* and *chain test* keywords to identify the data. This is the default.

  In Auto mode, the tool automatically detects IDDQ failure data from the fail log and runs IDDQ diagnosis.

  Scan — A literal that enables scan diagnosis only. If the failure file contains scan test failures, a scan diagnosis is performed.

  If a padding cycle of a scan chain fails, the scan chain is defective, and the entire chain is masked out before running logic diagnosis. The diagnosis resolution may suffer due to masked chain(s). You should run chain diagnosis in this situation.

  Chain — A literal that enables chain diagnosis only. If the failure file contains a *chain test* keyword and chain test failures, chain diagnosis is performed otherwise the failure file is skipped.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Iddq — A literal that enables IDDQ diagnosis only.

- -Verify_patterns { **ON** [**Noverbose** | **Verbose**] | **OFf** }

  A switch and literal pair that enables or disables verification of the external patterns specified with the diagnose_failures command.

  Note

  The -verify_patterns switch is deprecated. See "verify_patterns" on page 3021 for details and examples of creating, updating, and loading a startup cache.

  Siemens strongly recommends that you do not turn off pattern verification.

  ON — A literal that enables verification of the external patterns. If verification is enabled, the first diagnose_failures command entered simulates the external patterns and compares the capture values with the expected values in the external patterns. When a mismatch occurs, verification stops and an error is issued. Use the report_failures -Pdet command to determine the complete set of failing patterns.

  This verification is performed only once per session for a particular set of patterns.

  Noverbose — When pattern verification is enabled and a failure file mismatch occurs, then the tool does not print out the cell pathname associated with the mismatched bit. When you use this switch, the 'read failure' does not produce the cell paths either.

  Verbose — When pattern verification is enabled and a failure file mismatch occurs, then the tool prints out the cell pathname associated with the mismatched bit. This is the default.

  OFf — A literal that disables verification of the external patterns.

  Note

  Do not turn off pattern verification to achieve run time reduction. Test pattern verification is required for accurate diagnosis results. You can reduce execution time by using a startup cache.

- -Report { **Default** | ***percentage*** }

  A switch and value pair that changes the diagnosis report generated by the tool. You have the following two report choices:

  Default — A literal that specifies to report the following for each symptom:

  - All suspects with score greater than or equal to 80.
  - Up to top three suspects even if the suspects' scores are less than 80.
  - If the suspect beyond the top three has the same score with the 3rd suspect, the tool reports the suspect only if:
    - The suspect has same score as suspect #3, and the suspect can explain same or higher number of failing patterns.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- In the case the suspect has same score and fail_match, the suspect is reported only if it has the same passing pattern mismatch.

percentage — An integer in the range 1 to 100 that specifies a percentage of the suspects the tool should report for each symptom. If you specify 75, for example, and there are four suspects for a particular symptom, the tool reports the top 75 percent (top three) of them.

Note

For definitions of the terms "symptom" and "suspect", refer to the *Tessent Glossary*.

- -AT_speed { **ON** | **OFf** }

  A switch and literal pair that specifies performing at-speed failure diagnosis. See "At-Speed Failure Diagnosis" in the *Tessent Diagnosis User's Manual* for complete information, including the conditions Tessent Diagnosis uses when diagnosing at-speed failures.

- -MAx_suspects { **Default** | ***limit*** }

  A switch and value pair limiting the number of suspects per symptom in the diagnostics report. Choose one of the following options:

  Default — A literal that specifies reporting 100 suspects per symptom.

  limit — An integer that specifies to report this number of suspects per symptom.

- -Time_limit { **OFf** | ***seconds*** }

  A switch and value pair that specifies a time limit, in seconds, for diagnosing a single failure file. This time includes the failure file verification time, but not the pattern verification time. By default, the test patterns and each failure file is verified for consistency when a diagnosis is run.

  If the specified time limit is reached before completion of the diagnosis, the diagnosis is aborted. A "Diagnosis aborted due to time limit (<time>)" message displays and no suspects are reported.

- -WALL_time_limit { **OFf** | ***seconds*** }

  A switch and value pair that specifies a wall time limit, in seconds, for diagnosing a single failure file. When you specify a wall time limit (***seconds***) and a diagnosis job exceeds the limit, then the tool aborts the diagnosis run with an error message.

  Note

  The wall time limit is independent of the CPU time limit (using the -Time_limit switch). If you set to both the CPU and wall time limit, then the tool aborts the diagnosis when either limit is reached.

  If the wall time limit is reached during diagnosis, the tool produces an incomplete diagnosis report file. If you specify the diagnose_failures command with the -csv switch, then the tool also creates an empty CSV file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Pattern_sampling { ***failing_pattern_limit*** | **OFf** } { ***passing_pattern_limit*** | **OFf** }

  Optional switch with two values that specifies pattern limits for logic diagnosis. The first value is for the failing pattern limit, and the second value is for the passing pattern limit. The default is off for both values.

  Use this switch to limit the patterns used for logic diagnosis and reduce run time. By default, all failing and passing patterns are used for diagnosis. To ensure adequate diagnosis resolution, the pattern sample must be 32 or greater.

  Logic diagnosis patterns with several capture cycles require more simulation time and have less resolution. To avoid losing resolution, Tessent Diagnosis selects patterns with the least amount of capture cycles for the diagnosis. Pattern sampling options include:

  failing_pattern_limit — Must be an integer of 32 or higher.

  passing_pattern_limit — Must be an integer of 32 or higher. Specified number of passing patterns are selected from the first passing patterns in the set.

- -FAILurefile_mismatch_verbosity { **Default** | ***number*** | **All** }

  A switch and value pair that controls the number of pattern mismatches displayed during the verification of the failure file. You have the following report choices:

  Default — A literal that specifies to display 20 failing patterns. This is the default.

  number — An integer that specifies to report this number of failing patterns.

  All — A literal that specifies to report all failing patterns.

- -BRidge_analysis { **ON** | **OFf** }

  A switch and literal pair that determines if diagnosis includes an analysis to identify bridge suspects. A bridge suspect is identified when faulty behavior is observed for the implicated nets and the appropriate aggressor-victim conditions are met. Currently 2-way and 3-way bridges are considered and this switch is applicable to both.

  Options include:

  ON — Bridge analysis is enabled. Default setting.

  OFf — Bridge analysis is disabled.

  There are two cases where it may be desirable to disable bridge analysis.

  - **Case 1** — Because the analysis is currently based only on logical responses, it is possible that a bridge suspect may not be physically possible. In cases where bridge suspects are verified as not physically possible, you may want to rerun the diagnosis with bridge analysis turned off.
  - **Case 2** — In cases where very few failing patterns are captured for a failing device, and bridge analysis may return a large number of suspects, you can disable the bridge analysis to reduce the number of suspects.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -CHEck_last_shift_value { **ON** | **OFf** }

  An optional switch and literal pair that controls whether last shift values are considered for transition launching in resistive open diagnosis as well as cell internal diagnosis.

  ON — Enables last shift values consideration for transition launching in resistive open diagnosis. This is the default.

  OFf — Disables considering last shift values for transition launching in resistive open diagnosis. Use this option if you believe that the shift is very slow speed such that the transition launched from last shift is irrelevant.

- -X2B_mismatch { **Ignore** | **Warning** [**Noverbose** | **Verbose**] | **Error** }

  A switch and literal pair that determines how X2B mismatches are handled during diagnosis. X2B mismatches occur during pattern verification when the simulator returns binary values instead of the expected X values.

  The tool most likely introduces X2B mismatches because it removes cell constraints when performing pattern verification and diagnosis. This enables the tool to perform diagnosis using one flat model for potentially multiple modes, each of which have different constraints, or to use flat models that were written at different times. You can ignore these X2B mismatches.

  Options include:

  Ignore — This is the default. Does not display X2B mismatch information. Pattern verification passes.

  Warning Noverbose — Warns at the end of pattern verification of any X2B mismatches. Pattern verification can still pass.

  Warning Verbose — Prints during pattern verification any mismatched X2B bits. Pattern verification can still pass.

  Error — Prints the X2B mismatches for the first mismatched pattern. Pattern verification fails, which terminates the diagnosis or verify_patterns command.

- -B2X_mismatch { **Ignore** | **Warning** [**Noverbose** | **Verbose**] | **Error** }

  A switch and literal pair that determines how B2X mismatches are handled during diagnosis. B2X mismatches occur during pattern verification when the simulator returns X values instead of the expected binary values. These mismatches could be introduced by different problems, for example, software enhancements to the simulator or using a mask file when reading patterns.

  Options include:

  Ignore — Does not display B2X mismatch information. Pattern verification passes.

  Warning Noverbose — This is the default. Warns at the end of pattern verification of any B2X mismatches. Pattern verification can still pass.

  Warning Verbose — Prints during pattern verification any mismatched B2X bits. Pattern verification can still pass.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Error — Prints the B2X mismatches for the first mismatched pattern. Pattern verification fails, which terminates the diagnosis or verify_patterns command.

- -CELl_internal_analysis { **ON** | **OFf** }

  An optional switch and literal pair that determines if the tool performs analysis to diagnose defective library cells and distinguish them from defects on nets interconnecting library cells.

- -MAX_Faulty_chains ***failed_chain_threshold***

  An optional switch and integer pair that specifies the maximum number of failing scan chains within a single datalog that are diagnosed using chain fault model. When a failure log has multiple failed_chains, there can either be multiple defects in scan chains, or a single defect affecting the scan control signals. The default *failed_chain_threshold* is 2.

  - If there are one or two failed_chains, then the tool assumes the defect is on each individual chain and uses the scan chain fault model to run diagnosis. This is the default behavior of the tool.
  - If failed_chains is greater than two, then the tool assumes the defect is on a global control signal. If "set_diagnosis_options -ABORT_DIAGNOSE_MANY_faulty_chains" is OFf, it uses the clock/scan_enable fault model to run diagnosis. Otherwise, the tool aborts diagnosis for this case.

  Changing *failed_chain_threshold* from the default value (2) causes the following effects:

  - If failed_chains is greater than *failed_chain_threshold*, then the tool assumes the defect is on a global control signal. If "set_diagnosis_options -ABORT_DIAGNOSE_MANY_faulty_chains" is OFf, it uses the clock/scan_enable fault model to run diagnosis. Otherwise, the tools aborts diagnosis for this case.
  - If there are one, two, ... *failed_chain_threshold* faulty chains, then the tool assumes the defect is on each individual chain and uses the scan chain fault model to run diagnosis on each of the defects.

  For example, a failure file contains six failed_chains. You can diagnose each of these six failed-chains as individual defects by setting this switch as follows:

  ```
  set_diagnosis_options -max_faulty_chains 6
  ```

  During chain diagnosis, if the tool reaches the max faulty chains limit, by default the diagnosis report includes information about the failing chains but not the failing scan cells. Suppose a failure file contains two failing scan chains and the -max_faulty_chains option is

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set to 1. The resulting diagnosis report displays information for the two failing chains. For example:

```
#total_symptoms=2 #total_suspects=0 total_CPU_time=0.00sec

diagnosis_mode=chain

#symptoms=2 #suspects=0  CPU_time=0.00sec  fail_log=flog1
symptom=1   #suspects=0   faulty_chain=chain1   fault_type=STUCK
symptom=2   #suspects=0   faulty_chain=chain3   fault_type=STUCK
```

- -CHAIN_DIAgnosis_result { **CEll** | **CHain** }

  Optional switch and literal pair that determines if a preliminary high-level chain diagnosis is performed versus a standard diagnosis. A preliminary high-level chain diagnosis only reports one failure per chain; the dominant failure.

  Options include:

  CEll — Runs the standard diagnosis of chain plus scan patterns and chain plus scan failures to determine cell defects. Default setting.

  CHain — Runs a preliminary high-level diagnosis of the chain failures in the failure file and reports the name, length, and fault type of the chains that failed.

- -CHAIN_LIB_internal_pathname { **OFf** | **ON** }

  An optional switch and literal pair that reports the internal instance pathname within the library for each scan latch in a suspect scan cell.

  When set to ON, the tool reports this information in the chain diagnosis report's "lib_internal_pathname" column. If there is no instance name, the tool reports "". If you specify the diagnose_failures command with the -csv switch, then the tool also includes this information in the CSV output.

  See "Chain Diagnosis Section" in the *Tessent Diagnosis User's Manual* for complete information.

- -ABORT_DIAGNOSE_COMPOUND_faults { **ON** | **OFf** }

  An optional switch and literal pair that specifies whether or not the tool diagnosis the logic part of compound defects during chain diagnosis. The default to "ON", meaning the tool does not diagnose the logic portion of compound defects.

  See "Compound Diagnosis Abort Logic Diagnosis Part" in the *Tessent Diagnosis User's Manual* for a complete discussion.

- -ABORT_DIAGNOSE_MANY_faulty_chains { **ON** | **OFf** }

  An optional switch and literal pair that instructs the tool to skip diagnosing when #faulty_chains is larger than the *failed_chain_threshold* specified with the set_diagnosis_options -MAX_Faulty_chains switch. The default is "ON".

  When you set this option to OFF and the number of faulty chains it greater than specified, the tool does not report each faulty chain.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

See "Abort Conditions for Chain Diagnosis" in the *Tessent Diagnosis User's Manual* for a complete discussion.

- -ABORT_DIAGNOSE_MINIMUM_Chain_failing_probability ***number***

  An optional switch and integer pair that specifies the minimum chain fail probability for the tool to abort chain diagnosis. The default is 1. The minimum is 0, which means no abort. The maximum value you can specify is 100.

  See "Abort Conditions for Chain Diagnosis" in the *Tessent Diagnosis User's Manual* for a complete discussion.

- -ABORT_DIAGNOSE_MINIMUM_Scan_pattern_failing_probability ***number***

  An optional switch and floating point pair that specifies the minimum scan pattern fail probability for the tool to abort chain diagnosis. The default is 0.1. The minimum is 0, which means no abort. The maximum value you can specify is 100.

  See "Abort Conditions for Chain Diagnosis" in the *Tessent Diagnosis User's Manual* for a complete discussion.

- -REPORT_UNSCORED_CHAIN_suspects { **OFf** | **ON** }

  An optional switch and literal pair that instructs the tool to report all the cells in the failing scan chain. The default is OFf.

  In some cases, chain diagnosis is unable to produce a diagnosis suspect within the identified failing chain. For those cases, by default only the failing chain is reported (that is, no suspects are reported). Using the switch -report_unscored_chain suspects on, you can direct the tool to report all the cells in the failing scan chain. In this scenario, all of these cells in the failing chain are reported but are unscored and therefore have a score of 0 in the diagnosis report.

- -IGNORE_TOOL_version { **OFf** | **ON** }

  An optional switch and literal pair that instructs the tool to ignore the tool version when loading a startup cache. If the tool versions are different—that is, the startup cache was created by an older version—the tool issues a warning. By default, this option functions as follows with the following verify_patterns options:

  - With -create_startup_cache: Has no impact. The tool creates the same startup cache whether -ignore_tool_version is on or off.
  - With -update_startup_cache: Causes a conflict error.
  - With -load_startup_cache: Disables the tool version check and uses the startup cache regardless of the tool version used to generate or update the cache.

  When set to OFF, the tool issues an error and does not load the startup cache if the tool version used to create the startup cache is older than the current tool version.

- -INCLude_fail_signatures_size { ***maximum_rows_per_table*** | **MAX** }

  An optional switch and integer pair that specifies the maximum number of rows per table for reporting failure signature information in the diagnosis report. The default is 256. If you do

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

not want the failure signatures included in the diagnosis report, then specify 0 (zero) with this switch. Conversely, if you require all signature information, then use the keyword MAX with this switch. See "Failure Signature Information in the Diagnosis Report" in the *Tessent Diagnosis User's Manual* for complete information.

- -INCLUDE_BRIDGE_to_power { **ON** | **OFf** }

  An optional switch and literal pair that specifies performing analysis and reporting of potential bridging defects between the STUCK suspect net, and a power or ground line during layout-aware diagnosis. The default is "ON".

  See "Power and Ground Bridge Reporting" in the *Tessent Diagnosis User's Manual* for complete information.

- -GRoss_delay { **ON** | **OFf** }

  An optional switch and literal pair that specifies performing analysis and reporting of gross delay defects. The default is "OFf".

  See "Gross Delay Defect Diagnosis" in the *Tessent Diagnosis User's Manual* for complete information.

- -COMPOUND_Hold_time_fault_diagnosis { **ON** | **OFf** }

  An optional switch and literal pair that specifies performing analysis and reporting of slow clock compound hold-time defects. The default is "OFf". This option only applies to hold times on shift paths, not capture paths.

  See "Slow Clock Compound Hold-Time Diagnosis" in the *Tessent Diagnosis User's Manual* for complete information.

- -INCLUDE_DFM_rules { **ON** | **OFf** }

  An optional switch and literal pair that specifies to perform DFM hit reporting. The default is "ON".

  See "Diagnosis for Design for Manufacturability Analysis" in the *Tessent Diagnosis User's Manual*.

- -INCLUDE_RCD_constants { **ON** | **OFf** }

  An optional switch and literal pair that specifies to populate the LDB with RCD constants. The default is "ON".

  See "Diagnosis for Root Cause Deconvolution Analysis" in the *Tessent Diagnosis User's Manual*.

- -EXPected_value { **ON** | **OFf** }

  An optional switch and literal pair that specifies whether to verify expected pattern values against actual pattern values and return the results in the failure log. When set to OFF, the tool suppresses this check and ignores any missing expected values.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Caution

The -expected_value switch turns off failure log verification, which means that it is unknown whether there is a discrepancy between the failure log and the patterns. Use this switch with extreme caution.

- -INTERNAL_FAiling_cells { **ON** | **OFf** }

  An optional switch and literal pair that specifies whether to enable internal scan cell profiling. See "Internal Scan Cell Profiling for Compressed Patterns" in the *Tessent Diagnosis User's Manual*.

- -CYcle_offset *offset*

  An optional switch and integer pair that specifies to adjust the cycles in a cycle-based failure file by the value *offset*. Tessent Diagnosis uses the adjusted cycles for failure verification and diagnosis. See "Cycle Offset Adjustment for Failure Files" in the *Tessent Diagnosis User's Manual*.

- -DYNAMIC_Partition master

  An optional switch and literal pair that specifies to generate a statup cache for dynamic partitioning-based diagnosis. You must specify this command before you load the test patterns. It signals the tool to extract clock signatures during read_patterns.

  You must use this option for dynamic partitioning-based diagnosis. Tessent Diagnosis supports startup caches generated using this option for all diagnosis uses.

- -MISSing_rcd_action { **Ignore** | **Warn** | **Error** }

  An optional switch and literal pair that checks whether the LDB is populated with RCD constants. The tool checks the LDB at the beginning of the diagnosis run and performs one of the following actions when the RCD constants for the current flat file and pattern set are not present:

  Ignore — Takes no action and diagnosis proceeds.

  Warn — Issues a warning and diagnosis proceeds. This is the default.

  Error — Issues an error and diagnosis halts.

  Use this switch to ensure that the LDB contains the RCD constants required for RCD analysis with Tessent YieldInsight. The warning and error outputs remind you to run create_feature_statistics, as needed, to generate the RCD constants.

- -CELL_PORT_BRIDGE_analysis { Auto | ON | OFf }

  An optional switch and literal pair that specifies whether to perform cell port bridge diagnosis. By default, the tool does not perform cell port bridge diagnosis for layout-aware diagnosis but does perform cell bridge diagnosis for cell-aware diagnosis. For more information, refer to "Cell Bridge Port Diagnosis Reporting."

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -CELL_FAULTS *udfm_pathname*

  An optional switch and string pair that specifies a UDFM (.udfm) file generated by Tessent CellModelGen, which is required for performing cell-aware diagnosis. For details, refer to "Cell-Aware Diagnosis" in the *Tessent Diagnosis User's Manual*.

  When specified, the tool performs a series of validation checks to ensure that within Tessent CellModelGen, the UDFM file was generated using the proper switches and values, and issues warnings if this is not the case. In addition, for flat models, the tool verifies the flat model against the UDFM and issues a warning if the flat model contains cells not defined in the UDFM file. For example:

  ```
  // Warning: Udfm file is missing definitions for 22 cells used
  1605118 times in the design (out of a total of 10345712 cells in the
  design). This may reduce accuracy of cell aware diagnosis.
  ```

- -RCAD ON

  An optional switch that enables RCAD constant creation and performing cell-aware diagnosis that includes the RCAD constants. Specify this switch with the -cell_faults switch. For details, refer to "Performing Cell-Aware Diagnosis With RCD" in the *Tessent Diagnosis User's Manual*.

- -LANDMARK_polygon_limit ***integer***

  An optional switch and integer that controls the number of landmark polygons that are written. The tool uses a proximity-to-defect algorithm to limit the number of landmark polygons for global signals such as power, ground, and scan enable. Specify an integer greater than 99. The default is 10000.

- -INCLUDE_CORE_INSTANce_name { **ON** | **OFf** }

  An optional switch and literal pair that specifies whether to generate the layout-aware hierarchical diagnosis report with chip-level pin and net names rather than the default core-level names. For more information, refer to "Running Layout-Aware Diagnosis Using a Core-Level LDB".

- -MBFF_TAG_SCI_TEMPlate ***template_string***

  An option and string pair that specifies to set the default internal pin tag naming convention for SCI pins for elements inside multi-bit flip-flops in the format "*template_string N*", where *N* is the index of a master element within the multi-bit flip-flop, starting with 1. For usage details, refer to "Multi-Bit Flip-Flop Handling" in the *Tessent Diagnosis User's Manual*.

- -MBFF_TAG_SCO_TEMPlate ***template_string***

  An option and string pair that specifies to set the default internal pin tag naming convention for SCOs for elements inside multi-bit flip-flops in the format "*template_string N*", where *N* is the index of a master element within a multi-bit flip-flop, starting with 1. For usage details, refer to "Multi-Bit Flip-Flop Handling" in the *Tessent Diagnosis User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -CHAINDIAG_PATTERN_SAMpling { ***integer*** | **OFf** } { ***integer*** | **OFf** }

  An optional switch and integer pair that specifies to limit diagnosis analysis to the specified number of failing and passing patterns, where the first integer is the number of failing patterns and the second integer is the number of passing patterns. The minimum value for *integer* is 32. This option reduces chain diagnosis run time with a potential minor impact on the quality of the diagnosis results. You can disable sampling for either failing or passing patterns.

- -CORE_INSTANCE_ERROR { **ON** | **OFf** }

  An optional switch and literal pair that specifies whether core name and core instance path mismatches between the tracking information in the failure file and an LDB are handled as warnings or errors. This option is used for hierarchical diagnosis.

  The default is on, in which mismatches are handled as errors. In addition, when opening an LDB that contains core instances, the layout fails to open with an error if the -chip_design_name option is unspecified.

- -INCLUDE_SRC_SINK_cells_in_marker { **ON** | **OFf** }

  An optional switch and literal pair that specifies whether to include source/sink cell Open suspect information in layout marker files. See "Source/Sink Polygon Layout Markers for Open Diagnosis Suspects" in the *Tessent Diagnosis User's Manual*.

- -ENCRYPT { **OFf** | **ON** }

  An optional switch and literal pair that specifies to encrypt all diagnosis files. The default value is off which means that encryption is turned off. The value of this switch (off or on) is stored in the flat model. If a flat model is saved with encryption turned on, encryption cannot be turned off in that flat model.

- -ENCRYPT_PINnames { **OFf** | **ON** }

  An optional switch and literal pair that allows pin names to remain unencrypted when encryption is active in the flat model as previously specified by "set_diagnosis_options -encrypt on". By default, pin names are also encrypted.

- -REPORT_PIN_LOCations { **ON** | **OFf** }

  An optional switch and literal pair that specifies whether to include a PIN_LOCATION table in the XMAP table of a layout-aware diagnosis report. The default is off, which does not print the table.

- -ADD_SPICE_LAYERS_TO_DIAGNOSIS_REPORT { **ON** | **OFf** }

  An optional switch and literal pair that specifies whether to include a spice_layer column in the CELL_DEFECT_LOCATION table of a layout-aware diagnosis report. The default is off, which does not print the column. Turn on this switch and enable cell-aware diagnosis to add the spice_layer column to the report.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example loads patterns from two files into the external pattern database for the tool. The failure file, *scan_failure_file*, contains just a scan test section, so the tool only performs a scan diagnosis for that file.

read_patterns pat_file1

read_patterns pat_file2 -append

set_diagnosis_options -mode scan

diagnose_failures scan_failure_file -output scan_diag.ascii -csv scan_diag.csv

Example 2

The following example runs a preliminary diagnosis on my_failure_file.

read_patterns pat_file1

set_diagnosis_options -chain_diagnosis_result chain

diagnose_failures my_failure_file

Example 3

The following example shows a preliminary diagnosis report.

```
tracking_info_begin
tracking_info_end
#faulty_chains=2 #symptoms=0 #suspects=0 CPU_time=0.00sec
fail_log=fail_logs/my_failure_file
Chain chain1 has error, fault type is STUCK_AT_0, the length of this chain
is 102.
Chain chain3 has error, fault type is STUCK_AT_1, the length of this chain
is 101.
```

Example 4

The following example sets the passing pattern limit to 64.

set_diagnosis_options -pattern_sampling off 64

Related Topics

diagnose_failures

report_failures

write_failures

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_display

Context: unspecified, all contexts

Mode: all modes

Sets the DISPLAY environment variable from the tool's command line.

Usage

set_display ***display_name***

Description

The set_display command sets the DISPLAY environment variable to display_name without exiting the currently running application. If you invoke the tool in command line mode (the default), then the DISPLAY variable is not required in order to use most commands successfully. If the variable is not set, however, and you issue a command such as open_visualizer that requires a GUI, the tool is unable to show the applicable window. This is because the display connection provided by the variable is absent. The set_display command enables you to set the variable, if needed for the display, without the inconvenience of exiting to the shell in order to set it.

Note

This command affects the DISPLAY setting within the currently running application only. When you exit the tool, the setting in the invocation shell is what it was when you invoked the tool.

Arguments

- ***display_name***

  A required string that specifies a valid display setting for the machine on which the tool is running.

Examples

The following example sets the DISPLAY variable when "open_visualizer" fails to open the Tessent Visualizer GUI on the local display. The example also uses the system command to pass a Linux "echo $DISPLAY" command to the shell in order to check the variable's setting.

open_visualizer

```
//  Note: Tessent Visualizer server started on host 'my_workstation'
//  and listening on port 12345.
//        Client authorization code: 12345.
//  Error: Failed to connect to an X display (DISPLAY environment
//  variable not set).
```

system echo $DISPLAY

set_display my_workstation:0.0
system echo $DISPLAY

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
my_workstation:0.0
```

Related Topics

system

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_drc_handling

Context: unspecified, all contexts

Mode: all modes

Specifies how the tool handles design rule violations.

Usage

```
set_drc_handling {drc_id [Error | Warning | NOTe | Ignore]
    [NOVerbose | Verbose]
    [NOAtpg_analysis | Atpg_analysis]   // C{1,3,4,5,6}, D6, E{4,5,8,10,11,12}
    [ATPGC]                             // E10
    [-Mode A clk_name]                  // C3, C4
    [-Interval number]                  // C3, C4
    [-CONSERvative {ON | OFf}]          // C6, D5
    [-IGNORE_SYNChronous_clock_groups {ON | OFf}]  // C6
    [-SKip_procedure {TEst_setup | OFf}]           // E4
    [-TIMing_exception_check {ON | OFf | Auto}]    // E5
    [-CONSIder_primary_outputs_observable {ON | OFf}] // E5
    [-MASk_interactions {ON | OFf }]               // C22
    [-INteractions {edge | any}]                   // C22
    [-PIS_AS_SEquential_elements {ON | OFf} ]      // C27
    [-Mode {Sequential | Combinational}]           // E10
    [-SIMULATE_ACROSS_Internal_primary_inputs {ON | OFf}]
    [-auto_fix]}
    |-ALLOW_COMmon_scan_cells_in_multiple_chains {ON | OFf}
    |-SCan_chain_tracing {Aggressive | Conservative}
```

Description

Caution

The purpose of DRCs is to protect against patterns failing on silicon, and to preserve diagnosis accuracy and resolution. The Tessent tools enable you to downgrade some DRCs. You should only downgrade the tool handling of a DRC as a last resort, with a complete understanding of the hazards involved.

The set_drc_handling command specifies the handling of the messages for the following design rule checks:

- Scan cell RAM rules (A rules)
- Some BIST rules (B2 and B4 rules)
- Clock rules (C rules)
- Some pre-DFT clock rules (DFT_C Rules)
- Custom DRC rules that you have registered using the register_drc command

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Data rules (D rules)
- Extra rules (E rules)
- EDT Finder rules (F rules)
- Some ICL semantic rules (ICL rules)
- Some EDT rules (K rules except for K15)
- Some procedure rules (P rules: P29-P33, P86, P89, P90, P92, and P94)
- Some scan pattern retargeting rules (R rules)
- Some scannability rules (S rules: S3 and S5)
- Trace rules (T rules)
- Some power-aware rules (V rules)
- Some timing rules (W rules)

You can specify that the violation messages for these checks be either error, warning, note, or ignore. If you do not specify error, warning, note, or ignore, then the tool uses the default handling or the most recent override.

Note

The set_drc_handling command does not support the scannability (S) rules except for the S3 and S5 rules. Unlike the other DRC rules, setting the handling type to “Error” does not enable the transcript verbosity for the S3 rule. The verbosity can only be turned on via the optional literal “verbose”.

The set_drc_handling command does not support any Flattening design rules (FN, FP, and FG rules). Use the set_flattener_rule_handling command to specify how design rule violations are handled for the Flattening rules.

Each rule violation has an associated occurrence message and summary message. The tool only displays the occurrence message for either error conditions or if you specify the Verbose option for that rule. The tool also displays the rule identification number in all rule violation messages.

The Atpg_analysis option provides test generation analysis when performing rules checking for some clock rules (C), for some data rules (D), and for some extra rules (E). For example, if Atpg_analysis is enabled for clock rule C1, and the tool simulates a clock input as X, the rule violation occurs when it is possible for the test generator to create a test pattern while that clock input is on, all defined clocks are off, and all constrained pins are at their constrained state.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

When Atpg_analysis is enabled, the tool requires some additional CPU time and memory to perform the test generation analysis. (The Atpg_analysis option is enabled by default for rules C1, E10, E11 and E13; you can disable it for these rules by specifying the NOAtpg_analysis option.)

Arguments

- ***drc_id***

  A required non-repeatable literal that specifies the identification of the exact design rule violation whose message handling you want to change.

  For more information on the design rules, refer to "Design Rule Checking".

- Error

  An optional literal that specifies for the tool to both display the error occurrence message and immediately terminate the rules checking.

- Warning

  An optional literal that specifies for the tool to display the warning summary message indicating the number of violations for that rule. If you also specify the Verbose option, the tool displays the occurrence message for each occurrence of the rule violation.

- NOTe

  An optional literal that specifies for the tool to display the summary message indicating the number of violations for that rule. If you also specify the Verbose option, the tool also displays the occurrence message for each occurrence of the rule violation

- Ignore

  An optional literal that specifies for the tool not to display any message for the rule's violations. The tool must still check some rules and they must pass to enable later performance of certain functions.

- NOVerbose

  An optional literal that, for a DRC whose handling is set to Warning or Note, specifies to display a summary message indicating the total number of occurrences of the rule violation. This is the default.

  If DRC handling is set to Error or Ignore, the tool behaves as follows regardless of whether you specify the NOVerbose (or Verbose) option:

  - Error — The tool stops at each error and displays detailed instance-specific violation information to help you debug it.
  - Ignore — The tool does not perform the check. The consequences of the check on ATPG, whether good or bad, do not occur. Also, the tool does not display or store instance-specific violation information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Verbose

  An optional literal that, for a DRC whose handling is set to Warning or Note, specifies to display the occurrence message for each occurrence of the rule violation. All instance-specific violations (maybe hundreds or thousands) are listed, one message per instance violating the rule. To obtain a summary message instead, use the NOVerbose option.

  If the DRC's handling is set to Error or Ignore, the tool behaves as follows regardless of whether you specify the Verbose (or NOVerbose) option:

  - Error — The tool stops at each error and displays detailed instance-specific violation information to help you debug it.
  - Ignore — The tool does not perform the check. The consequences of the check on ATPG, whether good or bad, do not occur. Also, the tool does not display or store instance-specific violation information.

- NOAtpg_analysis

  An optional literal that specifies for the tool not to use test generation analysis when performing rules checking. This is the default except for rules C1, E4, E10, E11 and E13.

- Atpg_analysis

  An optional literal that specifies for the tool to use test generation analysis when performing rules checking for clock rules (such as C1, C3, C4, C5 and C6), some D rules (such as D6) and some E rules (such as E4, E5, E8, E10, E11 and E12). This is the default for rules C1, E4, E10, E11 and E13.

  For clock rules C3 and C4, the Atpg_analysis option generates a check of the clocks of the source and sink to see if they are gated off.

  Note

  To use the constraint values during the D6 rule analysis, you must use the Atpg_analysis option.

- ATPGC

  An optional literal that specifies for the design rules checker to use all the current ATPG constraints when analyzing E10 rule violations. You can also use the "add_atpg_constraints -Static" command to do the same thing.

- -Mode A *clk_name*

  A switch, a literal (A), and a string triplet that specifies the name of a clock on which you want the tool to perform further analysis to screen out false C3 and C4 clock rules violations. The tool performs this extra analysis on one clock only.

  For more information on using the -Mode A option, refer to the C3 and C4 descriptions.

- -Interval *number*

  An optional switch and integer pair that you can only use with C3 and C4 clock violations to specify how often you want the tool to display a message during the ATPG analysis of those

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

violations. The *number* argument indicates multiples of violation occurrences that cause the tool to display a message. The default is 0.

The message includes the number of sequential elements that the tool checked, the number of sequential elements remaining to check, the current number of ATPG passes during the C3 or C4 clock rules checking, and the current CPU time used by the tool for clock rules checking.

The value of the *number* parameter must be either zero or a positive integer. You can only specify one *number* value that the tool uses for both the C3 and C4 violations. If you issue multiple set_drc_handling commands (one for C3 and one for C4) that specify different values for the *number* argument, the tool uses the last interval value you specified.

- -CONSERvative {ON | OFf}

  An optional switch and literal for C6 and D5 DRC analysis. The default is OFf.

  C6 DRC analysis:

  The C6 DRC may miss some violations when using "set_clock_restriction OFf". Setting -CONSERvative ON makes the tool run a more conservative analysis to identify these C6 violations. Siemens EDA generally recommends using "set_drc_handling C6 -CONSERvative ON" instead of turning off clock restrictions.

  D5 DRC analysis:

  By default, the D5 DRC assumes the clock port is faster than the data port and that any cross-domain paths are synchronous. Setting -CONSERvative ON makes the tool run a more conservative analysis to avoid simulation mismatches due to a C6 clock race condition or an asynchronous clock domain skew. Report the gates having these D5 races using the command "report_gates -type race_D5" while in analysis mode.

  Note

  Turning -CONSERvative ON for D5 DRC analysis may decrease test coverage due to an increase in the number of C6 DRC violations detected.

  When this switch is ON, the tool may change a D5 non-scan cell's classified type to Init-X to avoid simulation mismatches. In that case, when switching the system from setup to analysis mode, the tool prints a message such as the following:

  ```
  //  4 D5 cells have been changed to Init-X to prevent mismatches
  //  due to cross clock domain or C6 race.
  ```

  When the switch is OFf, the tool reports any D5 race condition having a known initial value, but does not change the classified type determined by the DRC simulation of the test procedures. In this case, the tool prints a message such as the following:

  ```
  //  4 D5 cells with potential race due to cross clock domain
  //  or C6 violation
  ```

- -IGNORE_SYNChronous_clock_groups {ON | OFf}

  An optional switch and literal for the C6 DRC analysis. The default is OFF.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

By default, C6 DRC analysis treats clocks in the same synchronous clock group as if they are equivalent. You can use the "set_drc_handling C6 -ignore_synchronous_clock_groups on" to disable this behavior.

For more information on synchronous clock groups, see add_synchronous_clock_group.

- -SKip_procedure {TEst_setup| OFf}

  An optional switch and literal pair that affects the tool's handling of the E4 rule. The default option specifies to skip these checks when simulating the test_setup procedure. The tool continues to check for contention during other test procedures. The OFf option specifies for the tool to perform E4 rule checks for the test_setup procedure, as well as other procedures, during DRC.

  It is very common during test_setup to have valid but momentary bus contention before the design reaches its initialized state. The E4 rule often detects these momentary occurrences of bus contention, and the tool may issue many E4 violation messages. To be notified about this type of bus contention during test_setup, use the OFf option to include test_setup for E4 rule checks. This switch has no effect if you have previously set E4 handling to Ignore.

- -TIMing_exception_check {ON | OFf | Auto}

  An optional switch and literal that enables the timing exception check for the E5 DRC. The default is Auto, in which case the tool considers timing exception paths during E5 DRC (checking for X sources) when the tool is in the patterns -scan context and makes the handling of E5 DRC an error when the tool is in "Logic_bist" context. When the setting is Auto and the tool is not in "Logic_bist" context, the tool ignores timing exception paths for E5 DRC.

- -CONSIder_primary_outputs_observable {ON | OFf}

  An optional switch and literal that enables checking the propagation of X sources to a primary output for the E5 DRC. The default is off, in which case the tool does not check whether X sources reach a primary output during E5 DRC.

- -MASk_interactions {ON | OFf}

  An optional switch and literal pair that controls the addition of false paths to avoid simulation mismatches due to C22 DRC violations. The default (on) enables the tool to add the cross domain false paths. The tool generates these messages as part of DRC to summarize the C22 false path addition:

  ```
  //  2 cross domain false paths are added to prevent simulation
  //  mismatches due to C22 DRC violations.
  //  Use "set_drc_handling C22 -mask_interactions off" in SETUP mode
  //  to disable automatic addition of false paths.
  ```

- -INTeractions {edge | any}

  An optional switch and literal pair that specifies the interaction type for C22 cross clock path checks. The default (edge) only checks same-edge interactions for C22. Use "-interactions any" to find cross clock path checks with any interactions.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -PIS_AS_SEquential_elements {ON | OFf}

  An optional switch and literal pair that changes how the tool considers PIs to determine C27 violations. This option is helpful at the block level when chip-level sequential elements drive the PIs.

  The default is OFF. The tool assumes the PIs are on a different transition edge from the sequential elements by default. When the value is ON, the tool considers the PIs for both LE and TE sequential elements during the C27 check.

  The tool ignores this switch when the design level setting is chip-level (using the "set_design_level chip" command). In this case, the tool assumes that the PIs are on different transition edges from the sequential elements.

- -Mode {Combinational | Sequential}

  An optional switch and literal for the E10 rule.

  Combinational — Default. This option checks for bus contention caused by mutual exclusivity. This checking differs from rule E4 in that it does not check for this condition during test procedures.

  Sequential — Considers the inputs to a single level of sequential cells behaving as "staging" latches in the enable lines of tri-state drivers. All of the latches found in a back trace must share the same clock. There must also be only a single clocked data port on each cell, and both set and reset inputs must be tied (not pin constrained) to the inactive state. This check ensures that there is no connectivity from the cells in the input cone of the sequential cells and enables of the tri-state devices except through the sequential cells.

- -SIMULATE_ACROSS_Internal_primary_inputs {ON | OFf}

  An optional switch and literal pair that changes how the tool simulates internal PI's during DRC. Internal PIs are primary inputs you previously defined using the add_primary_inputs or add_clocks commands.

  After you set this switch to ON in setup mode, during test procedure simulation the tool starts off simulating an internal PI as an internal signal by carrying over its driver's simulation value until the point when you first supply a simulation value by explicitly forcing this PI in any of the following test procedures: test_setup, load_unload, or shift.

  This switch is needed for cases where you may have a cut point (such as for capture modeling), yet you need test_setup to be simulated normally as if the cut point were not present. This is commonly needed when using IJTAG within test_setup, where you want the IJTAG network simulated normally even if the network includes a cut point. For more information about using IJTAG and test procedures, refer to "Test Setup and Test End Procedures" in the *Tessent IJTAG User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -auto_fix {ON | OFf}

  An optional switch used to control automatic fixing for specific DRC rules on internal nodes. It is on by default. Currently only DFT_C9 supports this feature. Use this switch to disable automatic corrections for DFT_C9 as follows:

  ```
  set_drc_handling dft_c9 -auto_fix off
  ```

  Note

  Automatic fixing only affects DRC failures on internal nodes. The tool fixes any primary input that fails DFT_C9, regardless of whether the -auto_fix switch is ON or OFf.

  If you are registering your own DRC rules using the register_drc command and you have used the -allow_auto_fix switch, it is possible to disable the automatic fixing using the set_drc_handling command.

- -ALLOW_COMmon_scan_cells_in_multiple_chains {ON | OFf}

  An optional switch and literal pair that specify whether or not scan cells can be shared among scan chains. Note that there are some restrictions on scan cell sharing, which result in a T2 violation when not observed:

  - Scan cells cannot be shared for compressed chains driven by a decompressor.
  - The shared scan cells must be at the logical scan cell boundary. A library model with multiple scan cells cannot be partially shared. In other words, you cannot have the merged point within the state elements of a logical scan cell.

- -SCan_chain_tracing{Aggressive | Conservative}

  An optional switch and literal pair that specify whether or not scan chain tracing passes with a T25 violation. The choices are as follows:

  Aggressive — Scan chain tracing passes with a T25 violation. This is the default. For more information on T25 violations, see the "T25" section.

  Conservative — Scan chain tracing does not pass with a T25 violation. In this case, scan chain tracing fails, design rule checking stops, and the tool issues a T3 violation. For information on correcting T3 violations, see the "T3" section.

Examples

Example 1

The following example changes the E4 rule check to be an error:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 outdata4
add_clocks 1 clock1
add_clocks 0 clock2
set_drc_handling e4 error
set_system_mode analysis
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following command enables the race checks during setup mode and activates the D5 DRC analysis to consider both the C6 DRC clock race check results, and any asynchronous clock domain paths to avoid simulation mismatches.

```
SETUP> set_drc_handling D5 -conservative on
SETUP> set_system_mode analysis
…
// 2 D5 cells have been changed to Init-X to prevent mismatches due to cross clock domain
// or C6 race.
…
```

Related Topics

get_drc_handling

report_drc_rules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_driver_restriction

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether multiple drivers on buses and multiple active ports on gates can be used.

Usage

set_driver_restriction **OFf** | **ON**

Description

The set_driver_restriction command lets you specify for the tool to report a contention problem whenever there are multiple nets driving values onto a bus, or when multiple ports are active on an individual gate. The default upon tool invocation is that multiple driving nets on a bus or multiple active ports on a gate can be used if the signals are driving the same value. If multiple signals are on and are not driving the same values, then the tool flags it as contention.

However, in some design processes a single driver can be on at a time regardless of whether the signals are driving the same values. The set_driver_restriction command lets you place this same restriction on the tool.

Arguments

- **OFf**

  A literal that specifies that multiple drivers can be on for a bus and multiple ports can be active for a gate if the driving signals are of the same value, and therefore there is no contention. This is the default behavior when you invoke the tool.

- **ON**

  A literal that restricts buses to only have one driver on at a time and gates (dff and latches with multiple clocks) to only have one active port; the tool flags multiple active drivers or ports as contention problems.

Examples

The following example creates a strict contention checking environment. The first command specifies for the tool to check for contention on both multiple port gates and buses. If there is a contention problem where signals are driving different values, the tool reports an error and stops the simulation. The second command further restricts the contention checking environment by specifying that multiple drivers cannot drive even the same values.

```
set_contention_check on -error -all
set_driver_restriction on
```

Related Topics

set_contention_check

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_edt_abort_analysis_options

Context: dft -edt, patterns -scan (EDT On)

Mode: all modes

Specifies the number of EDT Aborted (EAB) test cubes on which to perform EAB fault analysis.

Usage

set_edt_abort_analysis_options **-max_cubes_to_analyze** {unlimited | *maxCubes*}

Description

Use the set_edt_abort_analysis_options command to set the maximum number of EAB test cubes to store during the EDT session. By default, up to 1000 test cubes are stored. If you specify the unlimited keyword, the tool stores all EAB test cubes generated from the EDT session.

You must run this command before using the create_patterns command.

See "EDT Aborted Fault Analysis" in the *Tessent TestKompress User's Manual* for a complete discussion.

Arguments

- **-max_cubes_to_analyze**{unlimited | *maxCubes*}

  A required switch and value pair that specifies the number of EAB test cubes. Choose one from the following:

  unlimited — A literal that specifies all EAB test cubes for analysis.

  maxCubes — An integer that specifies the number of EAB test cubes for analysis. The default is 1000.

Related Topics

report_edt_abort_analysis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_edt_finder

Context: patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Controls whether the tool automatically identifies EDT logic and updates scan chain pins during test pattern generation.

Usage

set_edt_finder [OFf | ON] [-Verbose {OFf | ON}]

Description

This capability is referred to as EDT Finder. By default, EDT Finder is "on".

When EDT Finder is on, the F DRCs are enabled, in addition to the normal K DRCs. The F DRCs provide more precise identification of EDT logic errors. Additionally, when EDT Finder is on, EDT DRC run time is greatly reduced.

This command can be used in conjunction with the TCD flow for ATPG. For more information, see "Core Mapping for ATPG Overview" in the *Tessent Scan and ATPG User's Manual*.

You can use this command to get updated scan pin information when your design hierarchy changes after the EDT logic is generated. For more information, see "Updating Scan Pins for Test Pattern Generation" in the *Tessent TestKompress User's Manual*.

EDT Finder uses the defined scan chain pins if they match what EDT Finder itself found in the logic. That is, if in shift mode EDT Finder determines that there is a controlling path between the defined scan chain pin and the pin that EDT Finder found, EDT Finder uses the defined scan chain pin.

If no arguments are specified, the current EDT Finder settings display.

Note

EDT Finder is automatically disabled in cases that are not supported—for example, in BIST mode. Because EDT Finder is enabled by default, DI faults are automatically identified in EDT logic. You can disable this behavior by executing the following command:
set_fault_subclass_analysis -disable DI.EDT

For more information, see "Fault Sub-Classes" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- OFf

  An optional literal that disables EDT Finder.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ON

  An optional literal that enables EDT Finder. This is the default.

- -Verbose {OFf | ON}

  An optional switch and literal pair that enables the transcription of informational messages about the EDT logic found. The transcription of informational messages is disabled upon invocation of the tool.

Examples

The following example shows the output of EDT Finder for a design with two channels and eight internal scan chains. EDT Finder output is highlighted in bold.

```
dofile results/generated_edt.dofile
add_scan_groups grp1 results/generated_edt.testproc
add_scan_chains -internal chain1 grp1 /top_i/si1 /top_i/so1

add_scan_chains -internal chain2 grp1 /top_i/si2 /top_i/so2
set_edt_pins update EdtUpdate
set_edt_pins input_channel  1 ChannelInput_1
set_edt_pins output_channel 1 ChannelOutput_1
set_edt_pins input_channel  2 ChannelInput_2
set_edt_pins output_channel 2 ChannelOutput_2
set_mask_decoder_connection -mode_bit 1 5
...
set_system_mode analysis

// Flattening process completed, cell instances=221, gates=471, PIs=16,
// POs=11, CPU time=0.00 sec.
//  ----------------------------------------------------------------------
//  Begin circuit learning analyses.
//  ----------------------------------------------------------------------
...
//  ----------------------------------------------------------------------
//  Begin EDT finder analyses.
//  ----------------------------------------------------------------------
//  EDT Finder completed, EDT blocks=1, scan chains=8, CPU time=0.02 sec.
//  ----------------------------------------------------------------------
//  Begin scan chain identification process, memory elements = 54.
//  ----------------------------------------------------------------------
...
```

Related Topics

report_edt_finder

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_edt_instances

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Specifies the location in which the tool places the EDT logic.

Usage

set_edt_instances [-Edt_logic_top *edt_instance_name_or_pathname*]
[-Compactor *compactor_instance_name_or_pathname*]
[-Decompressor *decompressor_instance_name_or_pathname*]
[-Block_location *path_to_top_of_block*]

Description

IP Creation phase:

In the IP Creation phase, you use the set_edt_instances command with the -Block_location switch to specify the parent instance in which the EDT logic is to be inserted. See Example 1. Note, if you use any switch except -Block_location in this phase, the tool ignores the switch and reports a warning.

By default, the tool instantiates EDT logic at the top level of the design.

Pattern Generation phase:

Specifies an instance name/location for existing EDT logic or for the design block that contains the EDT logic (EDT block).

You can use this command in the Pattern Generation phase to:

- Tell the tool the new name of an already renamed EDT logic block, decompressor, or compressor. See Example 3.
- Name each EDT block uniquely, using the -Block_location switch, when an EDT block is used multiple times. See Example 4.

  The set_edt_instances command operates only on the current EDT block which is the last block specified by either the add_edt_blocks or set_current_edt_block commands. For more information, see "Modular Compressed ATPG" in the *Tessent TestKompress User's Manual*.

Arguments

- -Edt_logic_top *edt_instance_name_or_pathname*

  A required switch and string pair that specifies an instance name or pathname for the design block containing the EDT logic (EDT block). Names specified with this switch must be unique. If the specified edt_instance_name_or_pathname is not unique in the design, the tool issues a message; in this case, you can specify the full pathname.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Compactor *compactor_instance_name_or_pathname*

  A required switch and string pair that specifies an instance name or pathname for the compactor sub-block within the current EDT block. Names specified with this switch must be unique. If the specified compactor_instance_name_or_pathname is not unique in the design, the tool issues a message; in this case, you can specify the full pathname.

- -Decompressor *decompressor_instance_name_or_pathname*

  A required switch and string pair that specifies an instance name or pathname for the decompressor sub-block within the current EDT block. Names specified with this switch must be unique. If the specified decompressor_instance_name_or_pathname is not unique in the design, the tool issues a message; in this case, you can specify the full pathname.

- -Block_location *path_to_top_of_block*

  A required switch and string pair that specifies the location of the current EDT block within the top-level netlist.

  During EDT IP Creation, use this switch to specify the location in which the tool places the EDT logic.

  During Pattern Generation, use this switch to identify each EDT block uniquely when an EDT block is used multiple times, by specifying its parent instance pathname.

Examples

Example 1

The following command is issued during EDT logic creation to insert the EDT logic in the instance, pad_east/test_blk, instead of at the top level of the design:

set_edt_instances -block_location pad_east/test_blk

Example 2

The following example specifies the name of the instance that contains the EDT logic:

set_edt_instances -edt_logic_top my_core_edt_i

```
// Found edt_logic_top at path: "/my_core_edt_top/my_core_edt_i".
```

Example 3

The following example specifies the instance names of the decompressor and compactor for the EDT logic and reports the current instance locations:

**set_edt_instances -decompressor my_core_edt_decompressor_i \
-compactor my_core_edt_compactor_i**

```
// Found compactor at path:
//    "/my_core_edt_top/my_core_edt_i/my_core_edt_compactor_i".
// Found decompressor at path:
//    "/my_core_edt_top/my_core_edt_i/my_core_edt_decompressor_i".
```

report_edt_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// edt_logic_top
//    Setting: my_core_edt_i
//    Path   : /my_core_edt_top/my_core_edt_i
//
// decompressor
//    Setting: my_core_edt_decompressor_i
//    Path   : /my_core_edt_top/my_core_edt_i/my_core_edt_decompressor_i
//
// compactor
//    Setting: my_core_edt_compactor_i
//    Path   : /my_core_edt_top/my_core_edt_i/my_core_edt_compactor_i
```

Example 4

The following example specifies unique names for instances of the same EDT block in a top-level modular netlist:

set_edt_mapping on
add_edt_blocks my_core1_a

```
// my_core1_a is the current EDT block.
```

set_edt_instances -block_location /my_core1_a_edt_i
dofile my_core1_edt.dofile
...
report_edt_instances

```
// edt_logic_top
//    Setting: my_core1_edt_i
//    Path   : /my_core1_a_edt_i/my_core1_edt_i
...
```

add_edt_blocks my_core1_b

```
// my_core1_b is the current EDT block.
```

set_edt_instances -block_location /my_core1_b_edt_i
dofile my_core1_edt.dofile
...
report_edt_instances

```
// edt_logic_top
//    Setting: my_core1_edt_i
//    Path   : /my_core1_b_edt_i/my_core1_edt_i
...
```

The set_edt_mapping command enables the requisite mapping of block-level dofile commands to the top-level.

Related Topics

add_edt_blocks

report_edt_instances

set_current_edt_block

set_edt_mapping

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_edt_mapping

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Enables or disables automatic mapping of block-level dofile commands so block-level dofiles can be reused without modification for top-level pattern creation.

Usage

set_edt_mapping {**OFf** | **ON**} [-Verbose {**OFf** | **ON**}]

Description

Note

This command and the EDT Mapping functionality have been superseded by the newer TCD flow for ATPG. It is recommended to use the newer capability described in "Core Mapping for ATPG Overview" in the *Tessent Scan and ATPG User's Manual*.

The set_edt_mapping command determines whether or not the tool performs the automatic mapping necessary to reuse, for top-level pattern generation, the dofiles the tool generated during block-level IP creation. You must set this command to "on" prior to running block-level dofiles to set up for top-level pattern creation.

For more information about the modular Tessent TestKompress flow or the reuse of block-level dofiles, refer to the "Modular Compressed ATPG" chapter of the *Tessent TestKompress User's Manual*.

Note

Switch settings for this command are cumulative; that is, switches retain their settings over multiple set_edt_mapping commands until you specify a different setting

Arguments

- **OFf**

  A literal that disables automatic mapping necessary to reuse the block level dofiles generated during the IP Creation Phase. This is the default upon invocation of the tool.

- **ON**

  A literal that enables the automatic mapping necessary to reuse the block level dofiles generated during the IP Creation Phase.

- -Verbose {**OFf**| **ON**}

  An optional switch and literal pair that enables or disables the tool's transcription of informational messages whenever it changes any of the following block-level dofile objects as a result of automatic mapping:

  - scan group name

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- scan chain name
- pin pathname
- pin name

The transcription of informational messages is disabled upon invocation of the tool.

Examples

Example 1

The following example enables the automatic mapping of commands from the block-level dofiles to the top-level design hierarchy, then executes the commands in one of the block-level dofiles:

set_edt_mapping on

delete_edt_blocks -all

```
// All blocks deleted.  No EDT block now selected as current block.
```

add_edt_blocks created_core1

```
// created_core1 is the current EDT block.
```

set_edt_instances -block_location /x1
dofile created_core1_edt.dofile

```
// command: add_scan_groups grp1 ./edt_ip/created_core1_edt.testproc
// Note: Ignoring "add_scan_groups" command for grp1. Using top-level
// definition since scan group mapping is active.
// command: add_scan_chains -internal chain1 grp1 /m8_i/scan_in1
// /m8_i/scan_out1
...
// command: add_scan_chains -internal chain4 grp1 /m8_i/scan_in4
// /m8_i/scan_out4
// command: add_clocks 0 SCLK
// command: add_clocks 0 CCLK
...
```

Example 2

The following example shows the transcription that would have occurred had the -Verbose switch been in effect for the commands of the preceding example:

set_edt_mapping on -verbose on

delete_edt_blocks -all

```
// All blocks deleted.  No EDT block now selected as current block.
```

add_edt_block created_core1

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// created_core1 is the current EDT block.
```

set_edt_instances -block_location /x1
dofile created_core1_edt.dofile

```
// command: add_scan_groups grp1 ./edt_ip/created_core1_edt.testproc
// Note: Ignoring "add_scan_groups" command for grp1. Using top-level
// definition since scan group mapping is active.
// command: add_scan_chains -internal chain1 grp1 /m8_i/scan_in1
// /m8_i/scan_out1
// Note: Block level chain name (chain1) automatically translated to
// created_core1_chain1.
// Note: Block level path (/m8_i/scan_in1) automatically translated to
// /x1/m8_i/scan_in1.
// Note: Block level path (/m8_i/scan_out1) automatically translated to
// /x1/m8_i/scan_out1.
...
// command: add_scan_chains -internal chain4 grp1 /m8_i/scan_in4
// /m8_i/scan_out4
// Note: Block level chain name (chain4) automatically translated to
// created_core1_chain4.
// Note: Block level path (/m8_i/scan_in4) automatically translated to
// /x1/m8_i/scan_in4.
// Note: Block level path (/m8_i/scan_out4) automatically translated to
// /x1/m8_i/scan_out4.
// command: add_clocks 0 SCLK
// Note: Block level "add_clocks" command for SCLK is to be verified
// during DRC.
// command: add_clocks 0 CCLK
// Note: Block level "add_clocks" command for CCLK is to be verified
// during DRC.
```

Related Topics

add_edt_blocks

delete_edt_blocks

set_edt_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_edt_options

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis (dft -edt, dft -scan, and patterns -scan contexts only)

Determines whether compression is on and specifies parameters for the EDT logic, including scan channels, bypass circuitry, lockup cells, pipeline stages, and compactor type.

Usage

set_edt_options [OFf | ON | Auto]
[-CHannels *integer*]
[-INPut_CHannels *integer*]
[-Used_input_channels *integer*]
[-OUtput_CHannels *integer*]
[-LOCAtion {External | Internal}]
[-BYPASS_Chains *integer*]
[-OVERShift_cycles *integer*]
[-SIngle_bypass_chain {ON | OFf}]
[-BYpass_logic {ON | OFf | Use_existing_bypass_chains}]
[-CLocking {Edge | Level}]
[-PIpeline_logic_levels_in_compactor {OFf | *integer*}]
[-LOCKup_cells {ON | OFf}]
[-LONgest_chain_range {*min_number_cells max_number_cells*}]
[-RESet_signal {OFf | Asynchronous | ijtag_reset}]
[-COMpactor_type {XPress | BAsic}]
[-PULse_EDt_Before_shift_clocks {ON | OFf}]
[-FOrce_1hot_masking {OFf | ON}]
[-SEParate_control_data_channels (ON | OFf}]
[-SEGmented_decompressor {*integer* | Auto}]
[-MAX_Chains_per_decompressor_segment *integer*]
[-RETIme_chain_boundaries {OFf | ON}]
[-BYPASS_CHAIN_Change_edge {OFf | ON}]
[-BYPASS_CHANGE_EDGE_Clock {Scan_clock | Prefer_edt_clock}]

For dft -scan

set_edt_options [-Location {External | Internal}]
[-Channels *integer*]

For Hybrid TK/LBIST Flow IP Creation

set_edt_options **-lbist_misr_input_ratio** { *integer* | auto }
[-reset_signal asynchronous]
[-SIngle_bypass_chain {ON | OFf}]
[-BYpass_logic {ON | OFf}]
[-chain_mask_register_ratio *ratio*]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For Hybrid TK/LBIST Flow Fault Simulation

set_edt_options [-decompressor_seed ***hex_string***]

Description

To create EDT logic for a basic application, you only need to specify the number of channels.

When you set this command to Off, you can use the tool for ATPG and it behaves the same as Tessent FastScan in command-line mode.

Use the report_environment command to display the current settings.

The tool retains the settings specified by the set_edt_options command over multiple uses of the command, until a different setting is specified.

Hybrid TK/LBIST Flow Specifics

Sets options for EDT IP creation and LogicBIST fault simulation.

IP Creation: This command specifies options specific to the EDT/LogicBist blocks.

Fault Simulation: This command describes the initial seed of the PRPG.

Refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- OFf

  An optional literal that turns off compression (EDT) mode and enables analysis mode.

- ON

  An optional literal that enables compression (EDT) mode.

- Auto

  An optional literal that enables automatic detection of EDT mode (compression).

  Note

  This option is only available in the patterns -scan context.

- -CHannels *integer*

  An optional switch and integer pair that specifies the number of external scan channels. Use this switch when the number of input channels is the same as the number of output channels.

  Use this switch to set up the number of scan channels the tool configures the EDT logic with. During pattern generation, this switch specifies the number of scan channels the EDT logic contains. The dofile created when the tool generates EDT logic contains a set_edt_options command that specifies the number of channels for test pattern generation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -INPut_CHannels *integer*

  An optional switch and integer pair that specifies the number of scan input channels. Use this switch only if the number of input channels differs from the number of output channels.

- -Used_input_channels *integer*

  An optional switch and integer pair that specifies the number of EDT IP input channels used for pattern generation. This option is not valid during IP creation. This switch can only be used in setup mode.

  You can configure the EDT IP to use a subset of the input channels. For example, you might generate the EDT IP to support some number of input channels, but then when the IP or core is embedded into the design, only a subset of the channels can be driven due to scan port limitations.

  Note

  During EDT IP creation, the tool considers all channels as possible inputs/outputs for the bypass chains. If you plan to tie any of the input channels at some point, you must consider constraining the used input channels prior to EDT IP creation.

  Using this option can provide a flexible test compression planning flow. Because a core could be designed independent of the rest of the SoC, it could be difficult to determine the optimal number of scan input channels during the core design phase. You can pick a conservative compression ratio by using enough scan input channels at the IP creation phase. When integrating all cores into the SoC, if using only a subset of the input channels is sufficient for a core, you can use this feature to specify the used input channels and apply a specific constant value on the unused input channels on the boundary of the EDT IP during shift.

  The following usage conditions apply:

  - To use this switch, you must set the following in the IP Creation Phase:

    ```
    set_edt_options -separate_control_data_channels ON
    ```

    The only exception is if you are using the basic compactor and no low-power controller.

  - Only data channels with highest channel indexes can become unused. For example, consider an EDT block with one control channel (channel1) and three data channels (channel2, channel3, channel4). If you want to only use 3 input channels, you can only tie channel4 to “0”. If you want to only use 2 input channels, you can only tie both channel 3 and channel4 to “0”. You must have at least one data channel.

  - If the unused input channels are connected to top-level primary inputs, patterns at these pins must hold at “0”s. Do this by constraining the unused input channels using the add_input_constraints command:

    ```
    add_input_constraints XXX -C0
    ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

or

```
add_input_constraints XXX -C1 (if there is an inversion between
the tied pin and the EDT channel input)
```

Otherwise, the tool issues DRC violations during system mode transition.

- If internal signals drive the unused input channels, these channels must be controlled in the design to ensure that "0"s are injected into the decompressor from these unused channels.
- If logic that is set up to provide constant values during shift (for example, a TDR that is loaded with "0" in test_setup) drives the unused input channels, then you must apply the appropriate setup patterns.

Refer to "Core Instance Parameters" in the *Tessent TestKompress User's Manual*.

- -OUtput_CHannels *integer*

  An optional switch and integer pair that specifies the number of scan output channels. Use this switch only if the number of output channels differs from the number of input channels.

- -BYPASS_Chains *integer*

  An optional switch and integer pair that specifies how many scan chains are created for bypass mode. The specified value must be equal to or less than the number of bypass chains created by default. By default, the number of bypass chains created equals the number of input/output channels. If the number of input and output channels differ, the smaller number is used.

  For dual compression configuration applications, you must use this switch after both configurations are defined.

- -OVERShift_cycles *integer*

  An optional switch and integer pair that adds a specified number of shift cycles to a test pattern set.

  Use this switch before saving out a particular pattern set to adjust the number of shift cycles. During core level pattern generation, the patterns can be saved as is. While integrating core level patterns at the chip level, you know the differences in the number of shift cycles for the different pattern sets corresponding to the different cores. You can then use the switch to add the additional cycles to the pattern set with lesser number of cycles to balance out all the pattern sets. This facilitates pattern merging at the chip level.

- -SIngle_bypass_chain {ON | OFf}

  An optional switch and literal pair that creates compression logic with a dual bypass chain configuration. Use this option to create a bypass configuration that concatenates all scan chains together into one bypass chain in addition to the default bypass configuration. For more information, see "Dual Bypass Configurations" in the *Tessent TestKompress User's Manual*.

  Hybrid TK/LBIST Flow Only: -SIngle_bypass_chain {ON | OFf}

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Turn this switch off to disable generating single bypass chain logic on a per-block basis.

Caution

Turning this switch off is not recommended, because doing so also disables LBIST pattern diagnosis.

- -LOCAtion {External | Internal}

  An optional switch and literal pair that specifies a location for the EDT logic relative to the netlist read in. Options include the following:

  External — Creates a new top level "wrapper" module and instantiates both the design netlist read in and the EDT logic within it. The logic is thus kept external to the design netlist. This is the default.

  Internal — Instantiates the EDT logic in the existing top level of the design netlist.

- -BYpass_logic{ON | OFf | Use_existing_bypass_chains}

  An optional switch and literal pair that specifies a location for the bypass logic. Bypass logic enables you to bypass the compression circuitry and run traditional ATPG with a tool such as Tessent FastScan. Bypass logic is transparent during normal operation. Options include the following:

  ON — Includes bypass logic in the EDT logic. This is the default.

  OFf — The tool does not add any logic; it assumes that a bypass does not exist and inserts no logic.

  Use_existing_bypass_chains — Uses bypass logic already in the core logic. In this case, the tool inserts two muxes for each EDT channel (not chain) in order to select between the compressed and uncompressed path. Example 3 demonstrates how to use this option.

  Hybrid TK/LBIST Flow Only: Turn this switch off to disable generating bypass logic. When you do so, the tool also automatically disables generating single bypass chain logic.

  Caution

  Turning this switch off or specifying Use_existing_bypass_chains is not recommended, because doing so disables LBIST pattern diagnosis.

- -CLocking {Edge | Level}

  An optional switch and literal pair that determines what type of architecture logic is used for the EDT logic. Options include the following:

  Edge — Specifies DFF-based logic. You can use DFF-based logic for designs that are LSSD-based, mux-DFF-based, or that incorporate a mixture of these two architectures. This is the default.

  Level — Specifies latch-based logic. This option is for pure LSSD designs only.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -PIpeline_logic_levels_in_compactor {OFf | *integer*}

  An optional switch and literal pair that enables pipeline stages in the compactor of the EDT logic. Options include the following:

  OFf — Disables pipeline stages in the EDT logic. This is the default.

  <*integer*> — Includes pipeline stages in the EDT logic compactor or compactors and specifies the maximum number of logic levels (2-input XOR gates) that can be used in the compactors before inserting a pipeline stage.

- -LOCKup_cells {ON | OFf}

  An optional switch and literal pair that determines whether lockup cells are inserted in the EDT logic.

  Caution

  It is recommended to leave this switch set to on. Turning off lockup cell synthesis can lead to DRC violations in the pattern generation phase and inability to generate valid patterns. For more information, see "Understanding Lockup Cells" in the *Tessent TestKompress User's Manual*.

  Options include the following:

  ON — Includes lockup cells in the EDT logic as needed between the decompressor and the core, between the core and the compactors (if pipelining is used), and between scan chains when they are concatenated for bypass mode. This is the default setting.

  OFf — Disables lockup cells in the EDT logic.

- -LONgest_chain_range *min_number_cells max_number_cells*

  An optional switch and integer pair that defines a range for the length of the longest scan chain. Use this switch to avoid having to regenerate the logic for minor changes in the length of the chains after the logic is created.

  Note

  This switch defines a range for the length of the longest scan chain in your design. This does *not* mean the range of lengths of all the scan chains in your design. Setting the min_number_cells option based on these considerations enables the tool to configure the EDT logic to ensure robust pattern compression.

  Note

  We recommend you use this switch when creating EDT logic with the skeleton flow. For more information, see "Longest Scan Chain Range Estimate" in the *Tessent TestKompress User's Manual*.

  Options include the following:

  <*min_number_cells*> — Specifies the minimum number of cells in the longest scan chain. When you use the -longest_chain_range switch with separate control and data

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

channels, the <*min_number_cells*> impacts the number of control channels that are created, according to this formula:

```
Number of EDT control channels = Ceil(total number of control bits /
min_number_cells)
```

This can result in higher pattern count if the final design has more control channels than necessary.

For example, if you specify the minimum longest chain length as 50 and the design has 100 control bits, the generated hardware has two control channels. However, if the actual longest chain length in the final design is 100 or more, one control channel is sufficient, but the EDT IP wastes one control channel and has one fewer data channel, which increases the pattern count unnecessarily. You should estimate the <*min_number_cells*> as close as possible to its final design.

<*max_number_cells*> — Specifies the maximum number of cells in the longest scan chain. Specify this value as accurately as possible. An overly large <*max_number_cells*> value unnecessarily increases the decompressor size. If the <*max_number_cells*> value is smaller than the longest scan chain, the EDT may be underspecified, the tool may produce a warning message during pattern generation, and some testable faults may remain untestable.

- -RESet_signal {OFf | Asynchronous | ijtag_reset}

An optional switch and literal pair that specifies whether an asynchronous reset signal, edt_reset, is included in the EDT logic. In most cases such a signal is not needed, but you can use one if design requirements dictate that all the sequential elements in the design are resettable. Options include the following:

OFf — Disables the asynchronous reset. This is the default.

Asynchronous— Includes an asynchronous reset signal in the EDT logic and a new top-level pin named “edt_reset” to control it. You can rename this pin or share it with a functional pin in the design core using the set_edt_pins command.

When you specify reset as asynchronous, it is described as a DataInPort in ICL, and you can use the edt_reset core instance parameter to assert the reset signal.

ijtag_reset — For designs with IJTAG, use this setting to connect the edt_reset signal to ijtag_reset. This avoids using a separate top-level reset port. This setting applies to both EDT and hybrid EDT/LBIST instruments. This option is not supported for type-3 LPCTs, because type-3 LPCT controllers are not IJTAG compatible. For type-2 LPCTs, the connection and off state come from “set_lpct_pins reset” command.

When you connect edt_reset to ijtag_reset, the behavior is as follows:

- EDT IP — The signal uses the pin specified with “set_edt_pins reset” command and uses the signal polarity as specified with the -active option for the set_edt_pins command. The default is active high.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Hybrid EDT/LBIST IP — The top-level EDT and LBIST modules do not have edt_reset pins; instead, they connect internally to the ijtag_reset pin already present. The signal follows the ijtag_reset active state.

When you specify reset as ijtag_reset, it is connected to ijtag_reset, so the reset is automatic, and you cannot influence it outside of calls to iReset.

Note

It is recommended to define the reset as a clock primary input using the add_clocks command during logic creation. This accounts for the signal off state (high or low) when writing out the EDT logic. By default, an active high is assumed at the boundary of the EDT logic.

- -COMpactor_type {XPress| BAsic}

  An optional switch and literal pair that determines which compactor is used in the EDT logic. Options include the following:

  XPress — Xpress compactor. The Xpress compactor increases compression, especially for designs that generate unknown (X) values. Default setting.

  BAsic — Basic compactor. The basic compactor is the first generation compactor.

  For more information, see "Understanding Compactor Options" in the *Tessent TestKompress User's Manual*.

- -PULse_EDt_Before_shift_clocks {OFf | ON}

  An optional switch and literal pair that determines whether the EDT logic clock is pulsed before the scan chain clock. By default, the EDT logic and scan chain clock are pulsed simultaneously. For more information, see the "Pulse EDT Clock Before Scan Shift Clocks" section in the *Tessent TestKompress User's Manual*.

- -FOrce_1hot_masking {OFf| ON}

  An optional switch and literal pair that determines whether Tessent TestKompress generates test patterns that observe a single scan chain or multiple scan chains on each channel of the Xpress compactor. Options include the following:

  OFf — Depending on unknown states and targeted faults, Tessent TestKompress determines which scan chains to observe. Default behavior.

  ON — One scan chain is observed on each compactor channel. You should use this setting for debug purposes. Generating test patterns with this setting increases test pattern count.

  This switch is only valid when the Xpress compactor is used. Observing only one scan chain output per channel enables you to easily map a failure on a channel to a specific scan cell. For more information on the Xpress compactor, see "Understanding Compactor Options" in the *Tessent TestKompress User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -SEParate_control_data_channels {OFf |ON}

  An optional switch and literal pair that, when enabled (set to on), specifies that the tool automatically determines the number of control channels and inserts control bits behind the control channels only. This is needed for channel sharing between non-identical blocks. For more information, see "EDT IP Creation With Separate Control and Data Input Channels" *Tessent TestKompress User's Manual*.

  Apply this setting per EDT block. For a design with multiple EDT blocks, you can enable some of the blocks using "-separate_control_data_channels on" and disable the remaining blocks using "-separate_control_data_channels off".

  Note

  You can only use this option if the user-specified number of input channels is greater than or equal to one plus the number of control channels; otherwise, the tool issues an error and aborts.

- -SEGmented_decompressor {*integer* | Auto}

  An optional switch and integer or literal pair that specifies the number of segments to split the decompressor into. If you specify an integer value for this argument, the tool determines the number of chains per segment based on the total number of chains and the specified number of segments. Lower numbers of chains per segment can help reduce routing congestion. The default value, auto, indicates that the tool uses the -MAX_Chains_per_decompressor_segment value to determine the number of segments. If you specify a value for both the -MAX_Chains_per_decompressor_segment argument and this argument, the tool issues a warning and uses this value.

- -MAX_Chains_per_decompressor_segment *integer*

  An optional switch and integer pair that specifies the maximum number of chains per segment when the tool segments the decompressor. By default, the tool segments the decompressor into equal parts with a maximum of 150 chains per segment. Lower numbers of chains per segment can help reduce routing congestion. If you specify a value for both the -SEGmented_decompressor argument and this argument, the tool issues a warning and ignores this value.

- -RETIme_chain_boundaries ON | OFf

  An optional switch and literal pair that specifies whether to add retiming cells (lockup cells) at scan chain inputs and outputs. These cells are added to ensure that each scan chain starts with an LE register and ends with a TE register. This switch replaces bypass lockup cells with retiming lockup cells that are part of the EDT scan chains; this eliminates the need for bypass-only lockup cells. For more information on bypass cells as part of the EDT logic, see "Differences Based on Inclusion/Exclusion of Bypass Lockup Cells in EDT Chains" in the *Tessent Shell Reference Manual*.

  The tool adds retiming cells only when it inserts lockup cells with the "-lockup_cells ON" option. It includes these cells as part of both EDT and EDT-bypass scan chains.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ON — Specifies to include retiming cells in EDT chains. This is the default for the Hybrid TK/LBIST flow (dft -edt -logic_bist context).

OFf — Specifies to not include retiming cells in EDT chains. This is the default for TestKompress EDT IP creation (dft -edt context).

For complete information on including retiming cells in EDT chains, see section "Lockup Cell Analysis For Bypass Lockup Cells Included as Part of the EDT Chain" in the *Tessent TestKompress User's Manual*.

- -BYPASS_CHAIN_Change_edge {OFf | ON}

  An optional switch and literal pair that specifies whether bypass chains always start with leading-edge (LE) cells.

  Options include the following:

  ON — Bypass chains always start with LE cells, and the tool changes the starting element if necessary.

  OFf — Bypass chains use the starting element of the scan chain.

- -BYPASS_CHANGE_EDGE_Clock {Scan_clock | Prefer_edt_clock}

  An optional switch and literal pair that specifies the clock used preferentially for the bypass chain change edge cell. By default, when a chain ends with a trailing edge (TE) and the bypass chain begins with a leading edge (LE), the cell is clocked by the scan clock, which is brought in from the top-level EDT using a dedicated input port. Use this switch to request that the tool use edt_clock preferentially when it is available.

  The edt_clock may not be available in the following situations:

  - When there are any bypass-mode input change edge cells. In this case, using the edt_clock may lead to clock skew problems and a potential T24 DRC violation.
  - When the output change edge cell becomes a copy cell.
  - When there are bypass lockup cells already driven by the scan clock.

  In these cases, the tool uses the scan clock even when edt_clock is requested, and it reports a warning to notify you that this has occurred.

- **-lbist_misr_input_ratio** { ***integer*** | **auto** } **(Hybrid TK/LBIST Flow IP Creation Only)**

  A required switch that specifies the ratio of the number of scan chains over number of MISR inputs. The specified value should be a power of 2.

  If the value is 1, all scan chain outputs are connected to the MISR. This is the default value. Setting this property to auto reverts the input ratio to the default 1 value.

  If the value is greater than or equal to *#scan_chains*/*#channels*, all EDT channel outputs are connected to the MISR.

  If any value in between is used, taps are taken from the middle of EDT compactor XOR tree corresponding to the value specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This argument only specifies the number of MISR inputs; the size of the MISR is implicitly determined based on this input. The minimum size of the MISR is at least 24 bits.

If you do not specify this switch, the tool uses a default MISR of 24 bits. When the design has more than 24 output channels, the default MISR size is increased to be greater than or equal to the number of output channels.

- -chain_mask_register_ratio *ratio*

  An optional switch and integer pair that specifies the number of scan chains controlled by a single EDT chain mask register bit. By default, the ratio is 1:1; there are as many bits in the chain mask register as there are number of scan chains.

  Use the -chain_mask_register_ratio switch to minimize the area impact of the chain mask register. For example, specifying 3 for *ratio* means a 3:1 ratio with each chain mask register bit controlling 3 scan chains. Specify the ratio on a per EDT block basis.

  The larger the ratio you specify, the more you trade off resolution for area impact. When any scan chain connected to a chain mask register bit is masked, all the other scan chains connected to the bit are also masked automatically.

  Refer to "Scan Chain Masking" in the *Hybrid TK/LBIST User's Manual* for more information.

- -decompressor_seed *hex_string* **(Hybrid TK/LBIST Flow Fault Simulation Only)**

  An optional switch and hexadecimal pair that specifies the initial seed of the PRPG. This seed applies to both the warm-up patterns and the actual patterns.

  The LogicBIST fault simulation dofile includes this switch with the hardware default value for arguments.

  If you intend to perform fault simulation with a different starting PRPG seed, modify the <*hex_string*> argument to specify the desired seed.

Examples

Example 1

The following example sets up a EDT logic for a design with four scan chains. The example sets up EDT logic for one scan channel and no bypass circuitry.

```
add_scan_groups grp1 atpg.proc
add_scan_chains chain1 grp1 edt_si1 edt_so1
add_scan_chains chain2 grp1 edt_si2 edt_so2
add_scan_chains chain3 grp1 edt_si3 edt_so3
add_scan_chains chain4 grp1 edt_si4 edt_so4
add_clocks 0 clk
set_edt_options -channels 1 -bypass_logic off
set_system_mode analysis
```

Example 2

The following example generates EDT logic with two input channels and one output channel. The number of bypass chains defaults to 1, corresponding to the smaller of the two specified channel values.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_edt_options -input_channels 2 -output_channels 1
```

Example 3

The following example demonstrates the use of the use_existing_bypass_chains option to the -bypass_logic switch. In this example, you have two bypass chains, because you plan to insert EDT with two input and two output channels. In this example, the my_bypass signal is active low.

1. Specify the bypass chains:

```
set_edt_options -bypass_logic use_existing_bypass_chains -bypass_chains 2
```

2. Specify the input and output of each chain:

```
set_bypass_chains -bypass_chain_number 1 -pins scan_in1 scan_out1 \
-edt_chains chain1 chain2 chain3 chain4

set_bypass_chains -bypass_chain_number 2 -pins scan_in2 scan_out2 \
-edt_chains chain5 chain6 chain7 chain8
```

The tool inserts a mux on scan_in1 and scan_in2 to select between the EDT channels and the bypass path. The tool inserts a mux on the output so that there are two muxes for each EDT channel.

The tool controls the newly inserted bypass muxes using the default edt_bypass pin, which is added unless you use the set_edt_pins command to specify your bypass control. If you do use the set_edt_pins command as shown here, the tool uses my_bypass instead of the default edt_bypass pin.

```
set_edt_pins bypass my_bypass
```

The tool assumes that the pin you specify here (my_bypass) is active high. If your bypass enable pin is active low, you must edit the generated RTL for EDT and add an inversion on the my_bypass signal coming into the EDT logic. With this change, when you set your bypass pin low, it configures all of your bypass muxes and the tool-inserted muxes to be in bypass mode.

Example 4 (Hybrid TK/LBIST Flow IP Creation)

This example connects the scan chains to MISR during IP creation by combining four scan chain outputs using XOR gates to each MISR input.

```
set_edt_options -lbist_misr_input_ratio 4
```

Example 5 (Hybrid TK/LBIST Flow IP Creation)

For a design with 20 scan chains, this example specifies the following command during IP creation to group 4 scan chains per chain mask register bit:

```
set_edt_options -chain_mask_register_ratio 4
```

The resulting chain mask register is a 5-bit register instead of a 20-bit register (the default).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 6 (Hybrid TK/LBIST Flow Fault Simulation)

The following example shows how to modify the PRPG seed in analysis mode during fault simulation:

```
set_edt_options -decompressor 00000001
set_system_mode analysis
simulate_patterns -source bist -store_patterns all
set_edt_options -decompressor 00110101
simulate_patterns -source bist -store_patterns all
```

Related Topics

report_edt_configurations

report_edt_pins

report_scan_volume

set_bypass_chains

set_compactor_connections

set_edt_pins

write_edt_files

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_edt_pins

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis (dft -edt and patterns -scan contexts only)

By default, all EDT pins are created as dedicated pins with default names and no inversions. The same set_edt_pins command must be used for both the pattern generation phase and the EDT logic creation phase to generate patterns correctly for modified pins.

Usage

Specifying a new or shared EDT pin

set_edt_pins {{Input_channel | Output_channel} {index [*pin_name*] [*internal_node_name*]} |
{*edt_pin* [*pin_name*] [*internal_node_name*]} |
{***pin_name*** -ON_edt_module}} [-active {low | high}]

Renaming and/or inverting EDT pins

set_edt_pins {{Input_channel | Output_channel} {{index [*pin_name*]} | {*edt_pin*}}
[-Noinv | -Inv]}

Inserting channel pipeline stages

set_edt_pins {{Input_channel | Output_channel}
{[*index*]{-PIpeline_stages *number_of_channel_pipeline_stages*}}}

Specifying the change edge for compactor outputs

set_edt_pins [Output_channel -CHange_edge_at_compactor_output
{Any | LEading_edge_of_edt_clock
| TRailing_edge_of_edt_clock}]
| [Output_channel *index* -CHange_edge_at_compactor_output
{Any | TRailing_edge_of_edt_clock}]

Specifying internally driven EDT control pins

set_edt_pins {{*edt_pin*} {-} [*internal_node_name*] [-Noinv | -Inv] [-ON_edt_module]}

Specifying internally driven EDT channel pins during IP creation

set_edt_pins {Input_channel | Output_channel} *index* {-} *internal_node_name*

Description

Except for added channel pipeline stages, a dofile generated during the EDT logic creation phase contains the correct set_edt_options commands for the pattern generation phase. If you add channel pipeline stages, you must specify those changes in the pattern generation phase with another set_edt_pins command.

Use the set_edt_pins command for the following operations:

- **Specify a new or shared EDT pin**

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When the design shares an EDT channel output pin with a functional output pin, the tool inserts a multiplexer in the top-level wrapper to share the signals between the output pins. The scan_enable signal controls the multiplexer. If you rename an EDT pin with a name already used by a functional pin in the core design, the EDT pin shares the functional pin.

- **Rename an EDT control pin**

  To rename an EDT pin, specify the *edt_pin* along with a *pin_name* argument that specifies the new name. To rename an EDT channel, specify an input/output channel and index number for the channel along with a *pin_name* argument that specifies the new name. To apply the new name to the pin at the EDT module level in the RTL, use the *pin_name* -On_edt_module switch.

- **Specify an inverted signal between the chip pin and the EDT logic**

  I/O pads added outside the EDT logic sometimes invert their signals. Use the -Inv switch to specify the inversion of the EDT control pins during EDT logic creation. This ensures that generated test procedures and dofiles operate correctly in the pattern generation phase after the I/O pads are synthesized. The *pin_name* argument is optional if you are only changing the inversion status of an EDT pin.

- **Specify that an EDT control or channel pin is driven internally**

  For more information, see "Internally Driven EDT Pins" in the *Tessent TestKompress User's Manual*.

  When you specify internally driven EDT pins during the IP creation phase, the *internal_node_name* value is required. During the pattern generation phase, the value is optional.

  When you specify an internally driven channel pin during the IP creation phase, the dofile and testproc file are not generated. Use the TCD mapping flow in this case for pattern generation.

- **Specify channel pipeline stages**

  Channel pipeline stages are pipeline stages added outside the EDT logic between a top level channel input pin/pad and the decompressor input or compactor output.

  Channel pipeline stages are typically added during top-level design integration to reduce signal propagation time between the top-level pads and the channel inputs and outputs. Channel pipeline stages are not created as part of the EDT logic, but you must specify them to generate the correct test patterns.

  To specify channel pipeline stages for a particular channel during the pattern generation phase, use the *index* number of the channel, the -Pipeline_stages switch, and the number of pipeline stages.

- **Specify the change edge for compactor outputs**

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Use this feature to specify whether compactor output data changes on the LE time or whether the tool aligns the change edge at compactor output with the TE of the EDT clock.

By default, the compactor output data changes on the TE time of the EDT clock. There are two usage scenarios as follows:

Without compactor pipelining (compactor pipeline register has not been inserted)

- If you specify LE for all channels, the tool adds a pipeline stage clocked by edt_clock (LE) at the compactor output; it adds the pipeline stage only if any of the last scan cells change at any time other than the LE of the EDT clock. It adds lockup cells if necessary either before the pipeline stage or at the output of the scan chains on a per-compactor basis.
- When you specify TE for all output channels, the tool adds a pipeline register driven by the LE of the EDT clock and a lockup cell driven by the TE of the EDT clock to all channels. When you specify TE for a subset of output channels, the tool adds only a lockup cell driven by the TE of the EDT clock for the channels specified as needed.

With compactor pipelining (compactor pipeline register has been inserted)

- The compactor pipeline guarantees the compactor output changes on the LE of EDT clock because edt_clock (LE) drives the pipeline stage inside the compactor. If the number of levels of logic is less than or equal to the number of pipeline levels specified with set_edt_options, the tool adds a pipeline stage only when necessary to ensure an LE change. In addition, it adds lockup cells when needed.
- If TE is specified for a channel, the tool adds an additional lockup cell on the output of every compactor to ensure the TE change edge.

The dofile written out during EDT logic creation reflects the pipeline stages added to meet the speed objectives or to ensure that the output of the compactor changes at the positive edge of the EDT clock.

For more information, see "EDT Control and Channel Pins" and "Using Pipeline Stages Between Pads and Channel Inputs or Outputs" in the *Tessent TestKompress User's Manual*.

Arguments

- Input_channel

  Optional literal that specifies an EDT input channel pin.
- Output_channel

  Optional literal that specifies an EDT output channel pin.
- *index*

  Optional integer that specifies the index number for a specified channel pin. The index must be positive and have a valid range from one up to the total number of scan channels.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- *pin_name*

  Optional replaceable string that specifies a pin name. Depending on usage, the *pin_name* may reflect the name of an existing functional pin in the core design or EDT logic, an existing pin on the EDT module, or the new name for an existing pin. This is a required argument if you are using the internal_node_name argument to create a connection between this pin and an internal node.

- *internal_node_name*

  Optional replaceable string that specifies the hierarchical name of an internal node to connect an EDT pin to. This argument defines the connection between an EDT pin and the internal node of an I/O pad for internally placed EDT logic.

  When defining the connection, you must also include a pin_name argument in the same command. If only one string is specified, the tool interprets it as the pin_name.

- *edt_pin*

  Optional literal that specifies an EDT control pin. Literal options include the following:

  Clock — edt_clock pin.

  Reset — edt_reset pin. Also see description for the -active option.

  Scan_enable — top-level scan_en pin.

  Bypass — edt_bypass pin.

  Single_bypass_chain — edt_single_bypass_chain pin.

  Update — edt_update pin.

  COnfiguration — edt_configuration pin for dual configuration applications.

  Master_clock — master_clock pin. Use in latch-based EDT logic for LSSD designs only.

  Slave_clock — slave_clock pin. Use in latch-based EDT logic for LSSD designs only.

  Low_power_shift_en — low power shift enable pin for low-power decompressor.

  Note

  The tool assumes that the pin you specify for the bypass enable signal is active high. If you have pre-existing bypass logic, specified with the “set_edt_options -bypass_logic use_existing_bypass_chains” command, and your bypass enable pin is active low, you must edit the generated RTL for EDT and add an inversion on the bypass signal coming into the EDT logic. With this change, when you set your bypass pin low, the tool configures all of your bypass muxes and the tool-inserted muxes to be in bypass mode.

- -active {low | high}

  Optional switch that specifies whether the edt_reset pin is active low or active high at the module boundary. You should specify this to be compatible with the polarity of the reset port/pin specified with the *pin_name* option.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This option is available for both asynchronous and IJTAG reset. By default, asynchronous reset signals are active high and IJTAG reset signals are active low.

This option is only valid for EDT reset pins. Specifying this option for any other pins causes the tool to report the following error:

```
// Error: Active state can be specified only for the EDT reset pin.
```

- [-Noinv | -Inv]

  Optional switch that specifies an EDT pin signal is inverted between the point where it is connected during EDT logic creation and the chip boundary in the final netlist used for pattern generation. The default is no inversion.

  You should specify -inv in the following situations:

  - **External flow** — Inversion between the wrapper created by the tool and the chip boundary.
  - **Internal flow** — Inversion between the internal connection node for the EDT pin and the chip boundary.

  Specifying this inversion information during the EDT logic creation phase generates the correct dofiles and test procedure files for the pattern generation phase but has no effect on the EDT logic.

- -ON_edt_module

  Optional switch that specifies a pin on the EDT module in the EDT RTL. Use this argument to apply a change to a specified pin at the EDT module level only.

- -Pipeline_stages *number_of_channel_pipeline_stages* **(Pattern Generation Phase only)**

  An optional switch and non-negative integer pair that specifies the number of pipeline stages on specified input or output channels. Use the channel *index* argument to apply a number of channel pipeline stages to a specific channel. If you do not use an *index* argument, the specified number of channel pipeline stages applies to all input or all output channels for the current EDT block.

  Use this argument during the pattern generation phase to specify the number of pipeline stages inserted between top-level pins or pads and EDT channel inputs or outputs.

  If your design contains channel pipeline stages, you must also make some changes in the test procedure file to accommodate them.

  Note

  The tool updates the number of pipeline stages with what was traced even if you have specified a different number using the -pipeline switch. If you have disabled EDT Finder, the tool does not update the number of pipeline stages.

  You cannot create test patterns for existing bypass chains from identical test patterns created for EDT, they must be newly generated. For more information on using bypass

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

chains defined in the core logic, see "Generating EDT Logic When Bypass Logic Is Defined in the Netlist"in the *Tessent TestKompress User's Manual*.

For more information, see "Using Pipeline Stages Between Pads and Channel Inputs or Outputs" in the *Tessent TestKompress User's Manual*.

- [Output_channel -CHange_edge_at_compactor_output {Any | LEading_edge_of_edt_clock | TRailing_edge_of_edt_clock}] | [Output_channel *index*-CHange_edge_at_compactor_output {Any | TRailing_edge_of_edt_clock}]

  Optional switch and literal pair that specifies which edge the compactor output data changes on. This switch does not support latch-based EDT. Options include the following:

  Any — Specifies that data can change at either edge and no additional logic is required.

  TRailing_edge_of_edt_clock— Specifies that compactor outputs need to change on the TE edge of the EDT clock. You can use the *index* option to specify a change edge for individual channels because it does not introduce an additional shift.

  When all scan chain outputs change at the same time as the LE of EDT clock, or when the compactor already has pipelining, the tool only adds a lockup cell, without a pipeline, to the end of the compactor.

  When the scan chain outputs do not change at the same time as the LE of EDT clock, the tool adds a pipeline and a lockup cell. Adding a pipeline for trailing_edge only happens when all channel outputs are specified with trailing edge.

  LEading_edge_of_edt_clock — Specifies that all compactor outputs need to change on the LE edge of the EDT clock. You cannot specify an *index*, because it may introduce varying amounts of shift cycles across output channels. When you specify this option and all scan chain outputs already change at LE, the tool adds a lockup cell and LE compactor pipeline register.

- -

  Optional literal (a single hyphen) that specifies there is no top-level pin associated with an edt_pin that is driven internally. Use this switch during EDT logic creation and test pattern generation for the internal flow and during test pattern generation (control pins only) for the external flow to prevent false K5 DRC violations.

  For more information, see "Internally Driven EDT Pins" in the *Tessent TestKompress User's Manual*.

Examples

Example 1

The following example configures the EDT logic with two scan channels, reports the names and inversion status of the EDT pins, and issues four set_edt_pins commands that do the following:

1. Share the EDT clock pin with a functional pin, a1.
2. Rename the dedicated EDT bypass pin to my_bypass and specifies the bypass signal is inverted between the PI pin and the EDT logic.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

3. Share EDT channel 1 input pin with functional pin, portain[7].
4. Change the inversion status of the edt_update pin.

A final report_edt_pins command displays the new EDT pin settings.

set_edt_options -channels 2
report_edt_pins

```
//
//    Pin description          Pin name           Inversion
//    ---------------          --------           ---------
//    Clock                    edt_clock              -
//    Update                   edt_update             -
//    Bypass mode              edt_bypass             -
//    Scan channel 1 input     edt_channels_in1       -
//      "     "    " output    edt_channels_out1      -
//    Scan channel 2 input     edt_channels_in2       -
//      "     "    " output    edt_channels_out2      -
//
```

set_edt_pins clock a1
set_edt_pins bypass my_bypass -inv
set_edt_pins input_channel 1 portain[7]
set_edt_pins update -inv
report_edt_pins

```
//
//    Pin description          Pin name           Inversion
//    ---------------          --------           ---------
//    Clock                    a1                     -
//    Update                   edt_update            inv
//    Bypass mode              my_bypass             inv
//    Scan channel 1 input     portain[7]             -
//      "     "    " output    edt_channels_out1      -
//    Scan channel 2 input     edt_channels_in2       -
//      "     "    " output    edt_channels_out2      -
//
```

Assume the settings defined by the preceding example are specified by similar set_edt_pins commands during the pattern generation phase. The next example specifies there are three pipeline stages between the scan channel 1 input pin (portain[7]) and the corresponding top level pad, and changes the inversion status of the pin. The example then displays the updated settings. (The Pin description column is not shown in this example in order to show the reporting command's output when pipeline stages are defined.)

set_edt_pins input_channel 1 -inv -pipeline_stages 3
report_edt_pins

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//
//  Pin name            Inversion       Channel pipeline stages
//  --------            ---------       -----------------------
//  a1                      -
//  edt_update             inv
//  my_bypass              inv
//  portain[7]             inv                   3
//  edt_channels_out        -                    0
//  edt_channels_in2        -                    0
//  edt_channels_out2       -                    0
```

The next example specifies new pin names to be used at the “edt” module interface for the two EDT input channels, the two EDT output channels, and the EDT clock signal:

```
//Input ch: use the name, ch_in_bus, at the edt-IP RTL module interface.
set_edt_pins input_channel ch_in_bus -on_edt_module

//Output ch: use the name, ch_out_bus, at the edt-IP RTL module interface.
set_edt_pins output_channel ch_out_bus -on_edt_module

//EDT clock: use the name tk_clk_P at the edt-IP RTL module interface.
set_edt_pins clock tk_clk_P -on_edt_module
```

The next example shows a complete dofile for an external logic location flow that implements user-specified EDT pin names at the “edt” module level:

```
dofile scan_setup.dofile

set_edt_options -channels 2 -bypass off

//
//clock: design level=PSEI, edt module level=tk_clk_P
set_edt_pins clock PSEI
set_edt_pins clock tk_clk_P -on_edt_module

//
//channel inputs: design level = AI[1]/AI[2], edt module level=ch_in_bus
set_edt_pins input_channel 1 AI[1]
set_edt_pins input_channel 2 AI[2]
set_edt_pins input_channel ch_in_bus -on_edt_module

//
//channel outputs: design level=OA[1]/OA[2], edt module level=ch_out_bus
set_edt_pins output_channel 1 OA[1]
set_edt_pins output_channel 2 OA[2]
set_edt_pins output_channel ch_out_bus -on_edt_module

set_system_mode analysis
report_edt_pins

write_edt_files created -verilog -replace
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The next example shows “edt” and “edt_top” modules (external logic location flow), with RTL changes resulting from the use of the -On_edt_module switch highlighted in bold. For brevity, only the relevant portions of the “edt_top” module are shown.

```
/*************************************************************************
** Module: my_core_edt
**
*************************************************************************/
module my_core_edt (tk_clk_P, edt_update, ch_in_bus, ch_out_bus,
                    edt_scan_in, edt_scan_out);
   input           tk_clk_P;
   input           edt_update;
   input           [ 1:0] ch_in_bus;
   output          [ 1:0] ch_out_bus;
   output          [15:0] edt_scan_in;
   input           [15:0] edt_scan_out;

   wire edt_mask;

   my_core_edt_decompressor my_core_edt_decompressor_i
                                        (.edt_clock(tk_clk_P),
                                         .edt_update(edt_update),
                                         .edt_channels_in(ch_in_bus),
                                         .edt_mask(edt_mask),
                                         .edt_scan_in(edt_scan_in));

   my_core_edt_compactor my_core_edt_compactor_i
                                        (.edt_clock(tk_clk_P),
                                         .edt_update(edt_update),
                                         .edt_scan_out(edt_scan_out),
                                         .edt_mask(edt_mask),
                                         .edt_channels_out(ch_out_bus));

endmodule
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
/*********************************************************************
** Module: my_core_edt_top
**
*********************************************************************/
module my_core_edt_top (NMOE,
                        NMWE,
                        DLM,
                        ...
                        scan_en,
                        edt_update);

   output NMOE;
   output NMWE;
   output DLM;
   ...
   input scan_en;
   input edt_update;
   ...
   my_core my_core_i (.NMOE(NMOE),
                  .NMWE(NMWE),
                  .DLM(DLM),
                  ...
                  .scan_en(scan_en));



   my_core_edt my_core_edt_i (.tk_clk_P(PSEI),
                          .edt_update(edt_update),
                          .ch_in_bus({AI[2], AI[1]}),
                          .ch_out_bus({edt_channels_out_2,
                                       edt_channels_out_1}),
                          .edt_scan_in(edt_scan_in),
                          .edt_scan_out(edt_scan_out));
   ...
endmodule
```

Example 2

The following example specifies that the output of the compactor should change at the LE of the clock.

set_edt_pins output -change_output leading

If the compactor has any outputs that change data on TE, the command adds a pipeline stage and the necessary lockup cells at the output of every compactor.

Example 3

The following example uses a design with 100 scan chains and the following parameters for compression:

set_edt_options -channels 10 -pipeline_logic_levels_in_compactor 2
set_edt_pins output -change_output leading

The example adds pipeline stages after the second XOR gates stage in the compactor. Accordingly, lockup cells are also added. However, no other pipeline stages are added at the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

output of the compactor because the compactor pipelining guarantees that the output of the compactors changes with respect to the LE of the clock.

Assume the same design and change the pipeline stages to:

```
set_edt_options -channels 10 -pipeline_logic_levels_in_compactor 6
set_edt_pins output -change_output leading
```

Now, no compactor pipeline stages are needed due to the minimum logic levels that you can use in the compactor. Under such circumstances, the last scan cell of all chains in the design are analyzed and if any of the scan outputs change at a TE edge, then the tool adds a pipeline stage at the output of all the compactors along with necessary lockup cells.

Example 4

The following example shows the commands used for a design with 100 chains and 10 channel inputs/outputs with some channel pins with pipelining.

```
set_edt_options -channels 10
set_edt_pins output 1 -change_edge trailing
set_edt_pins output 5 -change_edge trailing
```

The example adds lockup cells in the compactors that have pipeline stages in the corresponding channels.

Example 5

The following example sets the output of all compactors to change at LE except for one that changes on TE because of channel output pipelining.

```
set_edt_options -channels 10
set_edt_pins output -change_edge leading
set_edt_pins output 5 -change_edge trailing
```

This guarantees that all compactor outputs change at LE except for channel 5, which changes at TE with respect to the EDT clock.

Example 6

The following example specifies the EDT update control pin is internally driven by the /top/xyz/out node for Tessent TestKompress logic created with the internal flow.

```
set_edt_options -location internal
set_edt_pins update - /top/xyz/out
```

The following example specifies the EDT update control pin has no corresponding top-level pin during test pattern generation.

```
set_edt_pins update -
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 7

The following example specifies the connections between the channel input and output pins and their internal connection points for an internal flow. The example specifies top-level pins port_x and port_y because it uses the <*internal_node_name*> argument.

```
set_edt_options -location internal
set_edt_pins input_channel 1 port_x /ip_pad_block/port_x_pad/Z
set_edt_pins output_channel 1 port_y /ip_pad_block/port_y_pad/A
```

Example 8

The following example specifies the EDT input and output channels are internally driven by the scan_access_i/core1_edt_channel_in and scan_access_i/core1_edt_channel_out nodes, respectively, for Tessent TestKompress logic created with the internal flow.

```
set_edt_options -channel 1
set_edt_pins input_channel  1 - scan_access_i/core1_edt_channel_in
set_edt_pins output_channel 1 - scan_access_i/core1_edt_channel_out
```

Related Topics

report_edt_configurations

report_edt_pins

set_edt_options

write_edt_files

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_edt_power_controller

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Generates and inserts a power controller in the compression logic to enable low-power shift.

Usage

set_edt_power_controller **Shift** {None | Enabled | Disabled}
[-MIN_Switching_threshold_percentage *percentage_integer*]
[-Full_control {OFf | ON}]

Description

This command controls the bias of the low-power decompressor in the EDT IP creation phase. For more information, see "Low-Power Shift and Switching Thresholds" in the *Tessent TestKompress User's Manual*.

In the IP Creation phase, you can use this command to create the power controller, to specify its hardware capability (switching threshold), and to configure the generated dofile and test procedure file.

In the Pattern Generation phase, you can use this command only to indicate that the controller is present and whether it is enabled in the test procedure file read by the tool.

The power controller can be configured in both phases as either "enabled" or "disabled". If you are using the same configuration (enabled/disabled) during both IP creation and pattern generation, this command is already included in the generated dofile and you do not need to make any changes to the controller configuration. However, if you want to change how it was configured during IP creation, you need to specify the opposite value (enabled/disabled).

To generate patterns with the low power controller hardware enabled, you must do the following:

1. In the test procedure file, ensure the edt_low_power_shift_en signal is forced to the correct values during the shift procedure. For more information on the edt_low_power_shift_en signal, see "Power Controller Logic" in the *Tessent TestKompress User's Manual*.
2. Issue a "set_edt_power_controller Shift Enabled" command in Setup mode to enable the power controller before you generate test patterns.

To generate patterns with the low power controller hardware disabled, you must do the following:

1. In the test procedure file, ensure the edt_low_power_shift_en signal is forced to the correct values during the shift procedure. For more information on the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

edt_low_power_shift_en signal, see "Power Controller Logic" in the *Tessent TestKompress User's Manual*.

2. Run "set_edt_power_controller Shift Disabled" command in Setup mode to disable the power controller before you generate test patterns.

Note

You cannot use this command during ATPG to change the switching threshold of a controller that was already created; instead, use the set_power_control command to change the actual switching percentage during ATPG.

For more information, see "Low-Power Test" in the *Tessent TestKompress User's Manual*.

Arguments

- **Shift**

  A required literal that specifies the generated power controller enables low-power shift.

- **None**

  A required literal that disables creation of the power controller in the compression logic. This is the default setting.

- **Enabled**

  A required literal that, when used during IP creation, creates the power controller hardware as part of the compression logic. It also writes the ATPG dofile that enables the hardware and a test procedure file that forces the edt_low_power_shift_en control signal to its enabled value.

  During ATPG, this option tells the tool that the low power controller is enabled in the test procedure file read by the tool.

- **Disabled**

  A required literal that, when used during IP creation, creates the power controller hardware as part of the compression logic. It also writes the ATPG dofile that disables the hardware and a test procedure file that forces the edt_low_power_shift_en control signal to its disabled value.

  During ATPG, this option tells the tool that the low power controller is disabled in the test procedure file read by the tool.

- -MIN_Switching_threshold_percentage *percentage_integer*

  An optional switch and integer that specifies the minimum switching threshold value used to determine power controller parameters, such as input shift register size. Valid integer options are 1 to 50. If an integer greater than 50 is specified, the tool uses 50. The default setting is 15 percent.

  The tool uses this value to calculate the size of the power controller. The power controller must be large enough to control the number of scan chains required to achieve the minimum

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

switching threshold specified. For more information, see "Low-Power Shift and Switching Thresholds" in the *Tessent TestKompress User's Manual*.

- -Full_control {OFf | ON}

  An optional switch and literal pair that adds one bit per scan chain to control whether that chain loads a constant value. Enabling this permits the tool higher granularity in controlling the scan chain input masking. Choose one of the following options:

  OFf — Literal that disables adding these bits to the scan chain. This is the default.

  ON — Literal that enables adding one bit per scan chain.

  Normally, you only use this when your design has very few chains meeting the following conditions:

  - The low-power register count is not significant.
  - You need to control power to low thresholds or fine granularity even for those low-chain-count designs.

  When you set this switch to ON, any value specified with the -MIN_Switching_threshold_percentage switch is ignored.

  Note

  This switch does not enable you to explicitly specify the chain or chains to be controlled. The "on" value instructs the tool to insert hardware that enables the ATPG to have more fine-grained control over power thresholds. You must still specify your switching threshold with the set_power_control command, but with "-full_control on", the ATPG may be able to more closely meet the specified thresholds. Selecting "-full_control on" increases the size of the low power controller logic.

Examples

Example 1

The following example creates compression logic with a power controller configured to accommodate a minimum 20 percent switching threshold, and it creates the dofile and test procedure file that enable the hardware during pattern generation.

```
set_edt_power_controller shift enabled -min_switching_threshold_percentage 20
…
set_system_mode analysis
write_edt_files low_power_enabled
```

Given the compression logic created with the preceding command, use the following commands during the ATPG phase to turn the power controller on with a 22 percent switching threshold for scan chain loading and to generate test patterns.

```
set_power_control shift -switching_threshold_percentage 22
set_system_mode analysis
...
create_patterns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example creates compression logic with a power controller configured to accommodate a minimum 20 percent switching threshold. It also creates the dofile and test procedure file that disables the hardware during pattern generation.

```
set_edt_power_controller shift disabled -min_switching_threshold_percentage 20
…
set_system_mode analysis
write_edt_files low_power_disabled
```

Given the compression logic created with the preceding command, use the following commands to enable the power controller and set it to a 22 percent switching threshold. This application also requires that you modify the edt_low_power_shift_en pin value in the test procedure file.

```
set_edt_power_controller shift enabled
set_power_control shift -switching_threshold_percentage 22
set_system_mode analysis
...
create_patterns
```

Related Topics

report_edt_configurations

set_power_control

report_power_metrics

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_external_capture_options

Context: dft -edt, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup (patterns -scan context only), analysis

Specifies the number of tester cycles to be used in the capture window of all patterns.

Usage

set_external_capture_options {OFf | -MINimum_cycles *cycles* | -FIXed_cycles *cycles* | {-PLL_cycles *cycles timeplate_name*}} | {-CAPture_procedure *procedure_name*}

Description

The set_external_capture_options command enables you to specify a minimum or fixed number of external tester capture cycles when saving -mode_external patterns for the current test session, including chain test patterns with capture cycle and retention test patterns. The integer you supply with this command specifies the minimum or fixed number of tester cycles that must be present between the end of the load_unload procedure and the start of the next pattern, or next load_unload procedure for multiple load patterns. When saving -mode_internal patterns for purposes other than pattern retargeting, the tool ignores set_external_capture_options settings and saves the patterns created by the ATPG engine, using the clock_sequential and capture procedures as is normally done.

For techniques to handle slow scan enable transitions, see "Delaying Clock Pulses in Shift and Capture to Handle Slow Scan Enable Transitions" in the *Tessent Scan and ATPG User's Manual*.

Note

The options you specify with the set_external_capture_options command apply to all pattern retargeting patterns, including mode_internal.

The integer you supply, for fixed or minimum, adds extra padding cycles to the end of the capture window if the number of tester cycles used in the pattern is less than the specified number when saving mode-external simulation or tester interface format patterns. The ASCII and binary format patterns are not affected by these padding cycles. The number of tester cycles you designate also applies to scan loads used in multiple load patterns, so the launch cycles used between scan loads in the multiple load pattern is also padded to ensure the same number of tester cycles between scan loads.

The padding cycle is created from a copy of the last cycle from the capture procedure with all clocks turned off, all measure data removed, and all pulse-always clocks turned on. This padding cycle is added after the last cycle of the capture procedure and before the first cycle of the next pattern.

When using -PLL_cycles, instead of an annotation that lists the number of padding cycles being added, there is an annotation that lists the original ATPG sequential depth for this pattern.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When STIL and WGL patterns are read back into the ATPG tools, the tool flags padding cycles added to the end of the capture window as added placeholder cycles and does not use those padding cycles when converting the test cycles into ATPG pattern data. However, the tool uses these cycles when converting cycle-based fail information into pattern-based fail data.

As an alternative to directly specifying the number of external tester capture cycles, you can use the -CAPture_procedure switch and string pair to specify an external capture procedure that defines the external capture cycles to use with clock control definitions.

Issuing the command with no options reports the current settings. Any subsequent use of this command overrides all previous settings.

IDDQ Fault Type Usage Issues

If the fault type is set to IDDQ prior to using the set_external_capture_options command, and if you use the command to specify the fixed_cycles, pll_cycles, or external_capture option, the tool issues an error and does not set the option. This error is as follows:

```
// Error: The <option_name> option is not valid when generating IDDQ
// patterns.
```

Arguments

- OFf

  An optional literal that turns off any previous tester cycle settings. However, this option does not automatically disable maximum sequential depth. You must run “set_pattern_type -Max_sequential Off” to disable the setting for the maximum sequential depth.

- -MINimum_cycles *cycles*

  An optional switch and integer pair that specifies the minimum number of cycles present in the capture window when saving patterns in STIL or WGL format.

- -FIXed_cycles *cycles*

  An optional switch and integer pair that specifies the fixed number of tester cycles that needs to be present in the capture window when saving patterns in STIL or WGL format. The tool automatically calls “set_pattern_type -Max_sequential” to set the maximum sequential depth to be *integer* if the setting for maximum sequential depth is disabled or the maximum sequential depth is less than *cycles*.

- -PLL_cycles *cycles timeplate_name*

  An optional switch that specifies the fixed number of PLL capture cycles (integer value) and string that specifies the name of a timeplate that was loaded with the procedure file when saving patterns in STIL or WGL format. PLL_cycles can be used with clock_control definitions or with named capture procedures when they only have an internal mode. The restrictions for PLL_cycles for the hold_pi and mask_po also apply to -CAPture_procedure as well.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This switch is intended to be used with clock_control definitions only. To use the switch, all clocks must either be controlled by clock_control definitions, be constrained, or be pulse-always or pulse-in-capture. In addition, hold_pi must be on, and mask_po must be on for all outputs. The -PLL_cycles switch replaces the ATPG cycles with a set number of external capture cycles using the specified timeplate. It is somewhat similar to internal_mode and external_mode of the named capture procedures. The internal_mode is what is generated by the ATPG engine using the above restrictions, and the external_mode is the number of cycles specified with the designated timeplate, and the clocks that pulse are the pulse_always or pulse_in_capture clocks. It is possible to specify the number of pll_cycles to be less than the max_sequential depth setting. As with the -MINimum_cycles or -FIXed_cycles, -PLL_cycles is only used when saving -Mode_external patterns.

- -CAPture_procedure *procedure_name*

  An optional switch and string pair that specifies the name of an external_capture procedure, which is used for all capture cycles between each scan load, even when the pattern is a multiple load pattern. For more information about the external_capture procedure, refer to the section "Capture Procedures (Optional)" in the *Tessent Shell User's Manual*.

Examples

The following example adds one PLL capture cycle to the end of the capture window:

```
set_external_capture_options -pll_cycles 1 pll_timeplate
```

Related Topics

set_pattern_type

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_external_simulator

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the shell command to use to invoke an external simulator for verifying tool-produced patterns.

Usage

set_external_simulator {**-SIMulation_script** ***script_filename***} | {**vsim** [***options***…]}

Description

The command string must be valid for the simulator; therefore, you may need to refer to the simulator's user documentation for correct syntax and arguments.

Arguments

- **-SIMulation_script** ***script_filename***

  A required switch and string pair that specifies a simulation script to use for verifying patterns. This script is used for the analyze_simulation_mismatches command. Supported simulation tools include vsim, ncverilog, and vcs.

  For a script example, refer to this "Example" in the *Tessent Scan and ATPG User's Manual*.

- **vsim** [***options***…]

  A required literal and optional string that specifies a basic invocation command for Questa SIM in batch mode.

Examples

Example 1

The following example checks the current external simulator setting. It then specifies to invoke Questa SIM in batch mode with loading messages disabled when verifying simulation results:

report_external_simulator

```
../vsim -c
```

set_external_simulator vsim -c -quiet
report_external_simulator

```
../vsim -c -quiet
```

Questa SIM supports simulation with Standard Delay Format (SDF) back annotation. Generally, you can apply the timing information in an SDF file to any instance in a design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example includes arguments that direct the simulator to annotate maximum timing values from the SDF timing file *cpu1.sdf* to instance /my_design/inst1, and minimum timing values from the SDF timing file *cpu2.sdf* to instance /my_design/inst2:

```
set_external_simulator vsim -c -sdfmax /my_design/inst1=cpu1.sdf \
                               -sdfmin /my_design/inst2=cpu2.sdf
```

Example 3

The following example includes an argument that provides visibility into nets, ports, and registers while decreasing run time:

```
set_external_simulator vsim -c  -voptargs=+acc=npr
```

The visibility provided by the "+acc=npr" option should be sufficient for debugging advanced testbenches. For even more visibility but with less performance gain, you can modify the above example as follows:

```
set_external_simulator vsim -c  -voptargs=+acc
```

Related Topics

analyze_simulation_mismatches

report_external_simulator

report_measure_cycles

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_fails_report

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the design rules checker displays clock rule failures.

Usage

set_fails_report **OFf** | **ON**

Description

The set_fails_report command displays all clock rule failures of the design rules checker. The default mode upon invocation of the tool is off.

Arguments

- **OFf**

  A literal that specifies for the design rules checker to not display clock failures. This is the default upon invocation of the tool.

- **ON**

  A literal that specifies for the design rules checker to display clock failures.

Examples

The following example displays clock failures from the design rules checking process:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 outdata4
add_clocks 1 clock1
add_clocks 0 clock2
set_fails_report on
set_system_mode analysis
```

Related Topics

add_clocks

delete_clocks

report_clocks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_failure_mapping_options

Context: patterns -failure_mapping

Mode: analysis

Specifies filters you can apply when performing reverse mapping of top-level failures to the core in hierarchical designs flows.

Usage

set_failure_mapping_options [-CYCLE_offset *cycleOffset*] [-SHORT_core_name on | off]
[-ALLOW_missing_expected_values on | off]
[-on_chip_compare_verbose_messages {on | off}]

Arguments

- -CYCLE_offset *cycleOffset*

  An optional switch and integer pair that specifies to adjust the cycles in the top-level cycle-based failure file by the value *offset*. In hierarchical flows, Tessent Diagnosis uses the adjusted cycles during reverse mapping to the core-level failure cycles/patterns. For more information, refer to "Cycle Offset Adjustment for Failure Files" in the *Tessent Diagnosis User's Manual*.

- -SHORT_core_name on | off

  An optional switch and literal pair that specifies whether to truncate the names of core-level failure files so that they do not include the core instance pathnames of the files. At times, the addition of the core instance pathnames (the default behavior) can cause the filenames to exceed the maximum of 255 characters. When you turn this option on, the tool replaces the core instance pathnames with an index counter, "_*N*", where *N* is an integer.

- -ALLOW_missing_expected_values on | off

  An optional switch and literal pair that controls a check for missing expected values in the failure file. By default, the failure mapping process stops and reports the following error if it reads a failure file that does not contain expected values:

  ```
  //  Error: Failure file <top_flog> is missing expected values.
  //     If this is expected, please use 'set_failure_mapping_options
  //     -allow_missing_expected_values on' before reading the
  //     failure file.
  ```

  Turn this option on to relax the check and permit reading a file that does not contain these values.

- -on_chip_compare_verbose_messages on | off

  An optional switch and literal pair that controls the amount of information displayed by the error and warning messages for on-chip compare (OCComp) with SSN. By default, this option is off, so the functionality does not list the ICL instance names in the error and warning messages associated with OCComp. It is possible that the OCComp error and

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

warning messages could include hundreds of ICL instance names. Turn this option on to include the ICL instance names in the messages reported by OCComp.

Description

For hierarchical design flows, use the set_failure_mapping_options command to adjust the way that Tessent Diagnosis performs reverse mapping of the top-level failures to the core.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_fault_mode

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the fault mode is collapsed or uncollapsed.

Usage

set_fault_mode **Uncollapsed** | **Collapsed**

Description

The set_fault_mode command specifies whether the tool uses collapsed or uncollapsed fault lists for fault counts, test coverages, and fault reports. The default fault mode upon invocation of the tool is Uncollapsed. When you display a report on uncollapsed faults, the tool lists the representative fault first followed by its equivalent faults.

Note

This command is not valid for the bridge logic fault model (specified with the set_fault_type Bridge -Static command).

Arguments

- **Uncollapsed**

  A literal specifying that the tool include equivalent faults in the fault lists. This is the default mode upon invocation of the tool.

- **Collapsed**

  A literal specifying that the tool not include equivalent faults in the fault lists.

Examples

Example 1

The following example sets the fault mode to collapsed and then displays only the collapsed faults:

```
set_system_mode analysis
add_faults -all
set_fault_mode collapsed
report_faults -all
```

Example 2

The following shows an example when reporting uncollapsed tied faults as compared to reporting collapsed tied faults:

```
Uncollapsed:       Collapsed:
0 TI /I_140/I      0 TI /I_140/I
1 TI /II_140/O     1 TI /II_140/O
1 EQ /II_140/I
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

add_faults

delete_faults

read_faults

report_faults

report_testbench_simulation_options

set_fault_sampling

set_fault_type

write_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_fault_sampling

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the fault sampling percentage used for circuit evaluation or scan identification.

Usage

set_fault_sampling {**Off** | ***percentage***} [-Seed *integer*]

Description

Fault sampling enables you to process a fraction of the total faults to decrease processing time for evaluating large circuits. For faults other than bridge faults, once you specify a percentage, the tool randomly picks the fault samples to process. For bridge faults, the tool randomly picks bridges to process; if a particular bridge is included in a bridge sample, all faults associated with that bridge are included.

Arguments

- **Off**

  A switch that turns off fault sampling. This is the invocation default.

- ***percentage***

  A required, positive number that specifies the fault sampling percentage used for circuit evaluation or scan identification. Any number greater than 0 and less than or equal to 100 can be specified. The default is 100.

- -Seed *integer*

  A switch and hex number pair that specifies a seed value to use in the selection of fault samples. Specifying unique seed values for different runs can provide more accurate fault sampling results. The number value must be a lower case, 32-bit hex representation. 0 is not a valid seed value. The default is 0XCCCCCCCC.

Examples

Example 1

The following example performs a trial ATPG run in Tessent FastScan using a fault sampling of one percent of the total faults. Because the set_fault_sampling command overwrites parts of the original fault list, after the test run the example restores the fault list to the state it was in prior to the set_fault_sampling command:

```
set_system_mode analysis
add_faults -all
set_fault_sampling 1
create_patterns
delete_faults -all
set_fault_sampling 100
add_faults -all
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example performs the trial ATPG run, as in the preceding example, but uses faults from an external fault file named *my_fault_file*. The example then recovers from the set_fault_sampling command by clearing the internal fault list and reloading faults from the external file:

```
set_system_mode analysis
read_faults my_fault_file
set_fault_sampling 1
create_patterns
delete_faults -all
set_fault_sampling 100
read_faults my_fault_file
```

Related Topics

add_faults

read_faults

report_faults

set_fault_mode

set_fault_type

write_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_fault_selection_options

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Controls fault selection options for the write_faults, report_faults, and delete_faults commands.

Usage

set_fault_selection_options
[-fanout_off_path_stop {on | off}] [-fanin_off_path_stop {on | off}]

Description

The set_fault_selection_options command enables you to change the default for the -fanin_off_path_stop and -fanout_off_stop_path options. This command also specifies to report the path of the last fault selection command.

The set_fault_selection_options command without any arguments reports the current fault selection options set by the fault selection commands write_faults, report_faults, or delete_faults, including the stop cells you have specified.

Arguments

- -fanout_off_path_stop {on | off}

  This option specifies the global default of -fanout_off_path_stop option for the report_faults, write_faults, and delete_faults commands. The default is off. When you set this option to on, the default for -fanout_off_path_stop for the report_faults and write_faults commands is “on” unless you specify the option to be “off” explicitly in the commands.

- -fanin_off_path_stop {on | off}]

  An optional switch and literal pair that specifies the global default of -fanin_off_path_stop option for the report_faults, write_faults, and delete_faults commands. The default is off. When you set this option to on, the default for -fanin_off_path_stop for the report_faults and write_faults commands is “on” unless you specify the option to be “off” explicitly in the commands.

Examples

This example sets the -fanin_off_path_stop option default to on for the write_faults, report_faults, and delete_faults commands.

```
set_fault_selection_options -fanin_off_path_stop on
```

Related Topics

delete_faults

report_faults

write_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_fault_subclass_analysis

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Toggles the analysis for the specified fault class(es).

Usage

set_fault_subclass_analysis {-Enable | -Disable} {*fault_class* … }

Description

The set_fault_subclass_analysis command supports all of the fault classes and sub-classes supported by the set_relevant_coverage command, as well as the DI.EDT fault sub-class. To enable the analysis of a class of faults, you must run this command before adding the faults.

By default, AU and DI sub-classes analysis are enabled; in fact, there is no way to disable any DI sub-class other than DI.EDT. By default, the DI.EDT fault sub-class is enabled unless you have disabled EDT Finder (in which case, you cannot enable the DI.EDT fault sub-class). You can override this default and disable DI.EDT analysis. For more information about EDT Finder, refer to the set_edt_finder command.

You can use the set_relevant_coverage command to add or remove the AU and UU fault classes from the relevant test coverage reported by the tool. The UU sub-class is automatically enabled using the set_relevant_coverage command as shown in "Example 2" on page 2552. When enabled, the UU sub-class analysis is performed only for stuck, transition, and bridge fault types.

Note that AU.EDT and AU.FP sub-classes are enabled by default only if the tool can identify the EDT-related faults and false path faults. The tool can identify EDT-related faults if the EDT logic was generated with the default name or if you specified the naming of the EDT module. The tool can identify false path faults if you defined them using the add_false_paths command, or if you read the SDC file into the tool.

For more information about fault sub-classes, refer to "Fault Sub-Classes" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- -Enable

  Switch that enables analysis for the specified fault class.

- -Disable

  Switch that disables analysis for the specified fault class. There is no way to disable any DI sub-class other than DI.EDT.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- {*fault_class* … }

  Repeatable string that specifies the fault class or the DI.EDT fault sub-class. Table 8-19 on page 2825 lists the fault classes that can be specified with this command. You can specify a fault class either by name or code.

Examples

Example 1

The following example shows the report_statistics output when UU faults are classified into two sub-classes. Note, the set_fault_subclass_analysis command is issued first to enable UU analysis prior to adding faults.

```
SETUP> set_fault_subclass_analysis -enable UU
SETUP> set_system_mode analysis
...
ANALYSIS> add_faults -all
ANALYSIS> report_statistics
```

```
                            Statistics Report
                             Stuck-at Faults
-------------------------------------------------------------------------
Fault Classes                                                   #faults
                                                                (total)
-------------------------------------------------------------  -------------
  FU (full)                                                       144
  -------------------------------------------------------------  -------------
  UC (uncontrolled)                                                 4 ( 2.78%)
  UU (unused)                                                      14 ( 9.72%)
  AU (atpg_untestable)                                            126 (87.50%)
-------------------------------------------------------------------------
Fault Sub-classes
  -------------------------------------------------------
  UU (unused)
    CON (connected)                                                 1 ( 0.00%)
    FL   (floating)                                                14 ( 9.72%)
  AU (atpg_untestable)
    TC*  (tied_cells)                                             126 (87.50%)
  *Use"report_statistics -detailed_analysis" for details.
-------------------------------------------------------------------------
Coverage
  -------------------------------------------------------
  test_coverage                                                    0.00%
  fault_coverage                                                   0.00%
  atpg_effectiveness                                              97.22%
-------------------------------------------------------------------------
#test_patterns                                                        0
#simulated_patterns                                                   0
CPU_time (secs)                                                     2.7
-------------------------------------------------------------------------
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example shows how to include UU.CON faults in relevant test coverage, which automatically enables UU faults for sub-class analysis:

set_relevant_coverage -include UU.CON
set_relevant_coverage

```
Relevant Coverage Setting Report
----------------------------------------
Fault Subclass          Relevant Coverage
----------------------- -----------------
AU.BB                          yes
AU.EDT                          no
AU.PC                          yes
AU.TC                          yes
AU.CC                          yes
AU.FP                           no
AU.MCP                         yes
UU.FL                           no
UU.CON                         yes
TI                              no
BL                              no
RE                              no
----------------------------------------
```

Related Topics

report_statistics

set_relevant_coverage

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_fault_type

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the fault model type for ATPG.

Usage

```
set_fault_type {
      Stuck
    | TOggle
    | {Iddq [-NO_SEQuential_inputs_blocked | -SEQuential_inputs_blocked]
        | [-ALLOW_IDDQ_MEASURE_when_clock_active {ON | OFf}]}
    | {TRansition [-APPLY_HOLD_PI_attribute_to_shift_launch {ON | OFf}]}
    | {Path_delay [-Mask_nonobservation_points] [-ROBust_detection_only
        | -NO_Functional_detection] [-HAZARD_free_robust_detections]
        [-Expand_ambiguous_paths {ON | OFf | expansion_limit}]}
    | Bridge
    | {Udfm [-STATIC_faults]}
    | {Udfm -DELAY_faults [-APPLY_HOLD_PI_attribute_to_shift_launch {ON | OFf}]}
    | {Udfm  -IDDQ_faults [-NO_SEQuential_inputs_blocked | -SEQuential_inputs_blocked]
        | [-ALLOW_IDDQ_MEASURE_when_clock_active {ON | OFf}]}
    }
```

Description

The fault sites of all models are the input and output pins of the design cells in addition to external pins. 0 and 1 values are used to indicate the type of fault at the fault site for all fault models. Each fault model has its own separate fault collapsing according to the model's rules of equivalence.

When you change the fault type, both the current fault list and the internal pattern set are deleted.

Use the set_multiple_detection command to set the number of detections for each fault.

For more information on the different fault models, refer to the *Tessent Scan and ATPG User's Manual*.

Arguments

- Stuck

  A literal that specifies the single stuck-at fault model. This is the default upon invocation of the tool.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- TOggle

  A literal that specifies the toggle fault model. The tool supports multiple detection for the toggle fault type. If you are using a legacy simulator, the tool issues an error to indicate that it does not support toggle multiple detection.

- Iddq [-NO_SEQuential_inputs_blocked| -SEQuential_inputs_blocked | -ALLOW_IDDQ_MEASURE_when_clock_active {ON | OFf}]

  Optional literal and switch pair that specifies the Iddq fault model. Optional switches that control the treatment of Iddq faults include:

  -NO_SEQuential_inputs_blocked — An optional switch that specifies a pseudo-stuck-at model and prevents *a priori* categorization of faults as BLocked, which must propagate through sequential primitives. This enables the creation of test patterns that would not otherwise have been created, and can improve defect coverage. The additional tests created can improve defect coverage, especially for Iddq-detectable defects in scan cells. This is the default. For the pseudo-stuck-at model, many faults are marked BLocked during pre-analysis. The tool does this for cells where it cannot predict the transistors inside the cell and cannot observe to a cell output, such as a RAM cell. Such faults are not considered part of the fault population and do not show as undetected for test_coverage.

  -SEQuential_inputs_blocked — An optional switch that specifies a pseudo-stuck-at model and that retains historical behavior of *a priori* categorization of faults as BLocked which must be observed by propagation into or through a sequential primitive, such as a SL in a scan cell. For the pseudo-stuck-at model, many faults are marked BLocked during pre-analysis. This is done for cells where the tool cannot predict the transistors inside the cell and cannot know how to observe to a cell output, such as a Ram cell. Such faults are not considered part of the fault population and do not show as undetected for test_coverage.

  -ALLOW_IDDQ_MEASURE_when_clock_active ON | OFf — An optional switch and literal pair that enables the tool to measure IDDQ when the clock or clocks are active while generating IDDQ patterns. This switch is on by default.

  Note

  The -no_sequential_inputs_blocked, -sequential_inputs_blocked, and -allow_iddq_measure_when_clock_active switches behave the same for the UDFM fault model as for IDDQ faults.

  The tool generates a clock_po procedure and a timeplate used for clock_po (no pulse timing) for patterns that measure IDDQ while the clock is active if those files do not exist.

  The tool supports multiple detection for the IDDQ fault type. If you are using a legacy simulator, the tool issues an error to indicate that it does not support IDDQ multiple detection.

- TRansition

  A literal that specifies the transition fault model. The default behavior prevents the tool from creating transition tests that launch off of the last shift when it is unable to generate clock

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

sequential tests. It only considers generating transition patterns that launch in some cycle after the shift is completed (broadside).

Note

By default, the transition option does not prevent the tool from asserting the scan enable signal during at-speed tests. You must use other means (scan_enable pin constraint or capture procedure, for example) to prevent inappropriate assertion of the scan enable pin during the at-speed part of a multiple cycle sequential transition fault test.

The default behavior described applies to "udfm -delay_faults" and transition fault types.

- -APPLY_HOLD_PI_attribute_to_shift_launch ON | OFf

  An optional switch and literal pair that determines if the hold attribute defined at primary inputs for launch-off-shift patterns is used. The default for this switch is ON.

  During test generation for launch-off-shift patterns, the tool forces primary inputs with hold attribute to have the same value as the value in the last shift cycle except for the following conditions:

  - The PI is a clock pin or a read/write control pin.
  - The PI is explicitly forced in the last shift cycle by load/unload and/or shift procedures and the forced value is not the same as the pin constraint value if there is any.
  - All scan input pins are treated as TIE-X during test pattern generation.

  For these exception pins, the tool ignores the hold attribute defined at these pins.

- Path_delay

  A literal that specifies the path delay fault model.

- -Mask_nonobservation_points

  An optional switch that masks all non-observation points so path delay patterns are generated with values only at observation points. If there is already an internal pattern set in the tool and the fault type is already set to path delay, re-issuing the command with this switch alters the internal pattern response values by masking the non-observation points. If this switch was specified previously when setting the fault type to Path_delay, re-issuing the command without the switch causes the tool to re-simulate the internal patterns to restore the capture values at the non-observation points.

- -ROBust_detection_only

  An optional switch that generates only robust tests for path delay faults. By default, the tool first tries to detect a path delay fault using a robust test, but if unable to generate a robust test, uses a non-robust test or a functional test. This switch overrides the default behavior, forcing the tool to use only robust tests. If you include this switch, the tool classifies as ATPG_untestable (AU) all path delay faults that can only be tested non-robustly or functionally. For more information about robust, non-robust, functional, and hazard-free

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

detections, refer to "Path Delay Fault Detection" in the *Tessent Scan and ATPG User's Manual*.

Note

This switch is not cumulative; you must include it with each subsequent set_fault_type command or the tool returns to the default behavior.

- -NO_Functional_detection

  An optional switch that disables the generation of functional tests for path delay faults. By default, the tool first tries to detect a path delay fault using a robust test. If it is unable to generate a robust test, it uses a non-robust test or a functional test. This switch overrides the default behavior, forcing the tool to use robust tests or non-robust tests but not functional tests. If you include this switch, the tool classifies as ATPG_untestable (AU) all path delay faults that can be tested only with functional tests. For more information about robust, non-robust, functional, and hazard-free detections, refer to "Path Delay Fault Detection" in the *Tessent Scan and ATPG User's Manual*.

  Note

  This switch is not cumulative; you must include it with each subsequent set_fault_type command or the tool returns to the default behavior.

- -HAZARD_free_robust_detections

  An optional switch that specifies that the tool holds off-path inputs of the gates in a stable state when it propagates the transition through the path. When creating robust path delay patterns, this switch provides a delay test with additional restrictions than when just using the -robust_detection_only switch. This switch can be used with the -robust_detection_only switch to guarantee that any robust detection patterns created by the tool are hazard-free.

  Hazard-free robust detections reduce the chance of glitches because the output arrival time is not hampered by the arrival time of the off-path input. For more information about robust, non-robust, functional, and hazard-free detections, refer to "Path Delay Fault Detection" in the *Tessent Scan and ATPG User's Manual*.

- -Expand_ambiguous_paths {ON | OFf | *expansion_limit*}

  An optional switch and literal or integer pair that specifies whether the tool internally expands ambiguous paths to multiple unambiguous paths. An ambiguous path is a path that is only partially defined in a path definition file and has edge ambiguity, path ambiguity, or both. The default, "on", expands each ambiguous path with up to 10 unambiguous paths. Specify an integer to change the limit on the number of unambiguous paths to derive for each ambiguous path. Specify "off" to replace ambiguous paths when they are read with a single random unambiguous path.

  Internally expanded unambiguous paths act as the test alternative for faults added on ambiguous paths; hence, the numbers of fault sites and faults are unaffected.

  If you change the settings for expanding ambiguous paths with the set_fault_type command after running create_patterns, the path_delay faults, added with last create_patterns, are

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

deleted. This is because the fault classifications of those faults based on previous expanded paths may no longer be applicable when expanded paths are changed. This happens when the number of expanded paths is lowered or expanding ambiguous paths is disabled.

For more information about ambiguous paths, refer to "Path Definition Checks" in the *Tessent Scan and ATPG User's Manual*.

- Bridge

  A literal that specifies the static bridge fault model.

- Udfm

  Optional literal that specifies a user-defined fault model (UDFM).

- -STATIC_faults

  Optional literal that enables static faults only. This is the default.

- -DELAY_faults

  Optional literal that enables delay faults only. Optional switch details are described in the Transition fault model argument description.

- -IDDQ_faults

  Optional literal that enables IDDQ faults only. Optional switch details are described in the Iddq fault model argument description.

Examples

Example 1

The following example performs ATPG using the stuck fault model:

```
set_system_mode analysis
set_fault_type stuck
create_patterns
```

Example 2

The following example performs ATPG with the transition fault model and, by default, prevents the tool from generating basic combinational patterns that launch off the last shift:

```
set_system_mode analysis
set_fault_type transition
create_patterns
```

Example 3

The next patterns -scan example performs ATPG with the transition fault model, specifying to target three detections per fault during pattern generation, and to detect each DS fault three times during simulation before dropping it from the fault list. Data resulting from the multiple detection setting is highlighted in bold.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tip

When multiple detection is enabled, the number shown in the “#faults in list” column in the transcript is the cumulative tally of faults not yet detected the specified number of times. When multiple detection is not enabled, this column lists the number of faults not yet detected once. In either case, the “test cvrg” and “#faults detected” columns refer to faults detected at least once.

```
set_system_mode analysis
set_fault_type transition
set_multiple_detection -guaranteed_atpg_detections 3
set_pattern_type -sequential 2
create_patterns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//| ------------------------------------------------------------------
//|                     Analyzing the design |
//|
//|      Current clock restriction setting:      Clock_po
//|          Calling: set_clock_restriction domain_clock -any_interaction
//|
//|          Current split capture setting:      Off (optimal)
//|
//|   Current clock off simulation setting:      Off (optimal)
//|
//|            Current abort limit setting:      30
//|                               Calling:      set_abort_limit 300 100
//| -----------------------------------------------------------------
//|
//|               Current sequential depth:      0
//|               Optimal sequential depth:      2
//|    Warning: Broadside transition testing requires minimum depth of 2.
//|    Calling: set_pattern_type -sequential 2
//|
//| ---------------------------------------------------------------------
//  ATPG targeting 1 detection
//  ---------------------------------------------------------------------
//  Simulation performed for #gates = 250  #faults = 1080
//  system mode = ATPG    pattern source = internal patterns
//  ---------------------------------------------------------------------
//  #patterns  test      #faults  #faults    # eff.    # test      BCE
//  simulated  coverage  in list  detected   patterns  patterns    %
//  deterministic ATPG invoked with comb/seq abort limit = 300/100
//   ---        ------      ---       ---       ---       ---      ------
//   64         92.03%      772       993       61        61      84.06%
//   ---        ------      ---       ---       ---       ---      ------
//   128        96.47%      668        46       16        77      90.05%
//  ---------------------------------------------------------------------
//  ATPG targeting 2 detection, starting with 77 patterns
//  ---------------------------------------------------------------------
//  Simulation performed for #gates = 250  #faults = 67
//  system mode = ATPG    pattern source = internal patterns
//  ---------------------------------------------------------------------
//  #patterns     2-det. #faults  #faults    # eff.    # test      BCE
//  simulated  test cov. in list  detected   patterns  patterns    %
//  deterministic ATPG invoked with comb/seq abort limit = 300/100
//   ---        ------      ---       ---       ---       ---      ------
//   192        96.47%      510        67       28       105      93.40%
//  ---------------------------------------------------------------------
//  ATPG targeting 3 detection, starting with 105 patterns
//  ---------------------------------------------------------------------
//  Simulation performed for #gates = 250  #faults = 69
//  system mode = ATPG    pattern source = internal patterns
//  ---------------------------------------------------------------------
//  #patterns     3-det. #faults  #faults    # eff.    # test      BCE
//  simulated  test cov. in list  detected   patterns  patterns    %
//  deterministic ATPG invoked with comb/seq abort limit = 300/100
//   ---        ------      ---       ---       ---       ---      ------
//   256        96.47%      409        69       25       130      94.71%
                      Statistics Report
                      Transition Faults
-------------------------------------------------------------
Fault Classes                              #faults
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
                                          (total)
----------------------------  ---------------------------------
  FU (full)                                 1114
  --------------------------  ---------------------------------
  DS (det_simulation)                     1039          (93.27%)
  UU (unused)                               30          ( 2.69%)
  TI (tied)                                  4          ( 0.36%)
  RE (redundant)                             3          ( 0.27%)
  AU (atpg_untestable)                      38          ( 3.41%)
-----------------------------------------------------------------
Coverage
  --------------------------
  test_coverage                                           96.47%
  fault_coverage                                          93.27%
  atpg_effectiveness                                     100.00%
-----------------------------------------------------------------
#test_patterns                                                130
  #clock_sequential_patterns                                  130
#simulated_patterns                                           256
CPU_time (secs)                                               0.5
-----------------------------------------------------------------
Multiple Detection Statistics
-----------------------------------------------------------------
  Detections                          DS Faults    Test Coverage
     (N)                        (Detection == N) (Detection >= N)
----------------------------  ---------------- ----------------
      1                              0 (  0.00%)   1039 ( 96.47%)
      2                              0 (  0.00%)   1039 ( 96.47%)
      3                             93 (  8.35%)   1039 ( 96.47%)
      4                             56 (  5.03%)    946 ( 87.84%)
      5                             54 (  4.85%)    890 ( 82.64%)
      6                             56 (  5.03%)    836 ( 77.62%)
      7                             50 (  4.49%)    780 ( 72.42%)
      8                             46 (  4.13%)    730 ( 67.78%)
      9                             54 (  4.85%)    684 ( 63.51%)
     10+                           630 ( 56.55%)    630 ( 58.50%)
-----------------------------------------------------------------
bridge_coverage_estimate                              94.71%
-----------------------------------------------------------------
```

Example 4

The next example performs ATPG with the path delay fault model and only robust detection patterns. The example then repeats the pattern creation run after enabling the tool to use both robust and non-robust detection patterns. Be aware that when you change the type of test the tool can use to detect path delay faults, the tool automatically resets all AU faults identified in the preceding pattern creation run.

```
set_system_mode analysis
set_fault_type path_delay -robust_detection_only
read_faults my_path_delay_faultlist
create_patterns
...
set_fault_type path_delay -no_functional_detection
create_patterns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 5

The following example generates launch-off-shift patterns by holding all PIs:

```
set_pattern_type -sequential 0
add_input_constraints -hold -all
set_fault_type transition -apply_hold_pi_attribute_to_shift_launch on
create_patterns
```

Example 6

The following example sets the UDFM as the current fault type:

```
set_fault_type udfm
```

Related Topics

add_faults

delete_faults

identify_redundant_faults

read_fault_sites

report_faults

set_fault_mode

set_multiple_detection

write_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_flat_model_options

Context: unspecified, all contexts

Mode: setup

Controls the removal of unused library cell input ports in the flat model. Enables loadboard loopback emulation.

Usage

set_flat_model_options [-REMOVE_UNUSED_library_pins {Auto | OFf | ON}]
[-EMULATE_LOADboard_loopback_pairs {OFf | ON}]
[-ELIGIBLE_CELL_port_names_threshold_for_pin_removal *percentage*]
[-GATE_COUNT_reduction_threshold_for_pin_removal *percentage*]

Description

If you have a design that uses library cells with Vdd or ground ports that are not connected inside the cell, you can reduce gate count, memory required, and run time by removing the unused cell input ports. Do this with the -remove_unused_library_pins switch to control when or if to remove unused library cell input ports. The tool removes the corresponding instance pins for all instantiations in the read_verilog netlist and attempts to minimize the flow interruptions due to missing flat model pins referenced by tool commands or needed for backtracing.

The tool contains two levels of heuristics that attempt to provide control of what it removes, while providing a default that balances safety (preventing flow interruptions due to referencing removed flat instance pin names) with providing the benefit of a smaller flat model that has non-trivial gains.

Heuristic 1

The first heuristic attempts to identify cell inputs such as Vdd and ground. Typically, netlists that use such inputs for cells use them for all or most cells instantiated in the netlist. When the tool starts flattening the model, the first heuristic identifies all library models instantiated in the netlist and the unused ports in each cell, and determines the percentage of cells in which the port name exists and is unused. The tool counts each unique cell once regardless of the number of instantiations of each cell.

For example, it calculates the percentage, *p*, associated with the port <*portname*> as follows:

$p = (k/n) \times 100$

where *k* is the number of cells that have an unused port <*portname*> and *n* is the number of unique cells.

It then removes <*portname*> from all cells in which <*portname*> is unused if the percentage *p* exceeds the number specified by the -eligible_cell_port_names_threshold_for_pin_removal switch (or the default value 35%, if the switch is not used).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Heuristic 2

The second heuristic determines if there is enough benefit compared to the risk of removing the cell inputs such as Vdd and ground. As the inputs are identified using a heuristic, removing the inputs incurs some risk of flow interruption due to missing pins in the flat model. The tool uses the second heuristic only if at least one port exceeds the percentage specified with the -eligible_cell_port_names_threshold_for_pin_removal switch and is controlled by the -gate_count_reduction_threshold_for_pin_removal switch.

In the initial flattening stage, the tool counts the flat nodes per instance (which are approximately equivalent to the number of flat gates), as well as the approximate number of gates saved by removing unused ports, and calculates the percentage of gates saved, *pg*, as follows:

$pg = (sg/tg) \times 100$

where *sg* is the number of gates saved and *tg* is the total number of gates.

The tool uses the second heuristic if this percentage exceeds the number specified by the -gate_count_reduction_threshold_for_pin_removal switch (or the default value 5%, if the switch is not used).

Limitations

If a flat model exists when you run the "set_flat_model_options -remove_unused_library_pins" command, and the new setting would produce a flat model that differs from the existing one, a warning message is reported that you must run "delete_flat_model" for the new setting to be reflected in the flat model used subsequently.

Old flat models are considered to have been created with "set_flat_model_options -remove_unused_library_pins off" command. Any new setting except "off" causes the warning to be reported. The tool saves the necessary settings for new flat models to know if a subsequent "set_flat_model_options -remove_unused_library_pins" command would result in a different flat model. No warning is reported if re-flattening results in the same flat model.

Arguments

- -REMOVE_UNUSED_library_pins {Auto | OFf | ON}

  An optional switch and literal pair that specifies whether to remove unused cell library model input pins in the flat model and what message to report.

  Auto — This is the default. The tool does not remove any ports if the calculated reduction does not exceed the percentage specified by the "-eligible_cell_port_names_threshold_for_pin_removal *percentage*" switch, and a message is reported indicating that "approximately *p*% of the flat model gates" would be eliminated if set to "on". If the calculated reduction exceeds the percentage specified by the "-eligible_cell_port_names_threshold_for_pin_removal *percentage*" switch, a message and action similar to "on" occurs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

OFf — The tool does not remove any library cell input ports and does not report any messages.

ON — All ports identified by the first heuristic are removed. A message is reported, indicating that "approximately *p*% of the flat model gates" were eliminated, and to select "off" if any of the following ports need to be preserved, followed by the list of removed unused port names (once per unique port name).

- -EMULATE_LOADboard_loopback_pairs {OFf | ON}

  An optional switch and literal pair that specifies whether to emulate external loadboard loopbacks.

  OFf — This is the default. The tool does not emulate external loadboard loopbacks.

  ON — The tool emulates external loadboard loopbacks, where the output port drives what the input port originally drove. The tool disconnects the loopback input port and connects the loopback input port net to the loopback output port net, effectively making them one net.

- -ELIGIBLE_CELL_port_names_threshold_for_pin_removal *percentage*

  An optional switch and value pair that specifies the percentage threshold used by the first heuristic to decide whether to remove unused ports from the flat model. The default is 35%.

- -GATE_COUNT_reduction_threshold_for_pin_removal *percentage*

  An optional switch and value pair that specifies the percentage threshold used by the second heuristic to decide whether to remove unused ports from the flat model. The default is 5%.

Examples

Example 1

In this example the tool does not remove any library cell input ports and does not report any messages.

set_flat_model_options -remove_unused_library_pins off

Example 2

In this example, the tool removes all ports that exceed the -eligible_cell_port_names_threshold_for_pin_removal, regardless of the -gate_count_reduction_threshold_for_pin_removal setting, and reports a message stating the flat model reduction. The -remove_gate_reductions_percentage does not matter.

set_flat_model_options -remove_unused_library_pins on

```
Note: The following cell ports were removed for instances of cells
      inside which the port is unused :
          'vdd'  unused in 61% of all cells
          'gnd'  unused in 59% of all cells
      This reduced the flat model gate count reduced by approximately 6%.
     If any of those pins needs to be preserved in the flat model, issue
     the command 'set_flat_model_options -remove_unused_library_pins off'
     then recreate the flat model.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The tool does not report the above Note if it did not remove any ports.

Example 3

In this example, the tool removes all ports which exceed the -eligible_cell_port_names_threshold_for_pin_removal if the flat model savings calculated exceeds -gate_count_reduction_threshold_for_pin_removal, else none are removed. It reports a message indicating what would have been (or was) saved. The following message assumes default 5% gate reduction percentage, and calculated reduction of 6.3%.

set_flat_model_options -remove_unused_library_pins auto

```
Note: The following cell ports were removed from instances of cells
      inside which the port was unused :
          'vdd'  unused in 100% of all cells
          'gnd'  unused in 80% of all cells
      This reduced the flat model gate count by approximately 6%.
      If any of those pins need to be preserved in the flat model,
      issue the command
      'set_flat_model_options -remove_unused_library_pins off'
      then recreate the flat model.
```

The following message assumes default 5% gate reduction percentage, and calculated reduction of 4.3%.

```
Note: The following cell port could be removed from instances of cells
       inside which the port was unused :
          'vdd'  unused in 78% of all cells
      If no instance of that pin needs to be preserved in the flat model,
      issuing the'set_flat_model_options -remove_unused_library_pins on'
      and recreating the flat model would reduce the flat model gate count
      by approximately 4%.
```

Example 4

In this example, only ports whose names are unused in > 50% of all unique instantiated cells in the netlist are considered for removal. This only impacts the “on” (will be removed) or “auto” (removed only if calculated savings exceeds -gate_count_reduction_threshold_for_pin_removal) cases. There is no impact for the “off” case because no nodes are removed. There is only an indirect impact on messages (the list of ports removed can be fewer compared to the default of 35%), and it is possible no node is removed and therefore no message is reported.

set_flat_model_options -eligible_cell_port_names_threshold_for_pin_removal 50

Example 5

In this example, the calculated flat model savings must exceed 10% before tool removes the unused ports that exceed the -eligible_cell_port_names_threshold_for_pin_removal from the flat model. This only impacts the “auto” case. There is no impact for the “on” or “off” cases. There is only an indirect impact on messages because the list of ports is the same. It is possible, however, that the tool may not remove any (or as many) nodes, because the specified percentage threshold is changed. See Example 3 for messages that result in both of those cases.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_flat_model_options -gate_count_reduction_threshold_for_pin_removal  10
```

Related Topics

get_flat_model_option

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_flattener_rule_handling

Context: unspecified, all contexts

Mode: setup

Specifies how the tool globally handles flattening violations.

Usage

set_flattener_rule_handling ***rule_id*** [**Error** | **Warning** | NOTe | **Ignore**] [Verbose | Noverbose]

Description

The set_flattener_rule_handling command specifies the handling of the messages for net checking, pin checking, and gate checking. You can specify that the violation messages for these checks be either error, warning, note, or ignore. If you do not specify error, warning, note, or ignore, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the initial invocation setting as specified in the following list of rules.

Each rules violation has an associated occurrence message and summary message. The tool displays the occurrence message only for either error conditions or if you specify the Verbose option for that rule. The tool displays the rule identification number in all rules violation messages.

Arguments

- ***rule_id***

  A required, non-repeatable literal that specifies the identification of the exact flattening rule violations whose message handling you want to change.

  The flattening rule violations and their identification literals are divided into the following three groups: net, pin, and gate rules violation IDs.

  - Net flattening violations are described in sections “FN1” through “FN9”.
  - Pin flattening violations are described in sections “FP1” through “FP13”.
  - Gate flattening violations are described in sections “FG3” through “FG8”.

- Error

  An optional literal that specifies for the tool to display both the error occurrence message and immediately terminate the rules checking.

  If you do not specify the Error, Warning, Note, or Ignore option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Warning

  An optional literal that specifies for the tool to display the warning summary message indicating the number of times the rule was violated. If you also specify the Verbose option, the tool displays the occurrence message for each occurrence of the rules violation.

  If you do not specify the Error, Warning, Note, or Ignore option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

- NOTe

  An optional literal that specifies for the tool to display the summary message indicating the number of violations for that rule. If you also specify the Verbose option, the tool also displays the occurrence message for each occurrence of the rules violation.

  If you do not specify the Error, Warning, Note, or Ignore option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

- Ignore

  An optional literal that specifies for the tool to display no messages for the rule's violations. The tool must still enforce some rules and they must pass to enable certain functions to be performed later.

  If you do not specify the Error, Warning, Note, or Ignore option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

- NOVerbose

  An optional literal that specifies for the tool to display the occurrence message only once for the rules violation and give a summary of the number of violations.

  If you do not specify the Noverbose or Verbose option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

- Verbose

  An optional literal that specifies for the tool to display the occurrence message for each occurrence of the rules violation.

  If you do not specify the Noverbose or Verbose option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

Examples

The following example changes the handling of the FG7 flattening rule to warning and specifies that each occurrence should be listed:

set_flattener_rule_handling fg7 warning -verbose

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

report_flattener_rules

set_drc_handling

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_gate_level

Context: unspecified, all contexts

Mode: setup, analysis

Specifies the hierarchical level of gate reporting and displaying.

Usage

set_gate_level **Primitive** | **Design**

Description

The set_gate_level command specifies the hierarchical gate level at which the tool operates. This includes the reporting and schematic display of gate information. Once you set the gate level, the tool processes all subsequent commands using the new gate level.

Whenever you issue a command which invalidates the flat model, the tool also invalidates the hierarchical gate display structure. This causes Tessent Visualizer to clear the schematic view area. You can rebuild the hierarchical gate structure by creating a new flat model. To do so, either enter and exit the Setup mode, or use the create_flat_model command.

Arguments

- **Primitive**

  A literal that specifies to display gate information at the built-in primitive gate level.

- **Design**

  A literal that specifies to display gate information at the design library hierarchical gate level. These are the top-level cells of the design library which are instantiated in your design. This is the default upon invocation of the tool.

Examples

The following example sets the gate report level so that reporting and display show the simulated values of the gate and its inputs (assuming a rules checking error occurred when exiting the Setup system mode):

```
set_system_mode analysis
set_gate_level primitive
set_gate_report error_pattern
report_gates i_1006/o
```

Related Topics

create_flat_model

report_gates

set_gate_report

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_gate_report

Context: all contexts

Mode: all modes

Specifies the information displayed by the report_gates command and in the Tessent Visualizer tabs.

Usage

When EDT is Off

```
set_gate_report [
        Normal | CAPTURE_PROCedure procedure_name | CLock_cone pin_name |
        Error_pattern | Fault_status | PARallel_pattern 0..63 |
        PATtern_index number  [ -Internal | -External ] [ -SCan_test |
    -CHain_test ] [ -OBS_SCAN_UNLOAD_shift_cycle number] | SEq_depth_data |
    SImulation_context | Test_data | TIe_value | Trace |
        VCD time_value |
        Drc_pattern {
          { Test_setup | TEST_End } [ {-cycle | -time} n1 [ n2 | end ] ] |
          Load_unload | SHIft | SKew_load | SHADOW_Control | Master_observe |
          SHADOW_Observe | STATe_stability | STABLE_After_setup | STABLE_Capture |
          STABLE_Load_unload | STABLE_Shift
        } |
        DELay_data {
          LOcal_delay_data | STATIC_Arrival_time | STATIC_PRopagation_time |
          STATIC_Slack | STATIC_PAth_delay |
          { ACTUAL_Arrival_time | ACTUAL_PRopagation_time | ACTUAL_Slack |
           ACTUAL_PAth_delay } pattern_number [ -Internal | -External ]
        }
      ]
      [ -COnstrain_value { ON | OFf } ]
      [ -CLOCK_domains_in_capture_cycles { ON | OFf } ]
      [ -Timing_exceptions { ON | OFf } ]
      [ -MAX_Fanouts { number | UNlimited } ]
```

When EDT is On

```
set_gate_report [
        Normal | CAPTURE_PROCedure procedure_name | CLock_cone pin_name |
        Error_pattern | Fault_status | PARallel_pattern 0..63 |
        PATtern_index number  [ -Internal | -External ]
        [ -SCan_test | -CHain_test ] [ -OBS_SCAN_UNLOAD_shift_cycle number] |
    SEq_depth_data | SImulation_context | Test_data | TIe_value | Trace | VCD time_value |
        Drc_pattern {
          { Test_setup | TEST_End } [ {-cycle | -time} n1 [ n2 | end ] ] |
          Load_unload | SHIft | SKew_load | SHADOW_Control | Master_observe |
          SHADOW_Observe | STATe_stability | STABle_after_setup | STABLE_Capture |
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
        STABLE_Load_unload | STABLE_Shift |
        K19 { [ -Initial | -FIRst_random | -SECond_random |
              -Post_capture | -Low_power lp_sim# ]
            [ -All | Load_unload | Shift
              [ -All | cycle [ cycle ] ] ] } |
        K22 { [ -Mask No_mask | mask# ]
            [ -All | Load_unload | Shift
              [ -All | cycle [ cycle ] ] ] } |
        F_rule_name
      }
      DELay_data {
        LOcal_delay_data | STATIC_Arrival_time | STATIC_PRopagation_time |
        STATIC_Slack | STATIC_PAth_delay |
        { ACTUAL_Arrival_time | ACTUAL_PRopagation_time | ACTUAL_Slack |
         ACTUAL_PAth_delay } pattern_number [ -Internal | -External ]
      }
    ]
    [ -CLOCK_domains_in_capture_cycles { ON | OFf } ]
    [ -COnstrain_value { ON | OFf } ] [ -Timing_exceptions { ON | OFf } ]
    [ -MAX_Fanouts { number | UNlimited } ]
```

For dft -scan

```
set_gate_report [
        Normal | Error_pattern | SImulation_context | TIe_value | Trace |
        Drc_pattern {
          { Test_setup | TEST_End } [ {-cycle | -time} n1 [ n2 | end ] ] |
          Load_unload | SHIft | SKew_load | SHADOW_Control | Master_observe |
          SHADOW_Observe | STATe_stability | STABle_after_setup | STABLE_Capture |
          STABLE_Load_unload | STABLE_Shift
        }
    ]
    [ -CLOCK_domains_in_capture_cycles { ON | OFf } ]
    [ -MAX_Fanouts { number | UNlimited } ]
```

Description

The set_gate_report command specifies the information displayed by the report_gates command and the Tessent Visualizer schematic tabs.

When you exit the Setup system mode, the trace and any rules-checking error pattern results are not available with this command. For information on the format output by the different options in this command, refer to the report_gates command description.

The data field of the report created by the report_gates command displays one character for each event within the cycle.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

When EDT is On, the tool does not explicitly simulate the EDT logic during scan loading and capture cycles. Therefore, report_gates cannot report simulation values for the EDT logic during these cycles regardless of the option set with the set_gate_report command.

Arguments

- Normal

  A literal that specifies for the report_gates command to display only its standard information. This is the default mode upon invocation of the tool.

- CAPTURE_PROCedure *procedure_name*

  A literal that reports the value at each gate implied by forced and conditional assignments defined in the named capture procedure (NCP). This is useful for debugging NCP-specific issues like the inability to detect a fault or prevent bus contention.

- Clock_cone *pin_name*

  A literal and string pair that specifies the clock pin for which the report_gates command displays the clock cone data.

  The clock cone data from the report_gates command is the same data that is available as error data for clock rules violations. You can only use this option after flattening the simulation model using the create_flat_model command.

  The *pin_name* must be a valid clock pin or an error condition occurs. For example:

  ```
  set_gate_report clock_cone {{\atpgclk[0] }}
  ```

  The tool considers the pin equivalents when calculating the clock cones. State elements that the tool identifies as capturing on the clock's trailing edge does not propagate the clock effect cone. During the Setup system mode, this information is not available and the tool assumes all state elements capture with the leading edge of the selected clock.

  When you try to report gates with the clock_cone source selected and the clock cone data is no longer present, the tool issues a warning message and sets the gate report source to the default before executing the command. This can happen, for instance, when you re-read the design or issue another command that causes the flat model to be regenerated. This is the warning:

  ```
  //  Warning: The precomputed clock cone data is no longer available.
  //           Resetting the gate reporting mode.
  //           Please reissue the set_gate_report command to see clock
  //           cone data.
  ```

- Error_pattern

  A literal that specifies for the report_gates command to display the simulated value of the gates and its inputs for the pattern at which an audit error occurred.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Fault_status

  A literal that specifies for fault detection status of all gates to be preserved. Subsequent commands that cause the gate to be displayed annotates the pins with fault status data. If a schematic is currently displayed in a Tessent Visualizer schematic tab and you change the gate report data (by issuing the set_gate_report command), all fault sites are annotated with their fault detection status.

  Note that this switch is not valid for the bridge logic fault model.

  The format of the fault status data is as follows:

  <sa0-status:sa1-status> — for stuck-at faults

  <slow_to_rise-status:slow_to_fall-status> — for transition faults

  where sa0-status and sa1-status are one of the following:

  DS — Detected by simulation

  DI — Detected by implication

  PU — Possible detect untestable

  PT — Possible detect testable

  AU — Atpg untestable

  UC — Undetected uncontrolled

  UO — Undetected unobserved

  UU — Untestable unused

  BL — Untestable blocked

  TI — Untestable tied

  RE — Untestable redundant

  F — Recognized fault site, but no fault has been added yet

  N — Site is nofaulted, either because the fault site is inside a library model and internal faults are off, or because a user has nofaulted it with the add_nofaults command

  "- " — Nothing known about this pin; used for pins created by the flattening process or pins that are not fault sites (for example, the pins of an unnamed internal instance of a library cell)

  When you use the Fault_status option, subsequent report_gates commands display the fault sub-class for untestable faults (AU/UC/UO). For a complete list of these fault sub-classes, see Table 8-19.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Parallel_pattern *0..63*

  Note

  Use this option only if another command directs you to use it; for example, the analyze_bus command might transcript a message that explicitly says to use "set_gate_report parallel_pattern 0". If you use this option, be sure report_gates is the first command you enter after simulation; bogus simulation values may be reported if you enter any other tool command after simulation, but before you report_gates. The Pattern_index argument is the recommended way to specify a pattern for the report_gates command to use when displaying the value of a gate.

  A literal and integer pair that specifies the pattern number from the last simulation pass that you want the report_gates command to use when displaying the value of a gate. The pattern number must be an integer between 0 and 63.

  When reporting a sequential element, the gate report also displays in a pair of brackets ([ ]) at each output of the element. For non-scan elements, this is the value that resulted from capture and always matches the simulation value in the last frame. For scan elements, this value is the scan cell unload value, which may be different than the capture value (for example, for EDT masking values or cell constraints). For a multicycle pattern, the captured value is reported for the last cycle only, since for cycles that precede the last cycle, the captured value is already shown in the last frame.

  The bracketed value is useful because it gives you a direct view of the results of good machine simulation, which sometimes changes the value captured in the last cycle such as the examples mentioned in the previous paragraph.

- PATtern_index *pattern_index* [-Internal | -External] [-SCan_test | -CHain_test] [-OBS_SCAN_UNLOAD_shift_cycle *number*]

  A literal, integer, and optional switch triplet that specifies the pattern the report_gates command uses when displaying the value of a gate. The *pattern_index* must be a non-negative integer.

  The -OBS_SCAN_UNLOAD_shift_cycle switch is useful for reporting simulation mismatches in a serial testbench with observation scan enabled. The switch reports mismatches on a per-cycle basis when the monitor_scan_cells debugging feature is enabled. The *number* argument specifies the shift cycle of interest in the "mismatch" error message. For more information, see "Pattern Mismatch Debugging Based on Scan Cell Monitoring" in the *Hybrid TK/LBIST Flow User's Manual*.

  When reporting a sequential element, the gate report also displays in a pair of brackets ([ ]) at each output of the element. For non-scan elements, this is the value that resulted from capture and always matches the simulation value in the last frame. For scan elements, this value is the scan cell unload value, which may be different than the capture value (for example, for EDT masking values or cell constraints). For a multicycle pattern, the captured value is reported for the last cycle only, since for cycles that precede the last cycle, the captured value is already shown in the last frame.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The bracketed value is useful because it gives you a direct view of the results of good machine simulation, which sometimes changes the value captured in the last cycle such as the examples mentioned in the previous paragraph.

Glitch Reporting

When a leading-edge DFF clock is gating a leading-edge DFF structure, the tool displays the glitch on the CLK line as 0{1}00 and lets the downstream DFF capture. For example:

```
//  /sff9_pulse_inter (366)  DFF
//      "S"    I  (000-000)  188-
//      "R"    I  (000-000)  32-
//      CLK    I  (010-0{1}00)  298-/AND02_pulse_inter/Y
//      "D0"   I  (000-111) 284-
//      "OUT"  O  (100-011-[1])  152-  153-
```

The tool reports the clock glitch 0{1}00 on the input pin values of the state element only. It does not report the clock glitch on the output values of the clock driver.

False Path Reporting

The tool indicates any false path definitions in the pattern report; for example, “1{X0}0”. The “{X0}” in 1{X0}0 indicates that the value is unstable at the beginning of the second frame (leading edge frame). The instability is due to the hold time effect, and the X occurs due to the leading edge (LE) clock event on the false path source flip-flop. Because the LE clock event occurs at the second frame, “{X0}” indicates that these are the values at the LE frame.

Primitive or Design Level Gate Reporting

Depending on whether the tool is set up for primitive or design level gate reporting, you may see additional information as follows (use the report_environment command to check the current gate level):

- Primitive level (set_gate_level Primitive):

  If the sequential element you report on is part of a scan cell and the value captured by the element is not what is observed/unloaded from the scan cell for that particular pattern, the display shows “Unobs” next to the captured value and explanatory text is included indicating which element is observed for that scan cell and that pattern. “Unobs” and extra explanatory text is displayed, for example, if you report on the slave latch within an LSSD scan cell and the test procedure file includes a master_observe procedure (the slave is not observed when a master_observe procedure exists).

Note

When reporting gates at the primitive level, the tool displays captured/unload values only for the outputs of DFF and LA primitives, both scan and non-scan. Captured/unload values are not reported for transparent latches (TLAs).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Design level (set_gate_level Design):

  If the sequential element you report on is a scan cell and an output value resulting from capture does not correspond to what is observed/unloaded from the scan cell for that particular pattern, the display shows “Unobs” next to the captured value. Explanatory text is not displayed for these “Unobs” values.

For examples of captured/unload displays for different scan cells and reporting levels (primitive or design), see the Examples section.

If, after setting the gate report with this option, you alter the tool environment, be sure to reissue the “set_gate_report” command. This causes the tool to re-simulate the patterns, ensuring the values reported for a gate are up to date with the latest tool settings.

You can select from four pattern sources as specified here:

-Internal — An optional switch that specifies use of the tool’s internal pattern set. This is the default.

-External — An optional switch that specifies use of the tool’s external pattern set.

-Scan_test — An optional switch that specifies the report of the scan test pattern.

-Chain_test — An optional switch that specifies the report of the chain test pattern.

Together with the *pattern_index* argument, your option selection specifies the exact pattern to be reported as shown in Table 8-9.

Table 8-9. Pattern Sources

Display Pattern Type	scan_test	chain_test
internal	internal scan pattern (default)	internal chain pattern
external	external scan pattern	external chain pattern

- SEq_depth_data

  A literal that specifies for the report_gates command to display the calculated gate sequential depths.

  When you select a non-zero sequential depth, the tool performs a learning process to identify the minimum depths necessary to satisfy controllability and observability requirements for all gates using the current clock_sequential cells. The tool calculates both the 0-state and 1-state controllability depths. At the end of the analysis, the tool displays a summary message that indicates the largest sequential test depth, along with the largest control and observe depth. The test generator uses the sequential depth information in making decisions and avoiding paths whose sequential depth exceeds the maximum sequential depth.

  The report_gates command includes three values separated by a comma and a dash. The first value is the 0-state controllability depth, the second value is the 1-state controllability depth, and the third value is the observability depth. The maximum reported depth is 255. If controllability or observability is logically impossible or exceeds 255, the report displays an asterisk (*) in the corresponding fields.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- SImulation_context

  A literal that specifies for the report_gates command to display the simulated values from the current simulation context. Refer to "Example 11" on page 2602.

- Test_data

  A literal that specifies for the report_gates command to display previously-calculated control and observe values. To set the Test_data option, you must do so prior to rules checking, and you must re-execute the design rules checking process; otherwise, no data (-) from this option is available for gate reporting.

  This option is primarily for logic BIST purposes (when inserting control and observation points). The data for each pin of a reported gate consists of three integers indicating how many times the pin was controllable to 0, how many times it was controllable to 1, and how many times it was observable during the preceding analysis. The data is displayed in the following format:

  (# of times controlled to 0-# of times controlled to 1, # of times observed)

  Note

  When the Test_data option is in effect, the report_environment command shows "BIST data" as the current gate report setting.

- TIe_value

  A literal that specifies for the report_gates command to display the simulated values that result from all natural, tied gates and learned constant value non-scan cells.

  Tie_value is available at any time the flat model exists. The value displayed on the TIE node is set by simulation. All internal nodes are initially set to X for simulation. Subsequently, ATPG or various commands may put values on the nodes. Occasionally, a set of circumstances may occur where an X shows up on a TIE node.

  During operation, ATPG typically marks for simulation only those gates which are of interest for sensitization and propagation of a particular fault. If a gate is not marked, it is not simulated, and any nodes associated with it may remain at an "X" value, even though they are connected to a TIE gate.

  While tie_value reports the status of node values once it comes out of the DRC checks, the constrain_value tells you what the gate is suppose to act like for future consideration.

- Trace

  A literal that specifies for the report_gates command to display the simulated values of the gates during the scan chain tracing. The format of these values depends on the contents of the shift procedure. If the shift procedure contains additional frames, the additional frames are also displayed in the gate report data. The trace data relates to the simulation performed during the scan chain tracing. Use the Trace option to determine why a scan chain was not properly sensitized during the shift procedure.

  With this option, scan path "don't cares" are displayed as an S in the gate report to distinguish them from non-scan path "don't cares" which display as X.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For an example of the information displayed, refer to "Example 2" on page 2590.

- VCD *time_value*

  A literal and numerical pair that sets the gate report source to the VCD file used with the analyze_simulation_mismatches command. The Verilog simulation values after the specified simulation *time_value* can be seen using the report_gates command.

  The possible reported values are:

  0 — Logic 0

  1 — Logic 1

  Z — Tristate

  X — Unknown

  U — Unused

  . — No VCD data

- Drc_pattern *procedure_name*

  Two literals that specify the name of the procedure that the report_gates command uses to display a gate's simulated value.

  The valid literal choices for the *procedure_name* option (for use with Drc_pattern) are as follows:

  Test_setup — A literal that specifies the use of the test_setup procedure. In the test procedure file, this procedure sets non-scan elements to the state you desire for the **load_unload** procedure. The tool uses the entire test_setup procedure unless you restrict the report to a certain portion using the -Cycle or -Time switch. In order to conserve screen width, time values are listed vertically in Test_setup gate reports.

  Note

  The first time you run "set_gate_report Drc_pattern Test_setup", and the first time you run this command after running create_patterns, the tool may take a long time to generate the data for the simulated values of the gates. You can interrupt this operation by typing Ctrl-C at the command line.

  TEST_End — A literal that specifies the use of the test_end procedure. In the test procedure file, test_end is used to add a sequence of events to the end of a test pattern set. The tool uses the entire test_end procedure. In order to conserve screen width, time values are listed vertically in test_end gate reports.

  Note

  The first time you run "set_gate_report Drc_pattern TEST_End", and the first time you run this command after running create_patterns, the tool may take a long time to generate the data for the simulated values of the gates. You can interrupt this operation by typing Ctrl-C at the command line.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

-Cycle — A switch that indicates *n1*is a cycle number and specifies to start the report at cycle *n1*.

-Time — A switch that indicates *n1*is a time and specifies to start the report at time *n1*. The time units are based on the timescale defined in the test procedure file, which by default is 1 nanosecond.

n1 — An integer that specifies a cycle or time at which to start reporting. When used with the -Time switch, *n1* specifies a time.When used with the -Cycle switch, *n1* specifies a cycle. The tool numbers cycles beginning with 0; so, for example, to specify the second cycle, you would use "-cycle 1".

n2— An optional integer that specifies to report from cycle (or time) *n1* to cycle (or time) *n2* and stop reporting.

End — An optional literal that specifies to report from cycle (or time) *n1* to the end of the test_setup procedure.

Note

When using the -Cycle or -Time switch, if you do not include either the *n2* or End argument, gate reports show data for only the *n1* cycle (or time).

Load_unload — A literal that specifies the use of the load_unload procedure. The test procedure file must contain this procedure, which describes how to load and unload data in the scan chains.

SHIft — A literal that specifies the use of the shift procedure. The test procedure file must contain this procedure, which describes how to shift data one position down the scan chain.

With this option, scan path "don't cares" are displayed as S in the gate report to distinguish them from non-scan path "don't cares" which display as X.

The data in the report created by report_gates for shift data normally has three values, but if the procedure file defines four events, report_gates reports four values. One example may be when the force_pi occurs at some other time than at the beginning of the cycle. See Example 15.

SKew_load— A literal that specifies the use of the skew_load procedure. In the test procedure file, this procedure describes how to propagate the output value of the preceding scan cell into the master memory element of the current cell (without changing the slave), for all scan cells.

SHADOW_Control — A literal that specifies the use of the shadow_control procedure. In the test procedure file, this procedure describes how to load the contents of a scan cell into the associated shadow.

Master_observe— A literal that specifies the use of the master_observe procedure. In the test procedure file, this procedure describes how to place the contents of a master into the output of its scan cell.

SHADOW_Observe — A literal that specifies the use of the shadow_observe procedure. In the test procedure file, this procedure describes how to place the contents of a shadow into the output of its scan cell.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

STate_stability — A literal that specifies the display of the simulation values for the **load_unload** procedure and the capture clock cycle that the tool used to determine the constant value state elements at the initial load time. The report separates the **shift** procedure values, **load_unload** procedure values, and the capture clock cycle with parentheses. This format provides information that is helpful when you are trying to debug boundary scan.

Note

Only the simulation values for the first **load_unload** procedure that follows the **test_setup** procedure are displayed. Use the load_unload literal to display the final stable values in the **load_unload** procedure.

The state_stability option is not a procedure.

For detailed information about state stability reporting, refer to the "State Stability Issues" appendix in the *Tessent Scan and ATPG User's Manual*.

STABLE_Capture — A literal that specifies the display of the values that are constant throughout the capture procedures.

0, 1, or Z — Indicates a constant value exists throughout the capture cycles.

X — Indicates a constant value does not exist throughout the capture cycles.

STABLE_Load_unload — A literal that specifies the display of the values that are constant throughout the load_unload cycles.

0, 1, or Z — Indicates a constant value exists throughout the load_unload cycles.

X — Indicates a constant value does not exist throughout the load_unload cycles.

STABLE_Shift — A literal that specifies the display of values that are constant throughout the shift cycles.

0, 1, or Z — Indicates a constant value exists throughout the shift cycles.

X — Indicates a constant value does not exist throughout the shift cycles.

STABle_after_setup — A literal that specifies the display of the values that are constant in the period following the end of the test_setup procedure and the beginning of the test_end procedure. For scan patterns, the value is constant throughout scan and capture cycles. For ijtag patterns, the value is constant throughout any ijtag operations.

If a load_unload procedure exists, the values established at the end of test_setup are verified for stability using the add_input_constraints -c1 and c0 values which are also forced to a constant value during the load_unload procedure.

If a load_unload procedure does not exist, as in the ijtag context, stability analysis is only performed with the add_input_constraints -c1 and -c0 values.

0, 1, or Z — Indicates a constant value exists at the end of test_setup and is maintained for all subsequent events up to the beginning of test_end, if one exists, or through the end of the test if test_end does not exist.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

X — Indicates a constant value does not exist in the period following the end of test_setup and the beginning of test_end, if one exists, or through the end of the test if test_end does not exist.

This subsection describes Tessent FastScan and Tessent TestKompress gate reporting when Drc_pattern specifies a procedure.

When you set the gate report to a DRC pattern and procedure, most of the data reported by the report_gates command is simulation data regarding the specified procedure. For example, if you specify the load_unload procedure, you get simulation data regarding the load_unload procedure immediately following the test_setup procedure. Statements like **apply shift** are broken out by surrounding ()s.

The last group of data is more specialized. Its contents depend on the capture clock being set with the -atpg switch. The starting state for this simulation results from simulating the events of the test setup procedure, followed by the load_unload procedure and its apply procedures (shift and shadow_control).

- Case 1: No Capture Clock

  There are 1 or 2 values in the last pair of ()s. The first value is the simulation state that results from holding all PIs at their pin constrained value and setting all clocks to X at the end of load/unload.

  If any state element has a different binary value than the one it had at the end of simulating test setup, its value is changed to X, the affect propagated, and the final values saved in the second value between the ()s. See the following examples:

```
timeplate gen_tp1 =
   force_pi 0;
   measure_po 100;
   pulse myclk 200 100;
   period 400;
end;
procedure shift =
   scan_group grp1;
   timeplate gen_tp1;
   cycle =
      force_sci;
      measure_sco;
      pulse myclk;
   end;
end;
procedure load_unload =
   scan_group grp1;
   timeplate gen_tp1;
   cycle =
      force scan_en 1;
      force myclk 0;
   end;
   apply shift 2;
end;
report_gates clk
//  /CLK  primary_input
//      CLK    O  (0)(0X0)(0)(X)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The first (0) is the simulation of the events in the load_unload procedure prior to the apply shift, (0X0) is the simulation of the shift procedure, (0) is the simulation of the events in the load_unload after the shift, and finally (X) is the simulation of the clocks at X.

- Case 2: Capture Clock -Atpg

  There are 3 or 4 values in the last pair of ()s. The first three values result from simulating a pulse of the capture clock with all other clocks set to the off value.

  If any state element has a different binary value than the one it had at the end of simulation test setup, its value is changed to X, the effect is propagated, and the final values are saved in the fourth value between the ()s.

K19 [-Initial | -FIRst_random | -SECond_random | -Post_capture | -Low_power lp_sim#] [-All | Load_unload | {Shift [-All | cycle [cycle]]}]

Note

Use this argument only when necessary. For most common K19 debug tasks, such as checking for correct values on EDT control signals as well as sensitized paths from channel pins to the decompressor and from the decompressor to the scan chains during shift, the Drc_pattern argument is sufficient and less time-consuming than this one.

A literal and an optional, procedure pair that specifies to include a gate's simulation data resulting from the K19 design rules check (DRC) in the output displayed by the report_gates command. If you specify "K19" without any options, all K19 simulation values for every event specified in the test procedure file are displayed except test_setup events.

Note

The test_setup simulation data is available after you use the "set_gate_report drc_pattern test_setup" command.

The options enable you to limit the content of the display to the data for a certain procedure and/or for a certain cycle or range of cycles. For a detailed explanation of the K19 DRC, refer to "K19" and to "Understanding K19 Rule Violations" in the Tessent TestKompress User's Manual.

K19 reporting can slow EDT DRC run time and increase memory usage compared to Drc_pattern reporting, as the tool has to log simulation data for all simulated setup and shift cycles. Use K19 reporting when you need to see the specific simulation values applied in each shift cycle. This is not commonly required as most problems are usually due to setup problems, such as path sensitization and setting of control signals, that are usually not pattern-specific or cycle-specific.

You must set this option in Setup system mode. Also, DRC handling for the K19 rule must be set to either error (the default) or warning, but not to ignore.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following three switches specify to report simulated and emulated (where applicable) values for selected gates as described here:

-Initial — An optional switch that specifies to report the very first K19 simulation result.

-FIRst_random — An optional switch that specifies to report the first random initialization K19 simulation result.

-SECond_random — An optional switch that specifies to report the second random initialization K19 simulation result.

-Post_capture — An optional switch that specifies to report the second K19 simulation result (after capture); if low-power hardware exists and is enabled, its first simulation result is reported.

-Low_power lp_sim# — An optional switch that specifies to report the first or second simulation result for the low-power hardware, if one exists and is enabled. The lp_sim# argument can have a value of either 1 or 2.

The following three switches specify the use of the events in the test procedure file.

-All — An optional switch that specifies use of all events in the test procedure file except test_setup events. This is the default.

Load_unload — An optional literal that specifies use of the load_unload procedure.

Shift — An optional literal that specifies use of the shift procedure.

-All — An optional switch that specifies use of all shift cycles. This is the default.

cycle— A positive integer corresponding to a particular shift cycle. If you specify a single cycle number, data for only that cycle is displayed. If you provide a second integer, separated from the first by a space, the tool interprets the two integers as the endpoints of a range of cycles and displays the data for all cycles in the range. Cycles are counted from 1 and include independent shifts.

K22 [-Mask {No_mask | mask#}] [-All | Load_unload | {Shift [-All | cycle [cycle]]}]

Note

Use this argument only when necessary. For most common K22 debug tasks, such as checking for correct values on EDT control signals as well as sensitized paths from the scan chains to the compactor and from the compactor to the channel pins during shift, the Drc_pattern argument is sufficient and less time-consuming than this one.

An optional literal and procedure pair that include the gate simulation data resulting from the K22 design rules check (DRC) in the output displayed by the report_gates command. If “K22” is specified without any options, the K22 simulation values for every event specified in the test procedure file display. The options enable you to limit the content of the display to just the data associated with a certain procedure and/or to a certain cycle or range of cycles. For a detailed explanation of the K22 DRC, refer to “K22” and to “Understanding K22 Rule Violations” in the Tessent TestKompress User’s Manual.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

K22 reporting can slow EDT DRC run time and increase memory usage compared to Drc_pattern reporting, as the tool has to log simulation data for all simulated setup and shift cycles. Use K22 reporting when debugging incorrect compactor mask settings for specific patterns simulated by the design rule check. This is not commonly required, as most problems are usually due to setup problems, such as path sensitization and setting of control signals, that are usually not pattern-specific or cycle-specific.

Note

The K22 evaluations of compactor gates are synchronized with measure_sco statements in the test procedure file.

You must set this option in Setup system mode. Also, DRC handling for the K22 rule must be set to either error (the default) or warning.

-All — An optional switch that specifies use of all events in the test procedure file. This is the default.

Load_unload — An optional literal that specifies use of the load_unload procedure.

Shift — An optional literal that specifies use of the shift procedure.

-All — An optional switch that specifies use of all shift cycles. This is the default.

cycle— A positive integer corresponding to a particular shift cycle. If you specify a single cycle number, data for only that cycle is displayed. If you provide a second integer, separated from the first by a space, the tool interprets the two integers as the endpoints of a range of cycles and displays the data for all the cycles in the range. Cycles are counted from 1 and include independent shifts.

-Mask— An optional switch that sets the masking mode for the simulation data.

No_mask— A literal that together with -Mask specifies to report only non-masking mode simulation data.

mask#— A positive integer that, together with -Mask, specifies the identification number of a mask. The tool assigns a number to a mask according to the core scan chain the mask enables the tool to observe. Mask 3, for example, enables observation of the third core scan chain. This option thus limits reporting to the simulation data for the scan chain made observable by the specified mask.

F_rule_name — A literal that specifies an F rule between F1-F21.

Note, you must specify a specific occurrence of a F20 violation using the string "F20-occurrence#" where occurrence# is the violation occurrence. For more information, see the report_drc_rules command.

- DELay_data *delay_data_option*

A required literal pair that specifies to report delay data when using the report_gates command. For complete information, refer to "Timing-Aware ATPG" in the *Tessent Scan and ATPG User's Manual*.

Reported delay data is one of the following data types:

Number — Indicates delay data values.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

STAT (Static) — Indicates that no transition is possible on the pin so a delay value is not calculated or reported.

NA (Not Applicable) — Indicates the pin is not in the detection path so a delay value is not calculated or reported. (Static timing information)

- (dash) — Indicates the pin is not in the detection path so a delay value is not calculated or reported. (Actual timing information) For the *local_delay_data* literal, a dash (-) is reported in the case of a conditional I/O path delay; in this case, the timing information is included in the report as shown in Example 9.

The following *delay_data_option* literals are available:

LOCAL_delay_data

Reports the local delay data, including the wire delay (INTERCONNECT) and gate delay (IOPATH) associated with the flattened gate pin.

STATIC_Arrival_time

Reports the longest static rising and falling arrival time of the gate and direct fan-ins. Both minimal and maximum arrival time are reported in the following format:

(max_rising_arrival_delay, max_falling_arrival_delay) /

(min_rising_arrival_delay, min_falling_arrival_delay)

STATIC_PRopagation_time

Reports the longest static rising and falling propagation time of the gate and direct fan-ins.

STATIC_Slack

Reports the static test slack of the gate and direct fan-ins. The test slack is computed as the difference between the test clock period and the summation of longest arrival time and longest propagation time. The format of the reported data is:

(test_slack_of_rising, test_slack_of_falling)

STATIC_PAth_delay

Reports the longest static rising and falling path delay time through the gates. Both minimal and maximum path delay time are reported in the following format:

(max_rising_path_delay, max_falling_path_delay) /

(min_rising_path_delay, min_falling_path_delay)

ACTUAL_Arrival_time [-Internal | -External]

Reports the actual arrival time of the gate and direct fan-ins. This first simulates the pattern specified by pattern_index. When there are more than two at-speed cycles, the maximum arrival time of given cycle is reported. In addition to the actual arrival time, the simulation value of the specified pattern is also reported.

-Internal — An optional switch that specifies use of the tool's internal pattern set. This is the default for Tessent TestKompress and Tessent FastScan.

-External — An optional switch that specifies use of the tool's external pattern set.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ACTUAL_PRopagation_time [-Internal | -External]

Reports both the actual propagation time and simulation propagation time. This first simulates the pattern specified by pattern_index. When there are more than two at-speed cycles, the maximum propagation time of given cycle is reported.

-Internal — An optional switch that specifies use of the tool's internal pattern set. This is the default for Tessent TestKompress and Tessent FastScan.

-External — An optional switch that specifies use of the tool's external pattern set.

ACTUAL_Slack [-Internal | -External]

Reports both the actual and simulation slack values. This first simulates the pattern specified by pattern_index. When there are more than two at-speed cycles, the maximum slack of given cycle is reported.

-Internal — An optional switch that specifies use of the tool's internal pattern set.

-External — An optional switch that specifies use of the tool's external pattern set.

ACTUAL_Path_delay [-Internal | -External]

Reports both the actual and simulation path delay through the gates. This first simulates the pattern specified by pattern_index. When there are more than two at-speed cycles, the maximum path delay among all cycles is reported.

-Internal — An optional switch that specifies use of the tool's internal pattern set.

-External — An optional switch that specifies use of the tool's external pattern set.

Note

The calculations of arrival time and propagation time assume that all PIs change value at time 0. The delays at the clock port and the setup time of the sequential elements are ignored.

- -CLOCK_domains_in_capture_cycles {ON | OFf}

  An optional switch and literal pair indicating whether the report_gates command in the primitive level reports the clock domain information of the capture cycles. The default is off for no reporting of clock domain information.

  When this is set to ON, you can display the launch and capture clocks of a given gate in non-SETUP mode. Analysis of both the launch and capture clocks considers the constrained values, which may block a clock from being included. The report_gates command adds a "(cd)" annotation to cross clock domain input pins, which means the pin's launch and capture clocks are different. The report_gates command does not annotate output pins because the fanout to trace is unknown.

  Note

  SETUP mode does not show clock domain information because SETUP mode does not analyze capture cycle constrain values.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -COnstrain_value {ON | OFf}

  An optional switch and literal pair that specifies that the report_gates command displays the simulated values that result from all natural tied gates, learned constant value non-scan cells, constrained pins, and constrained cells, as well as the unused flag (UU) which indicates the gate is unreachable from any observation point.

  The report_gates command displays four values that are separated by a slash (/). These values are the gate constrained value (-, 0, 1, X, or Z), the gate forbidden values (-, 0, 1, Z, or any combination of 01Z), the fault blockage status (- or B, where B indicates all fault effects of this gate are blocked), and the unused flag (UU).

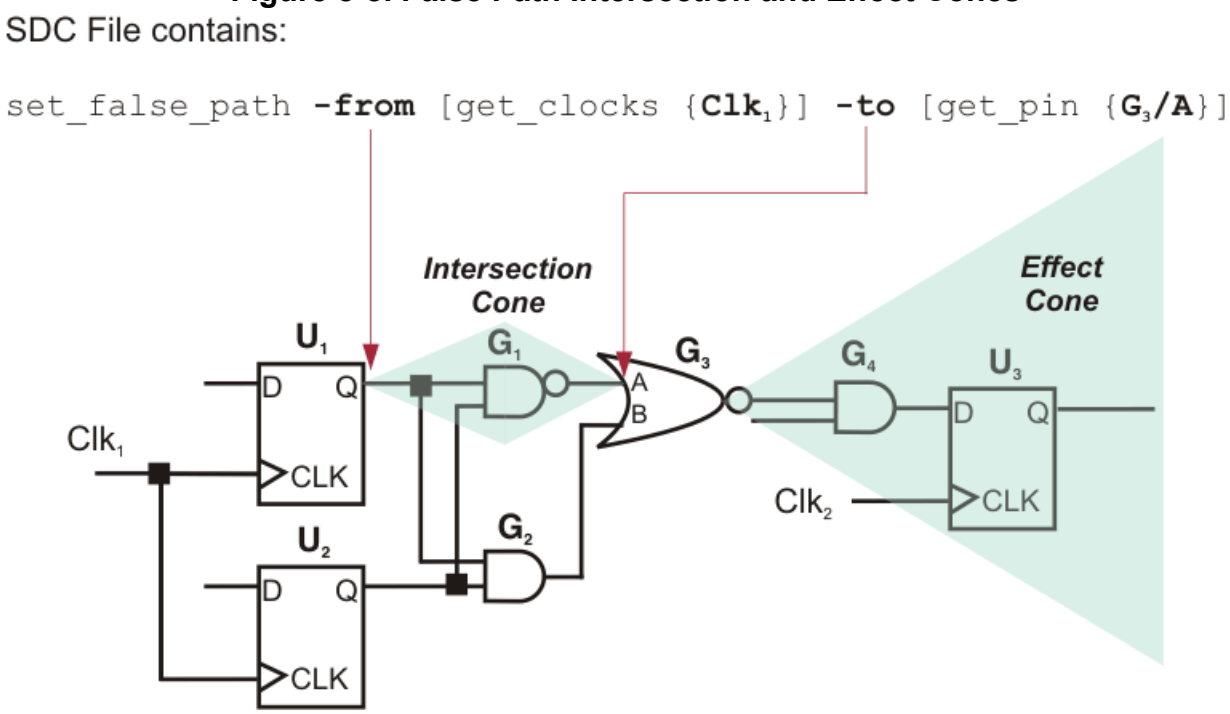
  The displayed values are only available at the end of DRC once the tool successfully leaves Setup Mode. While tie_value reports the status of node values once it comes out of the DRC checks, the constrain_value tells you what the gate is supposed to act like for future consideration. Refer to Example 14 for an example of this switch.

- -Timing_exceptions ON | OFf

  An optional switch and literal that turns on or off the timing exceptions (false paths or multicycle paths) information reporting for the report_gates command and the Tessent Visualizer schematic tabs. This option is only effective during non-Setup mode. The default is off.

  Figure 8-8 illustrates the false path definition point and the false path intersection cone and an effect cone.

Figure 8-8. False Path Intersection and Effect Cones



In this figure, the false path begins from flops in clock domain Clk1 to internal pin G3/A. The false path's intersection cone (G_1) is the logic between all -from points and the -to

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

point. The false path's effect cone (G_4 and U_3) is the combinational fanout cone of the -to point.

When you set the -Timing_exceptions switch to ON, any subsequent report_gates command can include one or more of the following false or multicycle path-related keywords in the report:

- **Fr** — Indicates the gate is a false or multicycle path from point.
- **To** — Indicates the gate is a false or multicycle path to point.
- **Th** — Indicates the gate is a false or multicycle path thru point.
- **In** — Indicates the gate is inside a false or multicycle path combinational intersection cone.
- **Ef** — Indicates the gate is in a false or multicycle path combinational effect cone.
- **--** — Indicates the gate is outside the false or multicycle path region.

The report can also include numbers, which indicate that a gate is part of at least one multicycle path. If there are two numbers, they indicate the minimum and maximum number of cycles. For example, "1,5" means the gate is on a multicycle path with minimum of 1 cycle and maximum of 5 cycles. If there is only one number, it means that the minimum and maximum number of cycles are the same. For example, "2" means the gate is on a 2-cycle path. If there is no number, then it is either on a 1-cycle path (when "--" is shown), or on a false path (when at least one of Fr, To, Th, Ef, In is shown). The following shows how the numbers display in set_gate_report command output:

```
//  /I3/reg1   sff
//      D      I  (--)   /cnt[2]
//      SI     I  (--)   /I0/i0/Q
//      SE     I  (--)   /scan_en
//      CLK    I  (--)   /clock
//      Q      O  (In/1,5)  /I3/ix28/S0  /I3/ix30/S0  /I3/ix32/S0
//      QB     O  (--)   /I3/reg3/SI
```

The "b" command with report_gates reports the first fan-in gate within that timing exception cone if it exists. If one does not exist, it reports the first fan-in gate without respect to the exception cone. Similarly, the "f" command reports the first fanout gate in that timing exception cone if it exists. If one does not exist, it reports the first fanout gate without respect to the exception cone.

The -Timing_exceptions argument does not reset the previous specified gate report option; consequently, you can use this argument in conjunction with other set_gate_report settings. Refer to "Example 8" on page 2599.

- -MAX_Fanouts {*number* | UNlimited}

  An optional switch paired with either a positive integer or keyword that specifies the maximum of fanout gates to be reported by the report_gates command. When you specify the Unlimited keyword, report_gates reports all fanout gates without truncation. The default

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is limited to 99 fanout gates. The tool issues a message if a reported gate drives more than the limit you specified with this command.

Examples

Example 1

The following example sets the gate report so that reporting and display show the simulated values of the gate and its inputs (assuming a rules checking error occurred when exiting the setup system mode):

set_system_mode analysis
set_gate_report error_pattern
report_gates i_1006/o

Example 2

The following example illustrates a shift procedure containing two clocks that are pulsed in sequence and the corresponding gate report display when the gate report is set to trace. The data displayed is with respect to the shift procedure.

```
procedure shift =
   force_sci     0;
   measure_sco   0;
   force clk   1 1;
   force clk   0 2;
   force tclk  1 3;
   force tclk  0 4;
   period        6;
end;
```

report_gates 32

```
//  /I_15 (32)  NOR
//      i0     I  (XXXXX)  16-/I_16/out
//      i1     I  (XXXXX)  17-/I_6/out
//      out    O  (XXXXX)  43-/S2
```

report_gates tclk

```
//  /TCLK (9)  PI
//      TCLK   O  (00010)  40-/I_13/clk
```

report_gates clk

```
//  /CLK (4)  PI
//      CLK    O  (01000)  20-/I_20/I_225/i0
21-/I_23/I_225/i0
```

Example 3

The following is sample output of the report_gates command for reporting path delay patterns.

set_system_mode analysis
create_patterns
set_gate_report pattern_index 0
report_gates /FF1

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /FF1  SLSDFFQ0FF
//      CLK    I  (L0-L1-L0)  /P_CLK1
//      M2SO   I  (L0-L0-L0)  /M2SOi0/Q
//      D      I  (H1-HX-HX)  /I2
//      SI2M   I  (L0-L0-L0)  /SI2Mi0/Q
//      SI     I  (0-0-0)  /FF2/SO
//      MHN    I  (L1-L0-L1)  /MHNi0/Q
//      Q      O  (0-0-1)  /IN2/A
//      SO     O  (X-X-X)  /P_SO1
```

The Ls and Hs in the gate report refer to clock-off values. These are only displayed for path-delay faults. They represent the clock off value and can be either X, L, H, or Z. The tool reports the clock off state prior to the actual pin value only for pins that are on a clock path.

The clock path value is a value that appears on a gate when pin constraints are considered and when all clocks are at their defined off states. For example, assume your design has an OR gate where one input is connected to primary input pin A. A is constrained to value 1, and the other pin is connected to primary input pin CLK, which is defined as a clock with off-state 0. The clock path values, where Y is the output of the OR gate, would be:

```
A     H
CLK   L
Y     H
```

Now, assume A is a regular input pin; that is, it has no pin constraint and is not a clock. The clock path values now are:

```
A     X
CLK   L
Y     X
```

Y is in the clock path and if you use “set_gate_report clock_cone CLK”, you see that Y is in the clock cone. Finally, assume that this gate is now an AND gate and A is still a regular input pin. The clock path values are:

```
A     X
CLK   L
Y     L
```

Since the clock path depends on both clocks and pin constraints, a pin that is in the clock path does not necessarily have to be in the clock cone.

An S followed by 0, 1, or X appears in patterns generated when you issue the command “set_pattern_type -Multiple_load On”. The S refers to these patterns being scan load operations.

Example 4

The following example shows sample output of the report_gates command when the report type is changed to report simulation data using the following test_setup procedure.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
timeplate gen_tp1 =
   force_pi 0;
   measure_po 10;
   pulse clk1 20 10;
   pulse clk2 20 10;
   period 40;
end;

timeplate gen_tp2 =
   force_pi 10;
   measure_po 30;
   pulse clk1 60 10;
   period 100;
end;

procedure test_setup =
   scan_group grp1;
   timeplate gen_tp1;
   // First cycle (cycle 0), one event (force)
   cycle =
      force clk1 0;
      force clk2 0;
      force reset 0;
      force A 0;
      ...
      force E 0;
   end;
   // Second cycle, three events (force, pulse on, pulse off)
   cycle =
      force B 1;
      pulse reset;
      pulse clk2;
   end;
   // Third cycle, three events (force, pulse on, pulse off)
   cycle =
      force A 1;
      pulse clk2;
   end;
   // Fourth cycle, three events (force, pulse on, pulse off)
   cycle =
      timeplate gen_tp2;
      force A 0;
      pulse clk1;
   end;
end;
```

set_gate_level primitive
set_gate_report drc_pattern test_setup
set_system_mode analysis
report_gates /ff20

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// /ff20 (54)  DFF
//    "IO"  I   15-
//    "I1"  I   14-
//    CLK   I   6-/clk2
//    D     I   7-/A
//    "OUT" O   24-

//    Cycle: 0   1   2   3
//    -----  -   ---  ---  ---
//                 11  111
//               467 801 389
//    Time:  0   000 000 000
//    -----  -   --- --- ---
//    "IO"   0   000 000 000
//    "I1"   0   000 000 000
//    CLK    0   010 010 000
//    D      0   000 111 000
//    "OUT"  X   X00 011 111
```

Cycle count starts at 0.

Force at cumulative time 40 (1st event of 2nd cycle).

Pulse off clk2 at cumulative time 70 (3rd event of 2nd cycle).
(Time is read vertically; time unit is based on timescale defined in test procedure file - 1 ns by default.)

Clock pulse on & output update are shown simultaneous.
(with zero delay)

set_gate_report drc_pattern test_setup -cycle 2 end
report_gates /ff20

```
//  /ff20 (54)  DFF
//    "IO"   I  15-
//    "I1"   I  14-
//    CLK    I  6-/clk2
//    D      I  7-/A
//    "OUT"  O  24-
//
// Cycle: 2   3
// ----- --- ---
//        11 111
//       801 389
// Time: 000 000
// ----- --- ---
// "IO"  000 000
// "I1"  000 000
// CLK   000 000
// D     111 000
// "OUT" 000 000
```

Example 5

The following examples show sample output of the report_gates command for sequential elements when the report type is set to report simulation data for a particular pattern. For more information about what the tool displays for each cycle, refer to "Event Simulation for DFFs and Latches" in the *Tessent Scan and ATPG User's Manual*.

- Mux-DFF — primitive level

  The DFF captured 1 (shown in "[]") at the end of the cycles. This DFF is also the observe element for this scan cell. Therefore, the captured value is also the unload value.

  set_gate_report pattern 6
  set_gate_level primitive
  report_gates /re1

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /re1 (26)  MUX
//      TE     I  (000-000)  19-/scan_en
//      D      I  (000-000)  2-/d[1]
//      TI     I  (000-000)  80-/re0/Q
//      "OUT"  O  (000-000)  81-
//  /re1 (81)  DFF
//      "S"    I  (000-000)  20-
//      "R"    I  (000-000)  21-
//      CP     I  (000-010)  62-/rh1/CPB
//      "D0"   I  (000-111)  26-
//      Q      O  (000-011-[1]) 103-/sout[0]
//      MASTER  cell_id=0  chain=chain0  group=group0
invert_data=FFFF
//  Reported all gates in instance re1
```

- Mux-DFF — design level

set_gate_report pattern 6
set_gate_level design
report_gates /re1

```
//  /re1  FD1SQM1D
//      D      I  (000-000)  /d[1]
//      CP     I  (000-010)  /rh1/CPB
//      TI     I  (000-000)  /re0/Q
//      TE     I  (000-000)  /scan_en
//      Q      O  (000-011-[1]) /sout[0]
```

- Non-scan DFF — primitive level

set_gate_level primitive
report_gates /rh1/UDP1

```
//  /rh1/UDP1 (25)  INV
//      "I0"   I  (011-111)  79-
//      "OUT"  O  (100-000)  62-
//  /rh1/UDP1 (79)  TLA
//      "S"    I  (000-000)  20-
//      "R"    I  (000-000)  21-
//      "C0"   I  (010-000)  61-
//      "D0"   I  (111-111)  38-
//      "OUT"  O  (011-111)  25-
//  Reported all gates in instance rh1/UDP1
```

- Non-scan DFF — design level

set_gate_level design
report_gates /rh1/UDP1

```
//  /rh1  LCLKRHDELLD350D
//      CP     I  (101-111)  /inv0/Z
//      LD     I  (000-000)  /ld[0]
//      TE     I  (000-000)  /scan_en
//      CPB    O  (100-000)  /fout1  /inv1/A  /rh11/CP
//                           /nonscan_d_b/D  /re0/CP  /re1/CP
//                           /nonscan_d_b/TI
```

- LSSD scan cell (master latch and slave latch with master_observe) — primitive level

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If reporting the master, which is the observed element, the captured value is also the unload value. If reporting the slave, an “Unobs” appears beside the captured value to remind you that this captured value is not the unload value. Notice also the additional message at the end of the report for the slave, telling you that the master (217) is the observation point.

set_gate_level primitive
report_gates 217 218

```
//  /I26/L1 (217)  LA
//      "S"    I  (000-000)  26-
//      "R"    I  (000-000)  20-
//      CLOCK  I  (010-000)  164-/clk_regen/I_crinv_c1/I2/OUT
//      "D0"   I  (111-111)  154-
//      SCANCLOCK  I  (000-000)  193-/clk_regen/I_inv_aclk3/I2/OUT
//      SCANIN  I  (111-111)  104-/testblock/msff_output/I12/OUT
//      Q      O  (111-111-[1])  33-/I26/I130/IN
//      MASTER  cell_id=1  chain=sc0  group=g1  invert_data=FFFF
//  /I26/L2 (218)  LA-IL
//      "S"    I  (000-000)  130-
//      "R"    I  (000-000)  20-
//      CLOCK  I  (000-010)  203-/clk_regen/I_crinv_mc2/I2/OUT
//      DATA   I  (111-111)  33-/I26/I130/OUT
//      Q      O  (111-111-[1 Unobs])  34-/I26/I12/IN
//      SLAVE  cell_id=1  chain=sc0  group=g1  invert_data=FFFF
//      Latch is not the observation point for reported pattern. //
      Observation point is at MASTER gate 217.
//      Learned latch port types: IL
```

- Masked LSSD scan cell — primitive level

  If the cell is masked (unload value = X due to an XX cell constraint added with the add_cell_constraints command, for example), the tool informs you that the cell is not observed for this pattern.

  set_gate_level primitive
  report_gates 311

```
// /ix301/slave (311) LA
// "S"   I (000-000) 43-
// "R"   I (000-000) 206-
// "C0"  I (000-010) 116-
// "D0"  I (000-000) 59-
// "OUT" O (111-100 [X Unobs]) 58-
// SLAVE cell_id=5 chain=chain2 group=grp_lssd invert_data=TTFT
//    Scan cell not observed for this pattern.
```

- LSSD scan cell, master and slave with master_observe — design level

  Notice that the tool does not show explanatory text at the design level for “Unobs” cells.

  set_gate_level design
  report_gates 217 218

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /I26/L1 (217)  LA
//      "S"    I  (000-000)  26-
//      "R"    I  (000-000)  20-
//      CLOCK  I  (010-000)  164-/clk_regen/I_crinv_c1/I2/OUT
//      "D0"   I  (111-111)  154-
//      SCANCLOCK  I  (000-000)  193-/clk_regen/I_inv_aclk3/I2/OUT
//      SCANIN  I  (111-111)  104-/testblock/msff_output/I12/OUT
//      Q      O  (111-111-[1])  33-/I26/I130/IN
//      MASTER  cell_id=1  chain=sc0  group=g1  invert_data=FFFF
//  /I26/L2 (218)  LA-IL
//      "S"    I  (000-000)  130-
//      "R"    I  (000-000)  20-
//      CLOCK  I  (000-010)  203-/clk_regen/I_crinv_mc2/I2/OUT
//      DATA   I  (111-111)  33-/I26/I130/OUT
//      Q      O  (111-111-[1 Unobs])  34-/I26/I12/IN
//      SLAVE  cell_id=1  chain=sc0  group=g1  invert_data=FFFF
//      Latch is not the observation point for reported pattern.
//      Observation point is at MASTER gate 217.
//      Learned latch port types: IL
```

Example 6

The following example sets the gate report so that reporting and display show all the K19-simulated values of each reported gate:

```
set_gate_report drc_pattern k19
// Warning: Data will be accessible after running DRC.
```

The report_environment command would show the following for the gate report:

```
gate report = display K19 sim data, procedure(s): test_setup,
load_unload, shift - all cycles
```

The following is sample output of the report_gates command when the report type is changed prior to DRC to report K19 simulation data for the following load_unload procedure and shift cycles 8 through 10. The example begins in Setup system mode.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
timeplate gen_tp1 =
   force_pi 0;
   measure_po 1;
   pulse clk 2 1;
   pulse edt_clock 2 1;
   period 4;
end;procedure load_unload =
   scan_group grp1;
   timeplate gen_tp1;
   cycle =
      force clk 0;
      force edt_bypass 0;
      force edt_clock 0;
      force edt_update 1;
      force scan_en 1;
      pulse edt_clock;
   end;
   apply shift 11;
end;
procedure shift =
   scan_group grp1;
   timeplate gen_tp1;
   cycle =
      force_sci;
      force edt_update 0;
      measure_sco;
      pulse clk;
      pulse edt_clock;
   end;
end;
```

set_gate_report drc_pattern k19 load_unload 8 10

```
// Warning: Data will be accessible after running DRC.
```

set_system_mode analysis
report_gates /edt_i/edt_decompressor_i/ix159/Y

```
// /edt_i/edt_decompressor_i/ix159 (142) NXOR
//       A0 I 130-/edt_i/edt_decompressor_i/ix241/Y
//       A1 I 84-/edt_i/edt_decompressor_i/lfsm_lockup_4/Q
//       Y  O 154-/edt_i/ix90/A
// Proc: ld_un ~ shi 8 sh 9 sh10
// ----- ----- ~ ----- ---- ----
// Time: i0234 ~ i0123 0123 0123
// ----- ----- ~ ----- ---- ----
// Sim:  XXX00 ~ 00000 0001 1110
// Emu:  ----- ~ 0---0 ---1 ---0
// No mismatch.
// Monitor: EDT decompressor chain 1 output.
```

In the preceding report, consecutive cycles are listed from left to right, with a "~" indicating a break in the data. A blank space separates each cycle's data from that of neighboring cycles. The numbers in the row labeled "Time" are the relative times of each event in a cycle as specified in the procedure file. An "i" indicates the start of a new sequence of events. Because the gate is one the tool monitors for the purpose of pinpointing where K19 errors first occur, the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report shows the values the tool emulated for the gate and whether there were any mismatches between the emulated and simulated values.

Example 7

The following example sets the gate report so reporting and display use the K22 simulated values for the load_unload procedure of the preceding example and shift cycles 4-8 for the fourth core scan chain.

set_gate_report drc_pattern k22 -mask 4 4 8

```
// Warning: Data will be accessible after running DRC.
```

The report_environment command would show the following for the gate report:

```
gate report = display K22 sim data mask 4, procedure(s):
     load_unload, shift - range 4 - 8
```

The following is sample output of the report_gates command when the report type is changed prior to DRC. The example begins in Setup system mode.

```
set_gate_report drc_pattern k22 -mask 4 4 8
// Warning: Data will be accessible after running DRC.
set_system_mode analysis
```

report_gates /edt_i/edt_bypass_logic_i/ix7/Y

```
// /edt_i/edt_bypass_logic_i/ix7 (216) OR
//      "I0"   I 215-
//       B0    I 76-/edt_i/edt_bypass_logic_i/ix5/Y
//       Y     O 256-/edt_channels_out1
//
// Proc: ld_un ~ shi 4 sh 5 sh 6 sh 7 sh 8
// ----- ----- ~ ----- ---- ---- ---- ----
// Time: i 234 ~ i 123 123  123  123 123
// ----- ----- ~ ----- ---- ---- ---- ----
// Sim:  11000 ~ 11100 0011 1111 1100 0011
// Emu:  ----- ~ --0-- -1-- -0-- -0-- -1--
// Mism:       ~   *    *    *    *    *
// Monitor: EDT module channel 1 output.
```

The report format is similar to that of the K19 report. Here too, the gate is one the tool monitors for the purpose of analyzing where K22 errors first occur, so in addition to the simulation results, the report shows the values the tool emulated for the gate and whether there were any mismatches. Notice that the emulated values occur earlier in each cycle than with the K19 DRC. The reason is that the K22 DRC synchronizes emulation comparisons with the measure_sco statements in the shift procedure, whereas the K19 comparisons always occur at the end of each shift cycle.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

If you enter a constraint using a tool command, Tessent TestKompress does not apply it until the end of the test_setup procedure. You may thus see Xs up until the end of test setup; however, the tool does correctly apply the constraint before the start of load_unload or capture.

Example 8

The following example demonstrates the reporting of the timing exception and an internal pattern simulation value, and illustrates using the report_gates command to trace the transition of the pattern along the timing exception paths:

set_gate_report -timing_exceptions on
set_gate_report pattern_index 1
report_gates /ix21/UD1
report_gates /ix21/UD1

```
// /ix21/UD1 (36) DFF
// "S" I (0-0)(--) 6-
// "R" I (0-1)(--) 20-
// "C0" I (1-0)(--) 10-
// "D0" I (1-1)(To) 32-
// "OUT" O (1-1 [0])(In/Ef) 9-
// MASTER cell_id=2 chain=chain1 group=grp1 invert_data=FFFT
```

f

```
// /ix21/UD1 (9) BUF
// "I0" I (1-1)(Fr/Ef) 36-
// "OUT" O (1-1)(In) 18- 19-
```

f

```
// /ix21 (18) BUF
// "I0" I (1-1)(In) 9-
// Q O (1-1)(To) 28-/fdgd/A 29-/ix19/A
```

f

```
// /fdgd (28) NAND
// A I (1-1)(To) 18-/ix21/Q
// B I (1-0)(--) 1-/s
// Z O (0-1)(Ef) 37-/y[2]
```

f

```
// /y[2] (37) PO
// "I0" I (0-1)(Ef) 28-/fdgd/Z
// y[2] O (0-1)(Ef)
```

Example 9

The following example reports the static propagation time of gate /U1:

set_gate_report delay_data static_propagation
report_gates /U1

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /U1 (24)  XOR
//      AO     I  (5.4000, 5.5000)/(5.0000, 5.1000)  1-/PI1
//      A1     I  (2.4000, 2.4000)/(2.0000, 2.0000)  16-/U11/Y
//      Y      O  (2.4000, 2.0000)/(2.4000, 2.0000)  25-/U12/Y
```

The following example reports the static slack test of /U2:

set_gate_report DELay_data STATIC_Slack
report_gates /U2

```
//  /U2  sff
//      D      I  (5.0000/2.0000)   /U21/Y
//      SI     I  (12.0000/12.0000) /U22/Q
//      SE     I  (NA/NA)  /SE
//      CLK    I  (8.0000/8.0000)   /CLK
//      RB     I  (STAT) /U23/Y
//      Q      O  (2.0000/3.0000)   /U24/SI /U25/A0 /U26/A1
```

The following example reports the local delay data for /U3. The output in bold font highlights the timing information used by ATPG. Note, to report local delay data, you must set the gate level to primitive before executing the report_gates command as shown here:

set_gate_level primitive
set_gate_report delay local_delay_data
report_gates U3

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /U3 (50)  BUF
//      "I0"   I  (-)  101-
//      Q      O  (-)  85-/U31/SI
//  /U3 (51)  INV
//      "I0"   I  (-)  101-
//      QB     O  (-)
//  /U3 (84)  MUX
//      SE     I  (-)  56-
//      D      I  (-)  78-/U32/Y
//      SI     I  (-)  48-/U34/Q
//      "OUT"  O  (-)  101-
//  /U3 (101)  DFF
//      "S"    I  (-)  69-
//      "R"    I  (-)  17-
//      CLK    I  (-)  54-
//      "D0"   I  (-)  84-
//      "OUT"  O  (-)  50-  51-
//
//   Sequential IOPATH delay starts through the fanin 3 of /U3/ (101):
//        1 condition delay(s) to /U3/Q (50)
//          condition delay: (1.0000, 2.0000)
//            condition variable list:
//              V[0]=54
//            condition expression in post fix format: (V[0]) posedge
//      TimingCheck for data port at the fanin 2 of /U3/ (84):
//        SETUP - clock port at the fanin 3 of /U3/ (101)
//          condition delay: (0.7000, 0.7000)
//            condition variable list:
//              V[0]=54
//            condition expression in post fix format: (V[0]) posedge
//        HOLD - clock port at the fanin 3 of /U3/ (101)
//          condition delay: (0.2000, 0.2000)
//            condition variable list:
//              V[0]=54
//            condition expression in post fix format: (V[0]) posedge
//
//      MASTER  cell_id=1  chain=c1  group=g1  invert_data=FFFF
//  Reported all gates in instance U3
```

Example 10

The following example reports gates with MCP (multicycle path) information, listing the minimum and maximum number of cycles. This example assumes that the following MCP information has been set from an SDC (Synopsys Design Constraint) file:

```
set_multicycle_path -through { get_pin [b1/I] }  -setup 2
```

set_gate_report -timing_exceptions on
report_gates /b1/ci2b cg2/TEN /cg2/i03

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /b1/ci2b (47)  INV
//      A      I  (To/Th/1,2)  40-/b1/ci2a/Y
//      Y      O  (Ef/1,2)  49-/cg2/CLKEN
//  /cg2 (49)  NOR
//      CLKEN  I  (Ef/1,2)  47-/b1/ci2b/Y
//      TEN    I  (--)  3-/se
//      "OUT"  O  (Ef/1,2)  51-
//  /cg2/i03 (51)  TLA
//      "S"    I  (--)  9-
//      "R"    I  (--)  6-
//      "C0"   I  (--)  44-
//      "D0"   I  (Ef/1,2)  49-
//      "OUT"  O  (Ef/1,2)  11-
```

When the minimum and maximum number of cycles are the same and larger than 1, the command reports only one number. Single cycle paths and false paths do not report any number.

Example 11

The following example specifies that a subsequent report_gates command display the simulated values from the current simulation context:

set_current_simulation_context stable_after_setup
set_gate_report simulation_context
report_gates {I1 I2}

```
/I1  primary_input
      I1     O  (0)  /AND2/A0
/I2  primary_input
      I2     O  (1)  /AND2/A1
```

Example 12

The following example enables the report_gates command to show the simulation values for the first F20 violation:

set_gate_report drc_pattern F20-1

Example 13

The following example shows how to display the clock domain information of a combinational gate and a sequential gate. The -clock_domains_in_capture_cycles switch, when combined with the pattern_index switch, shows both clock domain information as well as the pattern simulation values.

set_system_mode analysis
create_patterns
set_gate_level primitive
set_gate_report pattern_index 0
set_gate_report -clock_domains_in_capture_cycles on
report_gate /ff1_3/D

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /ff1_3 (30)  MUX
//      "SE"   I  (000)  9-/scan_en
//      "D"    I  (111)  11-/pi2
//      SI     I  (000)  128-/latch3/Q
//      "OUT"      O  (111)  129-
//      Launch clock:     '/clk1' (1)
//      Capture clock:     '/clk1'  (1)
```

report_gate /dff10/QB

```
//  /dff10 (109)  BUF
//      "I0"   I  (111)  62-
//      "QB"   O  (111)  121-/chain5_1/ff5_1/D 122-/or/A1
//      Launch clock:     '/clk2' (2)
//      Capture clock:     '/clk3' (3),'/reset' (18)
```

Example 14

This example demonstrates the effect of the -constrain_value switch on the report_gates command. When you have specified the “set_gate_report -constrain_value on”, the report_gates command shows when a gate is unreachable with the unused flag (UU). The report for the INV gate in /sdff2/lat pin QN shows (X/01/-/UU):

- X — Gate constrained value. In this case the gate is constrained to X.
- 01 — Gate forbidden values are 0 and 1.
- - (hyphen) — Fault blockage status. In this case - (hyphen) means no fault blockage.
- UU — Unused flag (indicating gate is unreachable from any observation point).

set_gate_report -constrain_value on
report_gates sdff2/lat

```
//  /sdff2/lat (18)  BUF
//      "I0"   I  (X/01/-)  34-
//      Q      O  (X/01/-)  36-/so
//  /sdff2/lat (19)  INV
//      "I0"   I  (X/01/-)  34-
//      QN     O  (X/01/-/UU)
//  /sdff2/lat (34)  TLA
//      "S"    I  (0/1/-)  9-
//      "R"    I  (0/1/-)  8-
//      CLK    I  (-/-/B)  26-/sdff2/u0/OUT
//      D      I  (-/-/-)  29-/sdff2/u1/OUT
//      "OUT"  O  (X/01/-)  18-  19-
//  Reported all gates in instance 'sdff2/lat'
```

Example 15

This procedure file defines four events in the shift procedure (force_pi is set to occur at 5 ns rather than at the beginning of the cycle).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set time scale 1.000000 ns ;
set strobe_window time 1 ;
 timeplate gen_tp1 =
    force_pi 5 ;
    measure_po 10 ;
    pulse CLOCK 20 10;
    period 40 ;
 end;
 procedure shift =
    scan_group grp1 ;
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force_sci ;
        measure_sco ;
        pulse CLOCK ;
    end;
end;
procedure load_unload =
    scan_group grp1 ;
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force scan_en 1 ;
        force CLOCK 0 ;
    end ;
    apply shift 3;
 end;
 procedure test_setup =
    timeplate gen_tp1 ;
    // cycle 0 starts at time 0
    cycle =
        force IN1 0 ;
    end;
 end;
```

The report_gates command reports four values, one for each event in the cycle.

set_gate_level design
set_gate_report drc shift
report_gates .reg1

```
//  /reg1  dfsc
//      CK     I  (0010)  /CLOCK
//      D0     I  (XXXX)  /gate1/OUT
//      D1     I  (SSSS)  /reg2/QBAR
//      SC     I  (1111)  /scan_en
//      Q      O  (XXXX)  /gate2/B
//      QBAR   O  (SSSS)  /scan_out1
```

Example 16

The following example shows how you can turn on reporting of cross clock domain pins for a gate using the -clock_domains_in_capture_cycles switch. The annotation (cd) is added to input pins having cross clock domain pins.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_system_mode analysis
create_patterns
set_gate_report -clock_domains_in_capture_cycles on
report_gates U89

```
// /U89 and03
// A0 I(cd) /sync6/\Q_reg[28] /Q
// A1 I(cd) /sync1/\Q_reg[28] /Q
// A2 I(cd) /async5/\Q_reg[28] /Q
// Y O /sync6/U20/A1
```

Related Topics

report_gates

set_gate_level

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_gzip_options

Context: unspecified, all contexts

Mode: all modes

Specifies GNU gzip options to use with the GNU gzip command.

Usage

set_gzip_options [-Path *gzip_path*] [-Fast | -Best | -*digit*]

Description

The set_gzip_options command specifies GNU gzip options the tool uses when compressing or decompressing files using the GNU gzip command. When file compression handling is enabled, the tool uses GNU gzip when processing files having a .gz or .Z extension.

The default speed and amount of compression corresponds to the gzip -4 option.

Arguments

- -Path *gzip_path*

  An optional literal that specifies the full path *gzip_path* to a gzip executable file. You need to provide this pathname only if gzip is not in your normal Linux search path. If you have specified a pathname with this option, you can restore the default behavior of using your Linux path to find gzip by issuing the command with the -Path gzip option.

  The following switches all control the speed of compression:

- -Fast

  An optional switch that specifies to use the fastest compression method, which yields the least compression. This corresponds to the gzip -1 option.

- -Best

  An optional switch that specifies to use the slowest compression method, which yields the most compression. This corresponds to the gzip -9 option.

- -*digit*

  An optional switch that specifies an integer from 1 to 9 that the tool passes to gzip to control the rate of compression. You obtain the least amount of compression with -1, the greatest amount of compression with -9.

Examples

The following example ensures file compression is enabled, sets gzip compression to the fastest method, then saves a file using the .gz file naming extension in order to activate gzip file compression handling:

```
set_gzip_options -fast
write_patterns fault.pat.gz
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_icl_extraction_options

Context: all contexts

Mode: all modes

Customizes the behavior of ICL extraction.

Usage

set_icl_extraction_options [-extract_learned_muxes {ON | OFf}]
[-extract_library_muxes {ON | OFf}]
[-ignore_muxes_at_data_out_ports {ON | OFf}]
[-extract_scan_interfaces {ON | OFf}]

Description

For more information, see "ICL Extraction" in the *Tessent IJTAG User's Manual*.

Arguments

- -extract_learned_muxes {ON | OFf}

  An optional switch and literal pair that specifies whether the tool creates ICL multiplexers from structures in the design that have been identified as muxes by the static learning process. The default for this switch is ON.

- -extract_library_muxes {ON | OFf}

  An optional switch and literal pair that specifies whether the tool creates ICL multiplexers from muxes that are described in the Tessent Cell library. The default for this switch in ON.

- -ignore_muxes_at_data_out_ports {ON | OFf}

  An optional switch and literal pair that specifies that ICL Extraction should not try to identify mux-like structures downstream of DataOutPorts with the connection_rule_option attribute set to "allowed_no_destination". The default for this switch in OFf.

- -extract_scan_interfaces {ON | OFf}

  An optional switch and literal pair that specifies whether the tool automatically creates the ScanInterface specifications of the new ICL top module. This automated creation tries to derive the ScanInterfaces of the top module from the ScanInterface specifications of the ICL instances, the connections between them and the connections between the ScanInterfaces of the instances and the extracted ICL top module ports. However, in case of complex IJTAG networks, the heuristic used may produce results that do not reflect your intent. In this case, the extract_scan_interfaces switch can be used to turn off the automated ScanInterface extraction. All ScanInterface specifications must then be done by means of the commands add_icl_scan_interfaces and set_icl_scan_interface_ports. An error is reported if you try to run ICL Extraction, but ScanInterface extraction is disabled and there are no manually specified ScanInterfaces. The default of this switch is ON.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following command switches off the recognition of multiplexer library cells in ICL extraction:

```
set_icl_extraction_options -extract_library_muxes off
```

Related Topics

get_icl_extraction_options

report_icl_extraction_options

set_static_learning

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_icl_network

Context: patterns -silicon_insight -ijtag

Mode: setup

Sets the current scan path to its default state or the diagnosis result to an unknown state.

Usage

set_icl_network [-reset_current_scan_path] [-reset_diagnosis_result]
[-top_client_interface *interface_name*] [-current_scan_chain_number *chain_number*]

Description

Use this command to reset the current scan path and diagnosis results to undefined states. You can reset the scan path to its default state in which the selected inputs for all SIBs and scan muxes are set to 0. You can also clear the diagnosis results from the last diagnosis run such that the status for all scan elements are set to Unknown.

Use this command when manually debugging IJTAG networks as described in "Diagnosing and Debugging IJTAG Networks" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

This command is generally used when you want to return to a default state to perform a new analysis.

Arguments

- -reset_current_scan_path

  An optional switch that resets the current input index for all nodes in the IJTAG network to 0.

- -reset_diagnosis_result

  An optional switch that resets the result for the last diagnosis of the network. The status for all nodes is set to Unknown.

- -top_client_interface *interface_name*

  An optional switch that sets the current top IJTAG client interface. Use this switch if you have multiple top interfaces. If there is only one top client interface, this interface is the default value.

- -current_scan_chain_number *chain_number*

  An optional switch that sets the currently active scan chain number for the currently set top IJTAG client interface. Use this switch when you have multiple scan chains in a top client interface.

Related Topics

analyze_icl_suspect

assert_icl_sib

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

create_icl_flush_pattern

create_icl_setup_pattern

deassert_icl_sib

get_icl_network

get_icl_scan_mux

isolate_icl_node

report_icl_network

set_icl_scan_mux

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_icl_scan_interface_ports

Context: dft, patterns -ijtag

Mode: setup, insertion

Sets the names of the ports in the specified scan interface.

Usage

set_icl_scan_interface_ports **-name** *name* **-ports** *port_objects* [-chain *chain_name*] [-allowed_to_be_tied_externally | -allowed_no_source_externally]

Description

The set_icl_scan_instances_ports command specifies the names and connection rules of the ports in the specified scan interface previously added by add_icl_scan_interfaces that ICL extraction creates in the new ICL top module.

Arguments

- **-name** *name*

  A required switch and value pair that specifies the name of scan interface for which the list of ports is created in the new ICL top module.

- **-ports** *port_objects*

  A required switch and value pair that specifies the names of the ports that are associated with the scan interface in the new ICL top module.

- -chain *chain_name*

  An optional switch and value pair that specifies the name of the scan chain of the scan interface. This switch is necessary for a scan interface which contains multiple scan chains that were specified in the -chains argument of the add_icl_scan_interfaces command. When the -chain argument is present, the port_objects list must contain exactly one port of type scan_in and one port of type scan_out.

- -allowed_to_be_tied_externally

  An optional switch that instructs the tool to set appropriate connection rules for the ports of the specified scan interface. Depending on the function of the port and the type of the scan interface, the following connection rules are chosen:

  ScanInPort: allowed_tied

  ScanOutPort: allowed_no_destination

  Controls (client/client-TAP ScanInterface): allowed_tied_low

  Controls (host/host-TAP ScanInterface): allowed_no_destination

  By using this switch, you can request that a ScanInterface be unused at the next level up without setting that as default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the -allowed_to_be_tied_externally and the -allowed_no_source_externally switches are not specified, the ports in the extracted ICL inherit their connection_rule_option attributes from the ports they traced to or from. In the case of an output port with a single driver, the inheritance is one to one. In the case of an input port with potentially more than one fanout, the most restrictive value of all fanouts is used. For example, if an input port fans out to two ports where one has connection_rule_option = allowed_tied and the other allowed_no_source, the allowed_tied value is used as it is the more restrictive. If any port in the fanout does not define the icl_port connection_rule_option attribute, the created port does not have it either.

- -allowed_no_source_externally

  An optional switch that instructs the tool to set appropriate connection rules for the ports of the specified scan interface. Depending on the function of the port and the type of the scan interface, the following connection rules are chosen:

  ScanInPort: allowed_no_source

  ScanOutPort: allowed_no_destination

  Controls (client/client-TAP ScanInterface): allowed_no_source

  Controls (host/host-TAP ScanInterface): allowed_no_destination

  By using this switch, you can request that a ScanInterface be unused at the next level up without setting that as default.

  If the -allowed_to_be_tied_externally and the -allowed_no_source_externally switches are not specified, the ports in the extracted ICL inherit their connection_rule_option attributes from the ports they traced to or from. In the case of an output port with a single driver, the inheritance is one to one. In the case of an input port with potentially more than one fanout, the most restrictive value of all fanouts is used. For example, if an input port fans out to two ports where one has connection_rule_option = allowed_tied and the other allowed_no_source, the allowed_tied value is used as it is the more restrictive. If any port in the fanout does not define the connection_rule_option attribute, the created port does not have it either.

Examples

Example 1

The following example creates the ports tck, tdi, tdo, tms and trst for the ScanInterface TAP, and the ports tdo, tdrEn1, fromTdr1, ce, se and ue for the ScanInterface Internal in the new ICL top module:

```
set_current_design tap1
add_icl_scan_interfaces {TAP Internal}
set_icl_scan_interface_ports -name TAP -ports {tck tdi tdo tms trst}
set_icl_scan_interface_ports -name Internal -ports {tdrEn1 fromTdr1 ce se ue}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The resulting ICL model looks like this:

```
Module tap1 {

   TCKPort          tck;
   ScanInPort       tdi;
   ScanOutPort      tdo        {  Source IRMux; }
   TMSPort          tms;
   TRSTPort         trst;
   ToSelectPort     tdrEn1     {  Source sel1;  }
   ScanInPort       fromTdr1;
   ToCaptureEnPort  ce;
   ToShiftEnPort    se;
   ToUpdateEnPort   ue;

   ScanInterface TAP {Port tck; Port tdi; Port tdo; Port tms; Port trst;}
   ScanInterface Internal {Port tdrEn1; Port fromTdr1; Port ce; Port se;
Port ue;}
```

Example 2

The following example creates a scan interface with two chains with the port pairs si_1 and so_1 as well as si_2 and so_2.

```
set_current_design top
add_icl_scan_interfaces { intf_1 } -chains { chain_1 chain_2 }
set_icl_scan_interface_ports -name intf_1 -ports { ce se ue }
set_icl_scan_interface_ports -name intf_1 -ports { si_1 so_1 } -chain chain_1
set_icl_scan_interface_ports -name intf_1 -ports { si_2 so_2 } -chain chain_2
```

The resulting ICL model would look like this:

```
Module top{
   ScanInPort si_1;
   ScanInPort si_2;
   ScanOutPort so_1 {
      Source i1.so;
   }
   ScanOutPort so_2 {
      Source i2.so;
  }
   CaptureEnPort ce;
   ShiftEnPort se;
  UpdateEnPort ue;
   ScanInterface intf_1 { Port ce; Port se; Port ue;
      Chain chain_1 { Port si_1; Port so_1; }
      Chain chain_2 { Port si_2; Port so_2; }
   }
```

Related Topics

add_icl_scan_interfaces

delete_icl_scan_interfaces

get_icl_scan_interface_list

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_scan_interface_port_list

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_icl_scan_mux

Context: patterns -silicon_insight -ijtag

Mode: setup

Configures the specified scan mux to select the input indicated by the specified index.

Usage

set_icl_scan_mux ***mux_name*** {**-selected_input *input_index*** }

Description

Use this command when you are configuring a scan path in the IJTAG network.

Use this command when manually debugging IJTAG networks as described in "Diagnosing and Debugging IJTAG Networks" in the *Tessent SiliconInsight User's Manual for Tessent Shell*.

Arguments

- ***mux_name***

  A required variable that specifies the name of the scan mux.

- **-selected_input *input_index***

  A required switch and variable pair that specifies the index for the programmed selected input.

Examples

set_icl_scan_mux tap_I1.DRMux -selected_input 1

set_icl_scan_mux block3_I1.M1 -selected_input 2

Related Topics

analyze_icl_suspect

assert_icl_sib

create_icl_flush_pattern

create_icl_setup_pattern

deassert_icl_sib

get_icl_network

get_icl_scan_mux

isolate_icl_node

report_icl_network

set_icl_network

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_icl_ssn_datapath_ports

Context: all contexts

Mode: setup, insertion

Specifies the ports that belong to a given SSN datapath and defines the order of the SSN bus data ports.

Usage

set_icl_ssn_datapath_ports **-name** ***datapath_id***
[-clock_inputs *clkin_object_spec*]
[-clock_outputs *clkout_object_spec*]
[-bus_data_inputs *datain_object_spec*]
[-bus_data_outputs *dataout_object_spec*]

Description

The bus data ports and the clock ports of a SSN datapath must be associated with this datapath by means of the icl_port attribute tessent_ssn_datapath_ids, and the position of a bus data port within the bus must be described by means of the icl_port attribute tessent_ssn_datapath_bus_index.

When the top module ICL is elaborated, Tessent Shell analyzes the existing SSN datapaths and attaches the attributes to the ports, if possible and unambiguous. Usually, Tessent Shell is capable of deducing the attribute values from the topology of the SSN datapaths and from the existing SSN datapath attributes deeper down the ICL hierarchy. However, sometimes the association between the ICL top module ports and the SSN datapaths is ambiguous, for example, if there are extra output paths. In this case, the add_icl_ssn_datapaths and set_icl_ssn_datapath_ports commands must be used to instruct ICL Extraction to attach the appropriate attributes to the ICL ports of the newly created ICL top module.

The command set_icl_ssn_datapath_ports enables you to specify which ports belong to a given SSN datapath. The specifications also determine the type of the ports (input or output) and the role of the ports (bus data ports or clock ports). In addition, the specification done by means of this command determine indirectly the order of the bus data ports within the given SSN datapath—simply by using the order of elements within the given collections.

The command enables the indirect specification of the “yet to be created” ICL ports by means of internal pins, if these pins can be traced towards ports of the current design that match those ICL ports in question. The tracing is done using a custom simulation context in which all ultimate sources of the DFT signal “ssn_en” are in their active state.

More information can be found in the description of the various arguments below.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- **-name *datapath_id***

  A required switch and string pair that specifies the name of the SSN datapath this command refers to. All port specifications for this SSN datapath must be done with one single set_icl_ssn_datapath_ports command. Calling this command again for the same SSN datapath overwrites previous specifications done by means of set_icl_ssn_datapath_ports.

- -clock_inputs *clkin_object_spec*

  An optional switch and collection of objects of type port or pin. ICL Extraction creates ICL ports of type “ClockPort” for each of the given port objects. ICL Extraction traces backwards from the given pin objects and creates ICL ports of type “ClockPort” if this tracing reaches an input port or bidirectional port of the current design. If the tracing does not reach any input port or bidirectional port of the current design, an I2 violation is issued. The tracing is performed assuming that the DFT signal “ssn_en” takes its active value.

  The attribute tessent_ssn_datapath_ids is attached to each of the newly-created ICL ports of type ClockPort, with the value taken from the ***datapath_id*** argument. A ClockPort can have several SSN datapath ids.

- -clock_outputs *clkout_object_spec*

  An optional switch and collection of objects of type port or pin. ICL Extraction creates ICL ports of type “ToClockPort” for each of the given port objects. ICL Extraction traces forwards from the given pin objects and creates ICL ports of type “ToClockPort” if this tracing reaches an output port or bidirectional port of the current design. If the tracing does not reach any output port or bidirectional port of the current design, an I2 violation is issued. The tracing is performed assuming that the DFT signal “ssn_en” takes its active value.

  The attribute tessent_ssn_datapath_ids is attached to each of the newly-created ICL ports of type ToClockPort, with the value taken from the ***datapath_id*** argument. A ToClockPort can have several SSN datapath ids.

- -bus_data_inputs *datain_object_spec*

  An optional switch and collection of objects of type port or pin. ICL Extraction creates ICL ports of type “DataInPort” for each of the given port objects. ICL Extraction traces backwards from the given pin objects and creates ICL ports of type “DataInPort” if this tracing reaches an input port or bidirectional port of the current design. If the tracing does not reach any input port or bidirectional port of the current design, an I2 violation is issued. The tracing is performed assuming that the DFT signal “ssn_en” takes its active value.

  The attribute tessent_ssn_datapath_ids is attached to each of the newly-created ICL ports of type DataInPort, with the value taken from the ***datapath_id*** argument. A DataInPort can have only one SSN datapath id.

  The attribute tessent_ssn_datapath_bus_index is also attached to the newly-created ICL ports of type DataInPort. The value of this attribute represents the position of the particular port within the SSN bus. It is derived from the position of the port object or pin object within the given collection *datain_object_spec*. The rightmost element of this collection

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

gets the bus index 0. The leftmost element of the collection gets the bus index *n*-1, where *n* is the number of elements in the collection.

- -bus_data_outputs *dataout_object_spec*

  An optional switch and collection of objects of type port or pin. ICL Extraction creates ICL ports of type "DataOutPort" for each of the given port objects. ICL Extraction traces forwards from the given pin objects and creates ICL ports of type "DataOutPort" if this tracing reaches an output port or bidirectional port of the current design. If the tracing does not reach any output port or bidirectional port of the current design, an I2 violation is issued. The tracing is performed assuming that the DFT signal "ssn_en" takes its active value.

  The attribute tessent_ssn_datapath_ids is attached to each of the newly-created ICL ports of type DataOutPort, with the value taken from the ***datapath_id*** argument. A DataOutPort can have only one SSN datapath id.

  The attribute tessent_ssn_datapath_bus_index is also attached to the newly-created ICL ports of type DataOutPort. The value of this attribute represents the position of the particular port within the SSN bus. It is derived from the position of the port object or pin object within the given collection *dataout_object_spec*. The rightmost element of this collection gets the bus index 0. The leftmost element of the collection gets the bus index *n*-1, where *n* is the number of elements in the collection.

Examples

```
SETUP> add_icl_ssn_datapaths {A B}
SETUP> set_icl_ssn_datapath_ports -name A -clock_inputs bus_clock \
     -bus_data_inputs bus_a_in[1:0] -bus_data_outputs bus_a_out[1:0]
SETUP> set_icl_ssn_datapath_ports -name B -clock_inputs bus_clock \
     -bus_data_inputs bus_b_in[3:0] -bus_data_outputs bus_b_out[3:0]
SETUP> extract_icl
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This will instruct ICL extraction to create ClockPorts, DataInPorts and DataOutPorts and attach the appropriate attributes, such that the result will be as follows:

```
Module chip {
   ClockPort bus_clock {
      Attribute tessent_ssn_datapath_ids = "A B";
   }
   DataInPort bus_a_in[1] {
      Attribute tessent_ssn_datapath_ids = "A";
      Attribute tessent_ssn_datapath_bus_index = 1;
   }
   DataInPort bus_a_in[0] {
      Attribute tessent_ssn_datapath_ids = "A";
      Attribute tessent_ssn_datapath_bus_index = 0;
   }
   DataInPort bus_b_in[3] {
      Attribute tessent_ssn_datapath_ids = "B";
      Attribute tessent_ssn_datapath_bus_index = 3;
   }
   DataInPort bus_b_in[2] {
      Attribute tessent_ssn_datapath_ids = "B";
      Attribute tessent_ssn_datapath_bus_index = 2;
   }
   DataInPort bus_b_in[1] {
      Attribute tessent_ssn_datapath_ids = "B";
      Attribute tessent_ssn_datapath_bus_index = 1;
   }
   DataInPort bus_b_in[0] {
      Attribute tessent_ssn_datapath_ids = "B";
      Attribute tessent_ssn_datapath_bus_index = 0;
   }
   DataOutPort bus_a_out[1] {
      Attribute tessent_ssn_datapath_ids = "A";
      Attribute tessent_ssn_datapath_bus_index = 1;
   }
   DataOutPort bus_a_out[0] {
      Attribute tessent_ssn_datapath_ids = "A";
      Attribute tessent_ssn_datapath_bus_index = 0;
   }
   DataOutPort bus_b_out[3] {
      Attribute tessent_ssn_datapath_ids = "B";
      Attribute tessent_ssn_datapath_bus_index = 3;
   }
   DataOutPort bus_b_out[2] {
      Attribute tessent_ssn_datapath_ids = "B";
      Attribute tessent_ssn_datapath_bus_index = 2;
   }
   DataOutPort bus_b_out[1] {
      Attribute tessent_ssn_datapath_ids = "B";
      Attribute tessent_ssn_datapath_bus_index = 1;
   }
   DataOutPort bus_b_out[0] {
      Attribute tessent_ssn_datapath_ids = "B";
      Attribute tessent_ssn_datapath_bus_index = 0;
   }

}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_iddq_checks

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the restrictions and conditions to use when creating or selecting patterns for detecting IDDQ faults.

Usage

set_iddq_checks [-NONe | {[-Int_float] [-Bus] [-WEAKBus] [-Pull] [-WIre]
[-VOLTGain] [-VOLTLoss] [-Clock] [-WRite] [-Read]}
[-ATpg | -NO_ATpg] [-WArning | -ERror] [-LOOP_oscillation {OFf | REPort | REJect}]

Description

These restrictions only apply during the actual time of the IDDQ measurement; they are ignored at other times during a pattern.

To ensure no input is floating due to an unconnected module or instance input, before the design is flattened, you should issue the following commands at the beginning of Setup:

```
set_flattener_rule_handling fn1 error
set_flattener_rule_handling fn3 error
```

When you use the tool to create Iddq patterns, a violation of any of the Iddq Check options you select causes the tool to reject an offending pattern.

During simulation, whenever violations of the restrictions occur, a message displays identifying the gate associated with the violation and the number of patterns in which the violations occurred. The handling of the violation can be either a warning or error.

When you select -Error, simulation terminates at the first occurrence of a violation. You can use the report_gates command to inspect the simulation values of all gates for patterns that violate the restrictions by first using the set_gate_report command with the Error_pattern option.

The report_environment command reports the current set_iddq_check settings.

For more information on IDDQ, see "IDDQ Test Set Creation" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- -NONe

  An optional switch that specifies to not perform any checks. This is the default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Int_float — An optional switch that specifies that no ZVAL primitive gate that drives any Boolean logic can have a floating (Z) value. A ZVAL gate is placed between a source (Z producer) and a sink (nonZ consumer).

  A Z producer is any primitive that has a Z output. Z producers include top-level inputs, bidi/inouts, tristate drivers, and FETs/switches. A nonZ consumer is any gate that treats Z the same as X for all inputs. NonZ consumers include any Boolean gate. You can see the primitive truth tables for all DFT-supported primitives, in section "Supported Primitives" of the *Tessent Cell Library Manual*.

  Note

  A ZVAL gate is not placed between a Z producer and a PO or the PO half of a bidi because both of these have the ability to output a Z value. However, a ZVAL gate is placed between the PI half of the bidi and any internal _buf as typifies the input path of a bidi/inout I/O pad. For those pads, if the output stage is disabled (Z out), and the bidi/inout PAD is driven externally by Z from the tester, and the I/O pad has no modeled pullup/down, the ZVAL in front of the _buf input path can have a Z on its inputs; it fails the -int_float check. These "floating" checks only occur at buses. To ensure no input is floating due to an unconnected module or instance input, before the design is flattened, you should issue the following commands at the beginning of Setup:

  ```
  set_flattener_rule_handling fn1 error
  set_flattener_rule_handling fn3 error
  ```

- -Bus

  An optional switch that specifies that contention conditions cannot be present on bus gates.

- -WEAKBus

  An optional switch that prevents contention conditions on weak-bus gates.

- -Pull

  An optional switch that prevents contention conditions on pull gates.

- -WIre

  An optional switch that specifies that all inputs of a wire gate must be set to the same value. There should be no contention on wires during IDDQ measurements because contention can raise the IDDQ current.

- -VOLTGain

  An optional switch that specifies that a PMOS transistor must not be at a logic zero unless a zhold0 or zhold bus_keeper DFT library attribute is between the PMOS transistor and the BUS root. A compromised (Vss+) logic zero is pulled down to the Vss rail (zero volts).

- -VOLTLoss

  An optional switch that specifies that a NMOS transistor must not pass a logic one unless a zhold1 or zhold bus_keeper DFT library attribute is between the NMOS transistor and the BUS root. A compromised (Vdd-) logic one is pulled up to the Vdd rail (Vdd volts).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Clock

  An optional switch that specifies that clock pins cannot be on during the IDDQ measure.

  Note

  The advantage of the -Clock switch, during pattern generation, is that it prevents creation of Clock PO style patterns, one of the causes of multiple timeplates. The switch also ensures that when a clock is turned on, it is pulsed at the normal time.

- -WRite

  An optional switch that specifies that write control pins cannot be on during the IDDQ measure.

- -REad

  An optional switch that specifies that read control pins cannot be on during the IDDQ measure.

- -ATpg

  An optional switch that specifies to satisfy any IDDQ restrictions set using the add_iddq_exceptions command. This is the default.

- -NO_ATpg

  An optional switch that specifies to not enforce IDDQ restrictions set using the add_iddq_exceptions command. Patterns that do not satisfy restrictions are rejected by the fault simulator.

- -WArning

  An optional switch that treats violations of IDDQ checks as warnings. This is the default.

- -ERror

  An optional switch that treats violations of IDDQ checks as errors and stops the simulation process immediately.

- -LOOP_oscillation {OFf | REPort | REJect}

  An optional switch and keyword pair that checks for oscillating loops when the fault type is IDDQ. The “Report” option reports on oscillating simulation values at specific gates. The “Reject” option automatically rejects any patterns that cause oscillation and issues a warning. By default, the tool does not check for oscillating loops.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example creates IDDQ patterns while checking that internal buses are not floating during an IDDQ measure and requires the deterministic ATPG to ensure that this is true:

```
set_system_mode analysis
set_fault_type iddq
set_iddq_checks -int_float
create_patterns
```

Example 2

This example uses the report_environment command to report the set_iddq_checks settings:

```
// command: set_fault_type iddq
// command: set_iddq_checks -Bus -Int_float -warning
// command: report_environment
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
abort limit =              30
atpg compression =         OFF
atpg limits =              none
bist initialization =      X
bus simulation method =    global
capture clock =            CLK
checkpoint =               OFF
clock restriction =        domain_clocks (same-edge interaction)
clock_off simulation =     OFF (optimal)
contention check =         ON,   mode = bus,   handling = warning
DRC transient detection = ON -NOVerbose
fails report =             OFF
fault mode =               uncollapsed
fault type =               IDDQ
gate level =               design
gate report =              normal
iddq checks =              bus int_float ,   handling = warning   -atpg
iddq strobe =              label
learn reporting =          OFF
logfile handling =         OFF
net dominance =            WIRE
net resolution =           WIRE
observation point =        master
pattern classification =   ON
pattern source =           internal
pattern type =             sequential_depth = 0,  multiple_load = OFF,
ram_sequential = OFF,  clock_po = ON
possible credit =          50%
pulse generators =         OFF
RAM initialization =       uninitialized
RAM test mode   =          static_pass_thru
random atpg =              ON
random clocks =            none
random patterns =          1024
screen display =           ON
shadow checking =          ON
skew load =                OFF
split capture cycle =      OFF (optimal)
stability check =          ON
system mode =              atpg
TLA loop handling =        ON
trace report =             OFF
Z handling =               int=X  ext=X
Zhold behavior =
ON
```

Related Topics

add_iddq_exceptions

analyze_restrictions

delete_iddq_exceptions

report_gates

report_iddq_exceptions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_fault_type

set_gate_report

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_ijtag_instance_options

Context: dft, patterns

Mode: setup, analysis, insertion

Sets options on instances for which there is a matched ICL module.

Usage

set_ijtag_instance_options ***ijtag_instance_spec*** [-use_in_dft_specification {off | auto}] [-connect_data_signals_already_connected_to_ports {on | off | auto}]

Description

The set_ijtag_instance_options command specifies options on instances for which there is a matched ICL module.

Arguments

- ***ijtag_instance_spec***

  A required string that is a Tcl list of names of one or more IJTAG instances, or a collection containing one or many IJTAG instances as reported by the get_instances or get_ijtag_instances commands.

  The instance objects must have an ICL module matched to them.

  Tip

  If your ***ijtag_instance_spec*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

- -use_in_dft_specification {off | auto}

  An optional switch and value pair that controls the consideration of specific instances for creating the DftSpecfication during the create_dft_specification command. This command overrides the value of the tessent_use_in_dft_specification attribute from the ICL module description of the specified instance. This override only applies to the specified instance. When set to auto, the tool will consider the instance's scan interfaces and data ports for inclusion during create_dft_specification. When set to off, the tool will not consider the instance's scan interface and data ports for inclusion during create_dft_specification. See the description of the create_dft_specification command for more information.

- -connect_data_signals_already_connected_to_ports {on | off | auto}

  An optional switch and value pair that specifies how DataInPorts and DataOutPorts on the specified instances are handled when they are seen to be already connected to PI/PO ports of the current design. When set to on, the create_dft_specification command adds a TDR to control the DataInPorts and observe the DataOutPorts even if they are already connected to PI/PO ports of the current design. When set to off, no TDR is added to control the DataInPorts and observe the DataOutPorts when they are already connected to PI/PO ports

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

of the current design. The assumption is that they are already observable or controllable by the top level ports. When set to auto, the value is taken from the DefaultsSpecification(policy)/DftSpecification/IjtagNetwork/ connect_data_signals_already_connected_to_ports property. You set this option to on when you want to add observability and controllability through ijtag using an internal register even if they are already observable/controllable by ports of the current design.

Examples

The following example sets the -use_in_dft_specification option to off on all Ijtag instances below the instance blockA.

```
> set_ijtag_instance_options [get_ijtag_instances -below_instance blockA] \
                                -use_in_dft_specification off
```

Related Topics

get_ijtag_instance_option

get_memory_instances

report_ijtag_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_ijtag_retargeting_options

Context: all contexts

Mode: all modes (except -tck_ratio, -inject_cycles)

Enables you to configure different aspects of IJTAG retargeting.

Usage

```
set_ijtag_retargeting_options [-test_setup_network_end_state { keep | reset }]
    [-test_setup_ireset { off | first_icall | when_icl_present }]
    [-iapply_target_annotations { off | dense | full }]
    [-compare_constant_capture_values { on | off }]
    [-compare_constant_data_out_ports { on | off }]
    [-ignore_non_reset_end_state_implicit_compares { off | auto | on } ]
    [-constrain_update_en_only { auto | on } ]
    [-default_scan_out_strobe_point { before_rising_edge_of_tck |
        before_falling_edge_of_tck | auto } ]
    [-physical_block_iapply_partitioning { off | all | top_physical_blocks } ]
    [-interface_type { auto | ijtag | tpsp }]
    [-max_operations_per_iapply integer ]
    [-merge_irunloop_only { on | off }]
    [-network_end_state_warnings { on | off }]
    [-disable_pio_operations  { on | off }]
    [-ignore_iwrite_when_value_already_present { on | off }]
    [-allow_skip_rti { on | off }]
    [-generated_timeplate_name timeplate_name]
    [-tck_ratio { auto | 1 | 2 | 4 | 8 | 16 | … | 1024 }]
    [-tck_period tck_period]
    {[-inject_cycles inject_cycles_list ] |
      {[-extra_control_setup_hold_cycles extra_cycles ]
       [-extra_tms_setup_hold_cycles extra_cycles ]}}
    [-annotation_parameter_values parameters]
```

Description

The set_ijtag_retargeting_options command specifies configuration values controlling IJTAG retargeting.

Arguments

- -test_setup_network_end_state { keep | reset }

Note

This switch only applies to ijtag retargeting when an iCall is used in test_setup (see the set_test_setup_icall command for more details).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

An optional switch and literal pair that specifies to switch off the reset and preserve the state of the ICL network after the last test_setup iCall. The default is to do the reset. This reset only affects the configuration of the scan paths and the selection of the scan interfaces. The reset does not trigger any activity on ports of type ResetPort or TRSTPort.

If you have a test mode where you are applying the load_unload data through the IJTAG network and do not want to reset the state of the ICL scan network after the last test_setup iCall, use the "set_ijtag_retargeting_options -test_setup_network_end_state keep" command to turn off the reset of the ICL network for those designs.

- -test_setup_ireset { off | first_icall | when_icl_present }

  An optional switch and literal pair that specifies if and when the tool automatically adds an iReset to the beginning of the test setup procedure. Choose one of the following options:

  - off

    The tool does not automatically add an iReset to the test_setup procedure.

  - first_icall

    The tool automatically adds an iReset when it encounters the first iCall or iMerge in the test_setup procedure and no user-specified iReset precedes this iCall or iMerge.

  - when_icl_present

    The tool automatically adds an iReset if there is ICL for the current design. This is the default setting. It triggers the creation of an appropriate reset sequence at the ICL ports even if the test_setup procedure does not contain any iCall or iMerge statements.

- -iapply_target_annotations { off | dense | full }

  An optional switch and literal pair that specifies the style of the annotations that show the processed iWrite and iRead targets of an iApply. Choose one of the following options:

  off — The tool does not create annotations.

  dense — The tool does not abbreviate any bit-blasting of registers and ports, or binary values with many bits. This is the default.

  full — The tool bit-blasts registers and ports, one bit per line, with no abbreviations.

- -compare_constant_capture_values { on | off }

  An optional switch and literal pair that turns on or off the automatic expected value generation for the scan registers.

  If set to on, the patterns do not include any extra scan loads. Instead, when a bit happens to be scanned out in a given scan load and the tool knows the value of this bit from the CaptureSource description in ICL, it automatically infers a compare value for that bit at no extra cost. This feature is useful for diagnosing issues with the network and you should keep it on unless the ICL is incorrect.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the test_setup procedure contains PDL commands, the inferred compare values (when they exist) are part of the validation of the test_setup procedure (see E14), and they are part of the pattern files that contain the vectors for the test setup. If the tool processes the PDL in an open pattern set in the "patterns -ijtag" context, the inferred compare values (when they exist) are part of the generated pattern sets. Therefore, they are also part of the pattern files and the retargeted PDL files that the tool creates from those pattern sets when using the write_patterns command. If the inferred compare values are unwanted, then you must turn off the related option of the set_ijtag_retargeting_options command before the PDL retargeting happens and during the retargeting run. It is not possible to suppress the inferred comparison in existing pattern files or existing retargeted PDL files if the "-compare_constant_capture_values" or "-compare_constant_data_out_ports" switches were on during the PDL retargeting phase.

- -compare_constant_data_out_ports { on | off }

  An optional switch and literal pair that turns on or off the expected value generation for the primary DataOut ports. If set to on, the patterns do not include any extra vectors. The tool infers an automatic compare value on top-level DataOutPorts if its value is known from its described source in ICL. This feature is useful for confirming the correctness of the ICL but the tool does not set it to on by default to avoid increasing the number of used ports in a pattern file. For example, if you are exporting the pattern in SVF format, you typically would not want to have to compare output pins other than TDO.

  The usage of the -compare_constant_data_out_ports switch impacts the validation of the test_setup procedure, the pattern files and the retargeted PDL. For more information, see the description of the -compare_constant_capture_values switch above.

- -ignore_non_reset_end_state_implicit_compares { off | auto | on }

  An optional switch and literal pair that specifies the behavior of implicit compares for ScanRegisters with a non-constant CaptureSource.

  An implicit compare (sometimes called "implied iRead") is an observation on a ScanOutPort added to the patterns by the tool. The tool creates it if an ICL ScanRegister on the active path has a CaptureSource that refers to a constant value or to another (or the same) ScanRegister with supposedly known content.

  The PDL commands in the test_setup, ssn_setup, ssn_end and test_end procedures are retargeted only once, and the tool stores the internal representation of the resulting patterns (including the implicit compares) internally and reuses it for subsequent invocations of the write_patterns command. However, the content of some of the ScanRegisters at the end of test_end might be different from what it was at the beginning of test_setup, because the test_setup, ssn_setup, ssn_end or test_end procedures or the scan patterns themselves might have modified those values. Therefore, reusing the test_setup procedure in the next write_pattern command can result in miscompares, if the test_setup procedure contains implicit compares of scan registers capturing the parallel outputs of other scan registers (or capturing themselves), and the values on the CaptureSource have been modified in the meantime.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Constant CaptureSource specifications and CaptureSource specifications referring to ScanRegisters that participate in the scan path configuration are considered “safe”. They have the same values after each invocation of write_patterns due to the way the tool handles the reset of the scan path configurations.

The -ignore_non_reset_end_state_implicit_compares switch enables you to control the creation of the implicit compares. If this is set to “off”, the tool does not differentiate between “safe” and “unsafe” CaptureSources and creates implicit compares for as many CaptureSource specifications as possible. If it is set to “on”, the tool skips “unsafe” CaptureSource specifications during the creation of implicit compares. If it is set to “auto”, the tool suppresses the creation of implicit compares only if there are potentially active SSN scan hosts in the design. This is because the presence of SSN is the one major reason for IJTAG ScanRegisters changing their content between the test_setup and the test_end procedure. Designs without SSN do not experience the modification of the ScanRegister content unless this is explicitly enforced by test_setup iCalls or test_end iCalls.

- -constrain_update_en_only { auto | on }

  An optional switch and literal pair that specifies how the tool should protect the IJTAG network from becoming disturbed during scan test. If it is set to “auto”, the tool will constrain the Select ports to their off state unless there are potentially active SSN scan hosts with IJTAG streaming capability in the design, in which case the Update Enable ports will be constrained to their off state. If it is set to “on”, the tool will constrain the Update Enable ports to their off state.

- -default_scan_out_strobe_point { before_rising_edge_of_tck | before_falling_edge_of_tck | auto }]

  An optional switch and literal pair that specifies the default strobe point used by IJTAG pattern sets when the “open_pattern_set -scan_out_strobe_point” switch is not set. Refer to “open_pattern_set -scan_out_strobe_point” in the *Tessent Shell Reference Manual* for information on the strobe point.

- -physical_block_iapply_partitioning { off | all | top_physical_blocks }

  An optional switch and literal pair that controls if the IJTAG retargeting engine splits iApplys based on the targeted physical blocks. In certain complex designs it is very difficult to optimize large iApplys that target elements throughout the IJTAG network. This is especially true in “star networks” where it is not possible to place all iApply targets on the active scan path at the same time.

  In such cases, it can be beneficial to split a larger iApply into smaller sub-iApplys based on the physical blocks of the targeted elements. This may increase the number of scan loads but significantly reduces the retargeting effort, thus improving pattern generation performance. The overall shift length may be slightly larger, but typically not in any significant amount.

  When set to “all”, each iApply is split based on the physical block in which the target element is located. When set to “top_physical_blocks”, each iApply is split based on the target element’s topmost physical block. This reduces the number of partitions but might offer a lower performance benefit.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The -physical_block_iapply_partitioning option must be set to “off” when the DiagnosisOptions/extract_flop_data property is set to “on”.

Note

Since the physical block iApply partitioning splits iApplys into smaller elements, actions that might have occurred simultaneously or in direct succession without partitioning, can now be separated by multiple additional scan loads. While this is compliant with the IEEE 1687 (IJTAG) standard, it might result in unexpected behavior for specialized hardware. You cannot use the iApply partitioning in such circumstances.

You cannot use physical block iApply partitioning for LBIST diagnosis when targeting multiple cores at the same time.

- -interface_type { auto | ijtag | tpsp }

  An optional switch and literal pair that specifies the type of scan interfaces that the IJTAG engine may retarget operation to.

  auto — All active scan interfaces. This is the default.

  ijtag — Only active scan interfaces of type IJTAG or of type TAP.

  tpsp — Only active TAP host scan interfaces of TwoPinSerialPort controllers.

- -max_operations_per_iapply *integer*

  An optional switch and integer pair that specifies the upper limit of any scan loads and parallel Input or Output operations used to retarget one single iApply. Valid values are in the range from 4 to 1024. The default is 64. In very complex ICL networks, it might be necessary to increase this limit to provide more degrees of freedom to the retargeting engine. Usually, the tool only reaches the predefined limit if it is impossible to resolve the current iApply, for example, because of conflicting iWrite commands or a deadlock in the ICL network. Increasing this limit can result in very long run times of the iApply processing.

- -merge_irunloop_only { on | off }

  An optional switch and literal pair that specifies whether the tool enables the iApply statements of concurrent iCalls to be merged during the processing of an iMerge block. Merging iApply statements can be very complex and the retargeter may spend a lot of time on this task. The actual benefit (reduction of test time) is negligible in the case of tests in which the tool determines the test time mainly by iRunLoop statements. In such situations, you can set this switch to “on” to suppress the merging of iApply statements and to speed up the iMerge computation.

- -network_end_state_warnings { on | off }

  An optional switch and literal pair that specifies whether the IJTAG engine prints warnings during a network end-state reset. If set to on, the IJTAG engine prints a warning and

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

explanation for each scan register or primary input that is not set to its reset state. The default is off.

If you run the command "close_pattern_set -network_end_state reset" and close an IJTAG pattern set, then the tool responds by setting the state of the ICL network to the reset state. You can achieve the same behavior through similarly named properties in a patterns specification during process_patterns_specification.

During the reset procedure, some scan registers or primary inputs may not be set to their reset state because they were subject to a previous iWrite or other constraint. For details on the reset procedure, refer to the close_pattern_set command.

By default, the network end state is also reset at the end of the test_setup procedure. Scan registers like those controlling DFT signals or SSN control logic are not reset at the end of the test setup.

- -disable_pio_operations { on | off }

  An optional switch and literal pair that specifies whether the tool enables the IJTAG retargeter to access the IJTAG network through parallel interfaces (DataInPorts, DataOutPorts, AddressPorts, and so on) of the current design. If set to on, the retargeter can only use scan loads to process iApply commands, but it cannot toggle any values at the top module DataInPorts, AddressPorts and so on, or observe values at the top module DataOutPorts.

- -ignore_iwrite_when_value_already_present { on | off }

  Note

  The IEEE 1687 (IJTAG) standard generally requires the IJTAG retargeting engine to perform an actual write for each iWrite operation. Therefore, enabling this option might not be compatible with all IJTAG-compliant hardware.

  An optional switch and literal pair that enables the IJTAG retargeting engine to skip iWrite operations to ScanRegisters and DataRegisters that already have the required value.

  If set to on, the retargeting engine checks if the value requested by the iWrite is already present in the target of the operation. If the value is already present, the retargeting engine is not required to write the ScanRegister or to write the DataRegister that is the target of the iWrite operation.

- -allow_skip_rti { on | off }

  An optional switch and literal pair that specifies whether the Run-Test/Idle (RTI) state of a TAP controller can be skipped. The default is "on." Specify "off" before opening a pattern set to require the RTI state between scan loads. For more information, refer to "Skipping the Run-Test/Idle State" in the *Tessent IJTAG User's Manual*.

  Note

  You cannot change this setting when a pattern set is open.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -generated_timeplate_name *timeplate_name*

  An optional switch and string pair that specifies a base name for the timeplate that the tool uses when generating the default timeplate during system mode transition. The base name can only consist of letters, digits, and underscores, and cannot start with a digit. Each command using this switch must specify a unique base name.

- -inject_cycles *inject_cycles_list*

  An optional switch and string pair that enables the insertion of "dead" cycles (cycles without TCK activity) while the TAP controller is in certain states. The -inject_cycles switch is mutually exclusive with either of the -extra_control_setup_hold_cycles or -extra_tms_setup_hold_cycles switches.

  Note

  The use of -inject_cycles is deprecated and we recommend using the -extra_control_setup_hold_cycles and -extra_tms_setup_hold_cycles switches instead.

  You should only use the -inject_cycles switch in setup mode when the tool is in one of the following contexts:

  - patterns -scan
  - patterns -scan_retargeting

  Specifying the -inject_cycles switch in analysis mode will not affect the output of the test_setup and test_end procedures because Tessent Shell generates them during system mode transition.

  The inserted cycles increase timing margins for the enable ports of instruments. The tool can only add extra setup margins for the capture and shift enable ports. For the update enable ports, depending on the off-state of TCK, the tool can add extra setup margins (TCK off state is 1) or extra hold margins (TCK off state is 0). The following table provides a summary:

Table 8-10. Extra Timing Margin With -inject_cycles

ICL Instrument Port	TCK off-state 0	TCK off-state 1
Capture enable	Extra setup margin	Extra setup margin
Shift enable		
Update enable	Extra hold margin	

  The number of injected cycles is programmable and depends upon the TAP state on entry. The maximum number of injected cycles you can specify is 256. The ***inject_cycles_list*** can be a single number or a Tcl list with an even number of arguments, where each pair of arguments specifies a TAP state and the number of injected cycles for that TAP state.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can choose one or more of the following TAP states:

Table 8-11. Valid TAP States

DRSELECT	IRSELECT
DRCAPTURE	IRCAPTURE
DRSHIFT	IRSHIFT
DREXIT1	IREXIT1
DRPAUSE	IRPAUSE
DREXIT2	IREXIT2
DRUPDATE	IRUPDATE
IDLE	RESET

The names of the TAP states are consistent with SVF.

The cycles are injected immediately after the TAP state machine has entered the specified state, when the TCK is back to its off-state again. This changes the behavior of the TAP state machine such that the related control signals of a specified TAP state are active for a longer time than they would be usually. Note that the TRST control signal of the TAP is inactive during injected cycles in the RESET state.

The specification of the dead cycles refers to the duration of a TCK cycle, not to the duration of a tester cycle.

The -inject_cycles option only affects pattern files in which the tool can represent the dead cycles during TAP state transitions. The output of the “write_patterns -svf” and “write_patterns -pdl” commands is not changed.

The effect on the pattern files is the same as using an “iRunLoop -time” command. This avoids the TCK pulse. The tool inserts iRunLoops at the appropriate places in the unrolled version of the pattern sets. These loops are also visible by the pattern set introspection command, “get_pattern_set_data”.

- -tck_ratio { auto | 1 | 2 | 4 | 8 | 16 | … | 1024 }

  An optional switch and literal pair that specifies the TCK ratio for the use of PDL in test_setup and test_end procedures and for the PDL retargeting in the context patterns -ijtag. The default is “auto”.

  The -tck_ratio switch can only be used in setup mode when the tool is in one of the following contexts:

  - patterns -scan
  - patterns -scan_retargeting
  - patterns -scan_diagnosis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can use the -tck_ratio switch in any system mode in the remaining contexts (for example, patterns -ijtag).

In test_setup and test_end procedures, the tool either creates the waveforms of the TCK ports appropriately (if they have not been specified before) or checks them with respect to compatibility with the -tck_ratio specification.

In the patterns -ijtag context, the tool uses the TCK ratio specification of the command set_ijtag_retargeting_options as the default value for the tck_ratio switch of the open_pattern_set command. The usage of this value depends on the parameters that are specified when opening a new pattern set. The different combinations of parameters and their impact on the TCK ratio and TCK period are described in detail in the open_pattern_set command.

The value of tck_ratio might depend on the maximum tck clock frequency that can be applied to the circuit. See the "IJTAG Network Performance Optimization" section in the *Tessent IJTAG User's Manual* showing how to maximize the frequency of the IJTAG network test clock.

- -tck_period *tck_period*

  An optional switch and string pair that specifies the TCK period for the PDL retargeting in the patterns ijtag context. The default value is 100 ns. Based on the TCK period, TCK ratio, and tester period, the open_pattern_set command generates an appropriate timeplate. You do not have to specify the time unit. When you do not specify the time unit, the default is ns.

- -extra_control_setup_hold_cycles *extra_cycles*

  An optional switch and integer pair that specifies the number of *extra_cycles* before and after critical control signal transitions to stretch the TCK clock pulses. The maximum number of extra control setup and hold cycles is 256. The selective TCK stretching has the effect of relaxing the setup and hold timing of the IJTAG control signals coming from physical block ports or the TAP controller: capture_enable, shift_enable, update_enable, and select. For further information, refer to "Selective TCK Stretching" in the *Tessent IJTAG User's Manual.*

  You cannot specify this switch if you already have an IJTAG pattern set open with the open_pattern_set command.

  The write_patterns command uses the last value set by this option. Specifying this option multiple times before write_patterns has no effect.

  Note

  This switch supersedes the -inject_cycles switch. The -extra_control_setup_hold_cycles switch is mutually exclusive with the -inject_cycles switch.

- -extra_tms_setup_hold_cycles *extra_cycles*

  An optional switch and integer pair that specifies the number of *extra_cycles* before and after TMS transitions to stretch the TCK clock pulses. The maximum number of extra TMS

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

setup and hold cycles is 256. The selective TCK stretching has the effect of relaxing the setup and hold timing of the IJTAG TMS signal coming into the TAP controller. For further information, refer to "Selective TCK Stretching" in the *Tessent IJTAG User's Manual*.

You cannot specify this switch if you already have an IJTAG pattern set open with the open_pattern_set command.

The write_patterns command uses the last value set by this option. Specifying this option multiple times before write_patterns has no effect.

Note

This switch supersedes the -inject_cycles switch. The -extra_tms_setup_hold_cycles switch is mutually exclusive with the -inject_cycles switch.

- -annotation_parameter_values *parameters*

  An optional switch and list of values to configure the behavior of notes, annotations and comments in Tessent IJTAG. The list must consist of an even number of elements, keys and values in alternating order. The following table shows an overview of the various configuration possibilities, default values are underlined:

Table 8-12. Annotation Parameter Values

key	possible values	explanation
iapply_targets	off | <u>dense</u> | full	Annotations about iApply targets, the same as set_ijtag_retargeting_options -iapply_target_annotations
icl_network_verify_activate	<u>on</u> | off	Comments indicating that the following test vectors establish a scan path in the ICLNetworkVerify patterns
icl_network_verify_flush	<u>on</u> | off	Comments indicating that the following test vectors are flush patterns, that is, patterns that are loaded and unloaded without performing an "update" and a "capture" operation in between
network_end_state	<u>on</u> | off	Comment indicating that the following test vectors establish the specified network end state, see: close_pattern_set -network_end_state set_ijtag_retargeting_options -test_setup_network_end_state
pragma_annotation	<u>on</u> | off	The output of "tessent_pragma annotation" in the pattern files, that is, the annotation with the original iRead targets, per scan load

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-12. Annotation Parameter Values (cont.)

key	possible values	explanation
pragma_bit_annotation	on | off	The output of "tessent_pragma bit_annotation" in the pattern files, that is, the annotation with the original iRead targets, per cycle
pragma_bit_variable	on | off	The output of "tessent_pragma bit_variable" in the pattern files
pragma_bscan_annotation	on | off	Activates the creation of annotations to help map miscompares to port names in JTAG BoundaryScan tests.
pragma_clock	on | off	The output of "tessent_pragma clock" in the pattern files
pragma_expect_z	on | off	The output of "tessent_pragma expect_z" in SVF files
pragma_icl_checksum	on | off	The output of "tessent_pragma icl_checksum" in the pattern files
pragma_pattern_set	on | off	The output of "tessent_pragma pattern_set" in the pattern files
pragma_tester_period	on | off	The output of "tessent_pragma tester_period" in the pattern files
pragma_variable	on | off	The output of "tessent_pragma variable" in the pattern files
scan_load_unload_value	on | off	Annotations about the values of a scan load/ unload
scan_path_registers	on | off	Annotations about the scan registers of a scan path
scanin	on | off	Activates the creation of annotations on all shift cycles. These annotations indicate the register bit that is filled with the value that is applied to the scan input port during the current cycle. The annotations are of type "tessent_pragma bit_annotation", see Comments and Annotations in Tessent IJTAG in the *Tessent IJTAG User's Manual*. The setting can be changed at any place in the dofile or in the input PDL, and takes effect on the retargeting result of the following iApply and iMerge commands.
svf_command_number	on | off	A comment with the command number in SVF files

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-12. Annotation Parameter Values (cont.)

key	possible values	explanation
tap_states	on | off	Annotations about the TAP state transitions in pattern files in which each scan load is unrolled into a series of test vectors. The vectors that correspond to a TAP state transition are annotated with the destination TAP state
tessent_marker_design_instance	on | off	Enables TESSENT_MARKER design instance mapping.
user	on | off	iNote specifications

Examples

Example 1

The following example loads the ICL and turns off the reset of the ICL network after the last test_setup PDL command.

```
SETUP> read_icl ../data/chip.icl
SETUP> set_ijtag_retargeting_options -test_setup_network_end_state keep
```

Example 2

The following example uses the -inject_cycles option to extend the time in which the TAP rests in “capture-DR” state or “capture-IR” state for the duration of three TCK cycles.

```
set_ijtag_retargeting_options -inject_cycles {DRCAPTURE 3 IRCAPTURE 3}
```

The resulting pattern set contains three extra cycles without TCK pulse after each vector that brings the state machine to Capture-DR, and three extra cycles without TCK pulse after each vector that brings the state machine to Capture-IR.

To inject five dead cycles in all of the twelve valid TAP states, omit the state when you issue the -inject_cycles command.

```
set_ijtag_retargeting_options -inject_cycles 5
```

If you want to reset everything, you can do so as follows:

```
set_ijtag_retargeting_options -inject_cycles 0
```

Related Topics

get_ijtag_retargeting_options

reset_ijtag_retargeting_options

iApply

iRead

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iWrite

report_ijtag_retargeting_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_insertion_options

Context: unspecified, all contexts

Mode: all modes

Specifies insertion option values that affect the behavior of the insertion commands.

Usage

set_insertion_options [-auto_uniquify_edited_modules {on | off | when_needed }]
[-edited_module_prefix *edited_module_prefix*]
[-net_uniquification_suffix *net_uniquification_suffix*]
[-module_uniquification_suffix *module_uniquification_suffix*]
[-instance_uniquification_suffix *instance_uniquification_suffix*]
[-allow_assigns {on | off | auto}]
[-use_buffer_cell *library_cell_name*]
[-inserted_object_comment_label *label*]
[-treat_common_ancestor_as_unique {on | off}]
[-preserve_paths_for_unrolled_v2001_loops {on | off}]
[-vhdl_port_pin_net_type {bit | std_logic | std_ulogic}]
[-open_input_pins {tie0 | filler_nets}]
[-remove_inactive_sections_in_unrolled_generate_loops {on | off}]
[-trace_across_module_boundaries_to_find_functional_drivers {on | off}]
[-undriven_created_input_message_severity {ignore | note | warning | error}]
[-create_unique_net_names_in_module {on | off}]
[-gen_loop_net_size_threshold {unlimited | *int*}]
[-edited_file_prefix *prefix_name*]
[-exclude_unused_incdir_from_file_dict {on | off}]

Description

The set_insertion_options command specifies the values for the insertion options. The option settings affect the behavior of the insertion commands.

Arguments

- -auto_uniquify_edited_modules {on | off | when_needed }

  An optional switch and literal pair that enables Tessent Shell to uniquify any module that is edited, either unconditionally or only when needed. This only occurs if a module has more than one instance in the current design.

  When set to on, a module is uniquified on the first edit. For example if all instances are edited, the number of unique copies created equals the number of instances, in addition to the original.

  When set to when_needed, a copy of a module is created and uniquified for each parameterized view rather than for each instance. For example if all instances are edited and they are instantiated over two parameterized views, the number of unique copies created equals two, in addition to the original.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The current design is the design you specified using the set_current_design command.

During uniquification, the new module name is constructed by prefixing the original module name with the specified or default module_uniquification_suffix. The instantiation call of the module in its parent module is automatically changed to use the new module name. If the parent module is also non-unique, it is uniquified until a unique parent module is reached.

- -edited_module_prefix *edited_module_prefix*

  An optional switch and string pair that specifies a prefix to add to every edited module. A valid prefix contains numbers, letters, or underscores. The value you specify is also applied as an optional prefix for ICL module matching and automatically takes place in the tool flow when assigning related ICL modules to the new prefixed modules.

  Note

  The *edited_module_prefix* list you specify is automatically used by the "set_module_matching_options -prefix_pattern_list *prefix_pattern_list*" command.

  By default, the -edited_module_prefix switch puts an underscore character between the prefix and the original module name.

  For example, if the module name is "xyz" and the prefix is set to "abc", the resultant module name after editing is "abc_xyz", not "abcxyz".When you use the set_module_matching_options -prefix_pattern_list command and switch, the tool does not assume the module prefix underscore, you must explicitly specify the underscore.

  Note

  The tool does not modify the root name of the module. Use the rename_module command if you also want to modify the root module name.

  Specifying a prefix is advisable when a module is used in multiple designs that are modified by Tessent Shell independently. For example, you have a module called Counter that is used in CoreA and CoreB, and those two cores are processed in isolation, adding a prefix that is a unique string to the module called Counter when editing the module in context of each core prevents you from having different views of the same module with the same name if CoreA and CoreB are loaded together into a tool such as a simulator.

  The -edited_module_prefix defaults to a null string, which means that no module name prefixing is done by default.

- -net_uniquification_suffix *net_uniquification_suffix*

  An optional switch and string pair that defines the string that Tessent Shell uses to uniquify a net when the net that is trying to be created already exists in the design. The string must contain letters, numbers, or underscores and one and only one # symbol at the end identifying a counter that is auto-incremented starting from 1 until the net name does not

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

already exist. For example, specifying -net_uniquification_suffix of _TS# appends the string _TS# to the net where # is replaced by the first integer above 0, making the net name unique. If net A, B, and B_TS1 already exist in the design, Tessent Shell would use A_TS1 if it needed to create a new net with default name A and would use B_TS2 if it tried to create a new net with default name B.

The net_uniquification_suffix defaults to "_ts#". If defined, the create_connections -net_uniquification_suffix command takes precedence.

- -module_uniquification_suffix *module_uniquification_suffix*

  An optional switch and string pair that defines the string that Tessent Shell uses to uniquify a module. The string must contain letters, numbers, or underscores and one and only one # symbol at the end identifying a counter that is auto-incremented starting from 1 until the module name does not already exist. For example, specifying -module_uniquification_suffix of _TS# appends the string _TS# to the module name where # is replaced by the first integer above 0, making the module name unique. If module A, B, and B_TS1 already exist in the design, Tessent Shell would use A_TS1 if it needed to create a new module with default name A and would use B_TS2 if it tried to create a new module with default name B.

  The module_uniquification_suffix defaults to "_#".

- -instance_uniquification_suffix *instance_uniquification_suffix*

  An optional switch and string pair that defines the string that Tessent Shell uses to uniquify a leaf instance which uses the create_instance command. The string must contain letters, numbers, or underscores and one and only one # symbol at the end identifying a counter that is auto-incremented starting from 1 until the module name does not already exist. For example, specifying -instance_uniquification_suffix of _TS# appends the string _TS# to the instance name where # is replaced by the first integer above 0, making the leaf instance name unique. If leaf instance A, B, and B_TS1 already exist in a module, Tessent Shell would use A_TS1 if it needed to create a new instance with default name A and would use B_TS2 if it tried to create a new instance with default name B.

- -allow_assigns <u>on</u> | off | auto

  An optional switch and literal pair that specifies that assign statements are to be converted into Verilog primitive buffers when writing out the design. For example, creating a connection between nets that are also pins causes the creation of an assign statement.

  on — Assign statements appear as assign statements in the output netlist.

  off — Assign statements appear as buffer primitives unless you use the -use_buffer_cell option to specify a specific library cell.

  auto — If -allow_assign is set to “off” and a cell is deselected by -use_buffer_cell with an empty string value, assign statements appear as buffer primitives, and -allow_assign is set to “auto”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -use_buffer_cell *library_cell_name*

  An optional switch and string pair to define a library cell to use instead of buffer primitives or assign statements. You can clear the buffer cell usage by calling -use_buffer_cell again with an empty string.

- -inserted_object_comment_label *label* (-rtl context only)

  An optional switch and string pair that defines a label that the tool adds to comments appearing before and after each inserted object in the netlist. By default, the label is the null string "" which means no comments are inserted. For more information, refer to Example 2.

- -treat_common_ancestor_as_unique on | off

  An optional switch and literal pair that specifies whether the create_connection command treats the common ancestor module of a one-to-many connection as unique. By default, the tool does not lift the connections as if the common ancestor is unique (that is, instantiated only once). This results in the same one-to-many connections being made in all instances of the common ancestor module. This is the setting used when instantiating the instruments as part of the process_dft_specification command.

  When set to off, the connection is lifted above the common ancestor if the common ancestor is non-unique and the connection does not include all repeated instances of the destinations. This enables the remaining destinations to be connected to a different source. This functionality is only available when auto-uniquify is off.

  If defined, the create_connections -treat_common_ancestor_as_unique command takes precedence.

- -preserve_paths_for_unrolled_v2001_loops on | off

  An optional switch and literal pair that specifies whether or not to preserve SDC paths to objects found inside unrolled generate loops in Verilog 2001 design files. If you issue the uniquify_instances command on an instance that is inside a generate loop or if the set_insertion_options -auto_uniquify_edited_modules option is on, the generate loop is unrolled as shown below based on the fact that -preserve_paths_for_unrolled_v2001_loops is on or off.

  For a detailed discussion of the pros and cons of the two option values, refer to "Unrolling of Generate Loops Side Effects" on page 5102 in "HDL Limitations in the Tessent Shell Flow" on page 5081.

  When on, each loop count is replaced by an if generate block where the block ID is the original block id with the original loop count. This block ID is escaped but the SDC path remains the same as before because in SDC paths, the escape characters are stripped out. In this format, the genvar that existed in the generate loops is no longer present so any occurrence of the genvar in the original Verilog or VHDL code is replace by the literal value of the genvar.

  When you set it to off, the unrolled loops are all replaced with single count loops so that the genvar is preserved, enabling the Verilog or VHDL code that references the genvar to be preserved unaltered.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Original loop:

```
wire [1:0] in1, in2;
generate
genvar i;
for( i = 0; i < 2; i++ )
begin : joe
`ifdef TECHNOLOGY_32NM
     DUT i0 ( .A(in1[i]), .B(in2[i]), ... );
`else
     DUT1 i0 ( .A(in1[i]), .B(in2[i]), ... );
`endif
end for
end generate
```

Unrolled loop when -preserve_paths_for_unrolled_v2001_loops is on:

```
wire [1:0] in1, in2;
if(1) begin : \joe[0]
     DUT1 i0 ( .A(in1[0]), .b(in2[0]), ... );
end
if(1) begin : \joe[1]
     DUT1 i0 ( .A(in1[1]), .b(in2[1]), ... );
end
```

Unrolled loop when -preserve_paths_for_unrolled_v2001_loops is off:

```
wire [1:0] in1, in2;
genvar i;
for( i = 0; i <=0 ; i++ ) begin : \joe[0]
`ifdef TECHNOLOGY_32NM
     DUT i0 ( .A(in1[i]), .b(in2[i]), ... );
`else
     DUT1 i0 ( .A(in1[i]), .b(in2[i]), ... );
`endif
end for
for( i = 1; i <= 1; i++ ) begin : \joe[1]
`ifdef TECHNOLOGY_32NM
     DUT i0 ( .A(in1[i]), .b(in2[i]), ... );
`else
     DUT1 i0 ( .A(in1[i]), .b(in2[i]), ... );
`endif
end for
```

- -vhdl_port_pin_net_type bit | std_logic | std_ulogic

  An optional switch and literal pair that sets the type for new ports, pins, and nets that are created in VHDL. This option has no default. If the option is specified, then new ports, pins, and nets get the specified type with two exceptions:

  - If a new signal is connected to an output pin, then the type of the signal is the same as the type of the pin.
  - If a new signal is connected to another signal with a user-defined type, then the type of the new signal is the same as the type of the other signal.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the option is not specified, then the types of new ports, pins, and nets are determined as follows:

- If the port, pin, or net is to be connected to an existing VHDL signal, then the type of the new signal is governed by the type of the existing signal to which it is to be connected.
- If the port, pin, or net is not to be connected to an existing VHDL signal, then the type of the new signal is 'bit'.

- -open_input_pins tie0 | filler_nets

  An optional switch and literal pair that specifies how Tessent Shell handles unconnected input pins for inserted objects. When the option tie0 is specified, the unconnected inputs are tied low. When filler_nets is specified, the unconnected inputs are tied to a common filler net that is not driven.

- -remove_inactive_sections_in_unrolled_generate_loops {on | off}

  An optional switch and literal pair that removes inactive parts of loops if the loop contains a part that is inactive for certain values of the loop index.

  The switch has only an effect when a VHDL loop is unrolled. Use this option to avoid parsing errors with the inactive section that can happen when the index value is substituted for the unrolled loop.

- -trace_across_module_boundaries_to_find_functional_drivers {on | off}

  An optional switch and literal pair that specifies whether the tool traces across module boundaries to identify the driver for the has_functional_source pin attribute. The default is off.

  When set to on, the tool continues tracing across module boundaries until it finds a cell driver, an RTL driver, or an open or a tie value to determine the value of the attribute.

- -undriven_created_input_message_severity {ignore | note | warning | error}

  An optional switch and literal pair that specifies the severity level of Tessent Shell messages that the tool issues when creating undriven input pins/nets in the design. By default, the tool issues a warning for these input pins/nets. This switch enables you to change this to ignore, note, or error.

  During custom design editing, it is possible to create input pins on instances that remain undriven at the end of the insertion command. This happens when the modified modules are instantiated multiple times, and you failed to specify a connection for all instances of the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

modified module. In this case, Tessent Shell creates the following message when changing from the insertion mode to the setup mode:

```
//  Warning: The design contains <number> created inputs that are
//  left open.
//  You can find them before leaving the insertion mode with:
//    set created_ports [get_attributed_objects -attribute_names is_created
//      -object_types port]
//    set open_inputs [get_pins -of_ports $created_ports -filter
//      {!has_functional_source&&!tie_value}]
//   You can also use 'set_insertion_options -open_input_pins tie0' to automatically
//   tie them to 0.
//   The following pins have been identified:
//     <list of the first three pins>
```

You can use the -undriven_created_input_message_severity switch to omit the message or change its severity. Setting the severity to ignore omits the message completely. Setting the severity to error prevents the tool from exiting the insertion mode when the situation is detected. You can use the commands in the above message to identify those pins and take corrective actions. In some cases, you can leave input pins with no drivers if you know that connections to them are to be made at a later step.

You can also use the -auto_uniquify_edited_modules switch to auto-uniquify all edited modules. This avoids creating input pins with no drivers as there are to be no edited modules instantiated multiple times given the unification process.

- -create_unique_net_names_in_module {on | off}

  An optional switch and literal pair that directs the insertion commands to make the names of new nets created inside generate blocks unique throughout the enclosing module scope.

  Note

  You must use this option before setting the current design.

- -gen_loop_net_size_threshold {unlimited | *int*}

  An optional switch and value pair that specifies the maximum size of the net created when unrolling a “generate loop” block during design editing. When the net needs to be larger than the specified threshold, the tool unrolls the generate loop into separate “if generate” blocks to avoid having to create the large net. Use this switch only if you have a tool in your downstream flow that cannot handle large nets.

- -edited_file_prefix *prefix_name*

  An optional switch and string value that specifies a prefix that the tool uses for writing out design include files and any other edited file. This can be useful when the tool writes out identical edited files to different TSDBs and then later reads them back in parallel (for example, to run simulations of the top level). By default, the tool uses the design name as the prefix for edited files. You can reset the prefix setting by specifying an empty string.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -exclude_unused_incdir_from_file_dict {on | off}

  An optional switch and literal pair that specifies whether the design source dictionary references unused include directories when you run the write_design command. This dictionary contains information about the design files that were written out and can be used to load a design.

  Note

  For a description of the file dictionary format, see the description of the *<design_name>.design_source_dictionary* component in "dft_inserted_designs" on page 4820.

  on — Unused include directories are excluded from the design source dictionary.

  off — All include directories during the reading of design file commands are referenced, even if they are not used or empty. This is the default.

Examples

Example 1

The following example sets various insertion options that affect the behavior of the insertion commands.

```
SETUP> set_insertion_options -edited_module_prefix MyCoreName_ \
                             -module_unification_suffix _uniq#  -allow_assigns Off
```

Example 2

The following example sets the comment label for inserted objects to "myFlow".

```
SETUP> set_insertion_options -inserted_object_comment_label myFlow
```

A resulting netlist would then look like this:

```
module top(a,z, b);
   input a;
   output z;
   /* myFlow Ports Start */
   inout b;
   wire b;
   /* myFlow Ports End */
```

Example 3

The following example deactivates automatic commenting of inserted objects. This is the same as the default behavior if you do not specify the -inserted_object_comment_label argument at all.

```
SETUP> set_insertion_options -inserted_object_comment_label { }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

The following example uses the -auto_uniquify_edited_modules switch while editing module instances or generate loops safely:

set_insertion_options -auto_uniquify_edited_modules on

Editing a non-unique module instance triggers automatic uniquification of the instance before the module is edited. Editing a non-unique generate loop causes loop unrolling before the scope is edited.

Note

You do not need to use the parent_instance and leaf_hash_name attributes in this case.

Example 5

The following examples uses the -auto_uniquify_edited_modules switch while editing module instances or generate loops but requires further action:

set_insertion_options -auto_uniquify_edited_modules off
set_insertion_options -auto_uniquify_edited_modules when_needed

Editing a non-unique module instance edits all instances of the module. Editing a non-unique generate loop edits the loop for all loop indices.

Note

In these cases, you must use the parent_instance and leaf_hash_name attributes to avoid errors. Alternatively, you can use explicit uniquification with the uniquify_instances command.

Example 6

The following example shows how to modify the root module name before using the set_insertion_option to add a prefix.

```
set_current_design blka

# Rename root module name
set_system_mode insertion
rename_module cypher -new_name edmod_prefix_blka
write_design ... -replace
set_system_mode setup

set_insertion_options -edited_module_prefix edmod_prefix
```

Related Topics

get_insertion_option

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_insert_test_logic_options

Context: dft -scan, dft -test_points

Mode: setup, analysis

Sets new prefix or infix values to use during test logic insertion.

Usage

Context: dft -scan, dft -test_points

Mode: setup, analysis

set_insert_test_logic_options
[-inserted_object_naming [compact | standard]]
[-inserted_object_prefix *inserted_object_prefix*]
[-persistent_cell_prefix *persistent_cell_prefix*]
[-persistent_clock_cell_prefix *persistent_clock_cell_prefix*]
[-logic_type *logic_type* -logic_infix *logic_infix*]
[-reuse_modules_when_possible { on | off }]
[-force_flat_logic_insertion { on | off }]

Context: dft -scan -test_points

Mode: setup

set_insert_test_logic_options [-generate_third_party_script { *dc* | *genus* | off }]

(Plus all of the switches available in Context: dft -scan, dft -test_points; Mode: setup, analysis)

Mode: analysis

(Same syntax as Context: dft -scan, dft -test_points; Mode: setup, analysis)

Description

The set_insert_test_logic_options command specifies new prefix or infix values that the tool uses when it creates logic with the insert_test_logic command.

Arguments

- -inserted_object_naming [compact | standard]

  An optional switch and string pair that specifies the naming style of the instances and associated nets. The “standard” naming style includes the name of the logic operators and operands that are involved in the creation of new HDL instances for the names of the instances and associated nets. This is the default. With the “compact” naming style, these logic operators and operands are substituted by a single numerical id, making the names substantially shorter when complex logic gets implemented during insert_test_logic.

- -inserted_object_prefix *inserted_object_prefix*

  An optional switch and string pair that specifies the prefix of HDL objects created during “insert_test_logic”. The default is “ts_”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -persistent_cell_prefix *persistent_cell_prefix*

  An optional switch and string pair that specifies the prefix of persistent cell instances created during “insert_test_logic”. The default is “tessent_persistent_cell_”.

- -persistent_clock_cell_prefix *persistent_clock_cell_prefix*

  An optional switch and string pair that specifies the prefix of persistent clock cell instances created during “insert_test_logic”. The default is “tessent_persistent_cell_”.

- -logic_type *logic_type*

  An optional switch and literal pair that specifies the logic type for which the infix must be set. The possible values for logic_type are as follows:

  generic — Used for generic logic and has the default infix “logic_”.

  clock_gater_fix — Used for clock gater control and has the default infix “cgfix_”.

  clock_control_fix — Used for clock control and has the default infix “clkfix_”.

  dedicated_input_wrapper — Used for dedicated input wrapper cells and has the default infix “diw_”.

  dedicated_input_holding_wrapper — Used for dedicated input holding wrapper cells and has the default infix “dihw_”.

  dedicated_output_wrapper — Used for dedicated output wrapper cells and has the default infix “dow_”.

  dedicated_output_holding_wrapper — Used for dedicated output holding wrapper cells and has the default outfix “dohw_”.

  lockup — Used for scan chain retiming cells and has the default infix “lockup_”.

  pipeline — Used for scan chain pipeline cells has the default infix “pipeline_”.

  ram_fix — Used for ram control and has the default infix “ramfix_”.

  se_pipeline — Used for scan_enable pipeline logic and has the default infix “se_pipeline_”.

  All scan_enable pipelining logic inserted by the tool receives the new infix “se_pipeline_” by default. As a result, the test logic summary shows a new category highlighted below:

```
============================
Test Logic Insertion Summary:
============================
…

  Logical Data:
  -------------
            Added input dedicated wrapper logic count:          3
           Added output dedicated wrapper logic count:          1
             Added scan_enable pipelining logic count:          4
                    Added scan chain count (ext_mode):          6
                    Added scan chain count (int_mode):          6
```

  testpoint_control — Used for control testpoint and has the default infix “cp_”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

testpoint_observation — Used for observation testpoint and has the default infix “op_”.

testpoint_observation_scan — Used for observation scan observe points and has the default infix “osp_”.

tri_state_fix — Used for tristate and bidi control and has the default infix “trifix_”.

x_bounding — Used for X-bounding and has the default infix “xb_”.

Note

The -logic_type switch must be used in conjunction with the -logic_infix switch.

- -logic_infix *logic_infix*

  An optional switch and string pair that specifies the infix to use when implementing HDL objects of a specified logic type during “insert_test_logic”.

  You can include a special token %s inside the *logic_infix* string. The tool replaces %s with the value of the variable_logic_infix attribute for each scan element that the tool inserts. The tool also replaces the %s token with the attribute in the instance name of combinational logic associated with an inserted scan element.

  Control the value of the variable_logic_infix attribute by using the set_attribute_value command.

  Note

  You must use the -logic_infix switch in conjunction with the -logic_type switch.

- -reuse_modules_when_possible { on | off }

  An optional switch and literal pair that specifies whether new HDL modules are uniquified for each instance. This is the default, but if you set -reuse_modules_when_possible to “on”, only one new module definition gets created and is used/reused by all inserted instances of that type.

- -generate_third_party_script { *dc* | *genus* | off }

  An optional switch and string pair that instructs the tool to generate the insertion script for the specified target insertion tool:

  dc — Generates the script for Synopsys Design Compiler® (DC).

  genus — Generates the script for Cadence Genus™ Synthesis Solution.

  off — Does not generate the insertion script. This is the default.

  This switch instructs the tool to write the insertion script for the test points to insert into the design and identifies the OST observe point scan cells to stitch into independent scan chains.

  Using this switch blocks your ability to use the analyze_wrapper_cells and analyze_xbounding commands. Additionally, this switch disables insertion of test logic fixes requested by the set_test_logic, set_bidi_gating and set_tristate_gating commands.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When you use this switch, you must use the insert_test_logic command with either the -write_in_tsdb or -write_insertion_script option to write the associated third-party insertion script.

Note

This switch does not work in the test_points context. If you want to write out an insertion script in the test_points context, you must use the insert_test_logic command.

For more information, see "How to Create a Test Points and OST Scan Insertion Script for DC or Genus" in the *Tessent Scan and ATPG User's Manual*.

- -force_flat_logic_insertion { on | off }

  An optional switch and literal pair that prevents the tool from creating new design hierarchy during scan and test point insertion. Use this switch before performing any analysis.

  on — forces a flattened hierarchy.

  off — allows hierarchy. This is the default.

Examples

Example 1

The following example demonstrates how you can use the set_insert_test_logic_options command in the setup mode of Tessent shell:

```
SETUP> get_insert_test_logic_option -logic_type  dedicated_input_wrapper

diw_

SETUP> set_insert_test_logic_options -logic_type  dedicated_input_wrapper /
          -logic_infix inreg_

SETUP> get_insert_test_logic_option -logic_type  dedicated_input_wrapper

inreg_

SETUP> set_insert_test_logic_options -inserted_object_naming compact

SETUP> report_insert_test_logic_options
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  'insert_test_logic' Command Option          Current Value
//  ------------------------------------------  ------------------------
//  -inserted_object_naming                     compact
//  -inserted_object_prefix                     ts_
//  -logic_type clock_control_fix               clkfix_
//  -logic_type clock_gater_fix                 cgfix_
//  -logic_type dedicated_input_holding_wrapper dihw_
//  -logic_type dedicated_input_wrapper         inreg_
//  -logic_type dedicated_output_wrapper        dow_
//  -logic_type generic                         logic_
//  -logic_type lockup                          lockup_
//  -logic_type pipeline                        pipeline_
//  -logic_type ram_fix                         ramfix_
//  -logic_type testpoint_control               cp_
//  -logic_type testpoint_observation           op_
//  -logic_type tri_state_fix                   trifix_
//  -logic_type x_bounding                      xb_
//  -persistent_cell_prefix                     tessent_persistent_cell_
//  -persistent_clock_cell_prefix               tessent_persistent_cell_
//  -reuse_modules_when_possible                off
```

Example 2

The following example script directs the tool to embed the corresponding port name and cluster name in the instance name of each dedicated wrapper cell:

```
set_insert_test_logic_options -logic_type \
   dedicated_input_holding_wrapper -logic_infix {dihw_%s_}
set_insert_test_logic_options -logic_type \
   dedicated_output_holding_wrapper -logic_infix {dohw_%s_}

analyze_wrapper_cells

foreach_in_collection virtualScanElement \
           [get_scan_elements -filter exists==false] {
  set port [get_single_name [get_port -of_wrapper_scan_element \
           $virtualScanElement]]
  set cluster [get_attribute_value -name cluster_name $virtualScanElement]

  # Some code might be needed here to remove escaping from the
  # port name (the attribute does not accept special characters).

  set_attribute_value $virtualScanElement -name variable_logic_infix \
           -value "${port}_${cluster}"
}
```

Afterwards, “ts_1_dihw_in_1_clusterA_494smodp1_i” is representative of an instance name containing the specified *logic_infix* for a dedicated_input_holding_wrapper.

Related Topics

get_insert_test_logic_option

report_insert_test_logic_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_internal_fault

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_diagnosis

Mode: setup

Specifies whether the tool reports faults within library models.

Usage

set_internal_fault **OFf** | **ON**

Description

The set_internal_fault command specifies whether the tool reports faults on internal nets of library models (internal faulting) or only on the library model boundary. The default upon invocation of the tool is to report faults only on the library model boundary.

Note

This command is not valid for the bridge logic fault model (specified with the "set_fault_type Bridge -Static" command).

Arguments

- **OFf**

  A required literal that specifies that the tool reports faults only on the boundary of the library models. This is the default.

- **ON**

  A required literal that specifies that the tool reports faults on the internal nets of library models.

Examples

The following example performs ATPG using a fault list that includes internal nets within the library cells. These internal nets are shown as fault sites when the report_faults command is executed.

```
set_context patterns -scan
set_internal_fault on
set_system_mode analysis
create_patterns
report_faults -all
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_internal_name

Context: all contexts

Mode: setup

Specifies whether to delete or keep pin names of library internal pins containing no-fault attributes.

Usage

set_internal_name **OFf** | **ON**

Description

The set_internal_name command specifies whether to keep internal library pins with no-fault attributes. Normally, you should delete these names for memory and performance reasons. The default operation (OFF) upon invocation is to delete these names.

Arguments

- **OFf**

  A literal that deletes the lowest level pin names if they have the nofault attribute. This is the default.

- **ON**

  A literal that keeps the lowest level pin names, even if they have the nofault attribute.

Examples

The following example deletes pin names with the nofault attribute.

```
set_internal_name off
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_io_mask

Context: dft -edt. patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Modifies the behavior of bidirectional pins so that their expected values are always X during test cycles in which the primary input portion of the bidirectional pin is being forced.

Usage

set_io_mask {OFf | ON}

Description

Typically, when the tool forces stimulus on an bidirectional pin, it expects to measure the same value on the corresponding primary output pin. This command lets you modify this behavior in cases where this is undesirable.

Arguments

- OFf

  An optional literal that disables the ability to mask the bidirectional pin output value. This is the default.

- ON

  An optional literal that enables the ability to mask the bidirectional pin output value.

Examples

The following example causes the tool to mask the bidirectional pin output value.

```
set_io_mask on
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_latch_handling

Context: dft -scan

Mode: setup

Specifies whether the tool considers non-transparent latches for scan insertion while test logic is turned on.

Usage

set_latch_handling **None** | **Scan**

Description

The set_latch_handling command specifies whether the tool can consider non-transparent latches as candidates for scan insertion. If you determine that the tool is to consider non-transparent latches as candidates for scan insertion, you must turn on the appropriate set_test_logic command settings before the tool performs the rules checking process.

If you use the set_drc_handling command to set the D6 rule to error or warning, D6 checks the non-scannable latches for transparency. By default, if they are not transparent and you turned test logic insertion on, the tool does not consider the non-transparent latches for scan insertion. However, if you use the Scan argument with the set_latch_handling command, the tool considers the non-transparent latches for scan insertion.

If you use the set_drc_handling command to ignore the D6 rule, then rules checking does not check the non-scannable latches for transparency and the tool automatically considers all non-scannable latches, whose test logic you turned on, for scan insertion.

Arguments

- **None**

  A literal that specifies to give no consideration to non-transparent latches for scan insertion. This is the default upon invocation of the tool.

- **Scan**

  A literal that specifies to consider non-transparent latches for scan insertion when test logic is turned on.

Related Topics

set_drc_handling

set_test_logic

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_layout_core_instance

Context: patterns -scan_diagnosis

Mode: setup, analysis

For hierarchial layout-aware diagnosis, sets the current layout core instance for inter-scan cell logic reporting.

Usage

set_layout_core_instance ***instance_path***

Description

The set_layout_core_instance command specifies the current layout core instance for inter-scan cell logic reporting with the report_scan_polygons and report_scan_path commands.

You must use this command when you perform inter-scan cell logic reporting in the context of hierarchical layout-aware diagnosis. The command is required because inter-scan cell logic reporting occurs outside the processing of a diagnosis report. The set_layout_core_instance command has no effect on diagnose_failures because the tool automatically sets the layout core instance based on the core_instance keyword in the tracking_info section of the failure file when processing a diagnosis report.

See "Diagnosis for Hierarchical Design" and "Inter-Scan Cell Polygon Reporting for Chain Diagnosis" in the *Tessent Diagnosis User's Guide* for more information.

Arguments

- ***instance_path***

  A required string that specifies the path to a layout core instance.

Related Topics

get_layout_core_instance

report_scan_polygons

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_lbist_controller_options

Context: dft -logic_bist, patterns -scan, patterns -scan_diagnosis

Mode: setup

Specifies global options to configure the LogicBIST controller.

Usage

IP Creation Usage

set_lbist_controller_options [off | on] **-max_shift_cycles** {***integer1*** [*integer2*]}
-max_capture_cycles {***integer1*** [*integer2*]}
-max_pattern_count {***integer1*** [*integer2*]}
[-max_warmup_pattern_count {***integer1*** [*integer2*]}]
[-max_chain_test_pattern_count {***integer1*** [*integer2*]}]
[-max_async_set_reset_pattern_count {***integer1*** [*integer2*]}]
[-controller_chain_mode {off | on} [-controller_chain_clock {tck | edt_clock}]
[-segment_per_instrument {off | on}]
-capture_procedures {***procname*** [*integer%*] | ***count*** | ***integer%***}…
[-shift_counter_resolution {bit | byte}]
[-pre_post_shift_dead_cycles *integer*]
[-tessent_occ {auto | on}] [-burn_in {off | on}]
[-single_chain_mode_skip_edt_blocks {off | on}]

Fault Simulation Usage

set_lbist_controller_options [off | on] **-capture_procedures** {***procname integer%***}…
[-warmup_pattern_count *integer*]
[-initialize_dft_signals_in_child_physical_blocks {on | off}]

Description

This command is used with the hybrid TK/LBIST flow. Refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- off | on

  An optional literal that specifies enabling or disabling LogicBIST controller configuration. The default is on.

- **-max_shift_cycles** {***integer1*** [*integer2*]}

  A required switch and integer pair or triplet that specifies the maximum possible shift length the tool uses to synthesize the LogicBIST shift length counter.

  An optional second argument, *integer2*, can be used to specify the value to be loaded in the shift length counter for the hardware default mode of operation. This value should be less than or equal to the ***integer1***, the maximum shift cycles.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If you do not specify a value, then the maximum shift cycles value is used for the hardware default mode.

- **-max_capture_cycles** {***integer1*** [*integer2*]}

  A required switch and integer pair or triplet that specifies the maximum possible capture width the tool uses to synthesize the LogicBIST capture width counter.

  An optional second argument, *integer2*, can be used to specify the value to be loaded in the capture width counter for the hardware default mode of operation. This value should be less than or equal to the ***integer1***, the maximum capture width.

  If you do not specify a value, then the maximum capture width is used for the hardware default mode.

- **-max_pattern_count** {***integer1*** [*integer2*]}

  A required switch and integer pair or triplet that specifies the maximum number of LogicBIST patterns that is applied when the tool synthesizes the LogicBIST pattern counter.

  An optional second argument, *integer2*, can be used to specify the value to be loaded in the pattern counter for the hardware default mode of operation. This value should be less than or equal to the ***integer1***, the maximum pattern count.

  If you do not specify a value, then the maximum pattern count is used for the hardware default mode.

- -max_warmup_pattern_count {***integer1*** [*integer2*]}

  An optional switch and integer pair that specifies the maximum number of Logic BIST warm-up patterns that the tool can apply. The default is 255. *integer1* gets rounded up to a value of one less than the next power of 2.

  You can use an optional second argument, *integer2*, to specify the number of warm-up patterns to be applied for the hardware default mode of operation. This value must be less than or equal to *integer1*, the maximum pattern count. If you do not specify a value, then no warm-up patterns are applied for the hardware default mode.

  If you do not want the tool to insert the warm-up pattern count register in the logic BIST controller, set *integer1* to 0. Refer to "Warm-Up Patterns" in the *Hybrid TK/LBIST Flow User's Manual* for details.

- -max_chain_test_pattern_count {***integer1*** [*integer2*]}

  An optional switch and integer pair that specifies the maximum limit and a hardware default value for the number of chain test patterns to apply. The *integer1* value specifies the maximum number of patterns, and the *integer2* value specifies a hard-coded value to load into an LBIST controller register for use when running in hardware default mode. For more information about using this switch, see the equivalent functionality documented in the wrapper "ChainTestPatternCount" on page 4229.

- -max_async_set_reset_pattern_count {***integer1*** [*integer2*]}

  An optional switch and integer pair or triplet that specifies the maximum possible patterns to test the asynchronous set/reset signals of scannable flip-flops.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can use an optional second argument, *integer2*, to specify the number of asynchronous set/reset patterns to be applied for the hardware default mode of operation. This value must be less than or equal to *integer1*, the maximum pattern count. If you do not specify a value, then the tool applies no asynchronous set/reset patterns for the hardware default mode.

- -controller_chain_mode {off | on}

  An optional switch and literal pair that specifies whether to enable or disable the Controller Chain Mode (CCM) in the hybrid TK/LBIST dofile flow. When enabled, the tool creates RTL scan segments so that it can calculate test coverage for the LBIST controller and EDT/LBIST blocks. The default is off.

  See "Controller Chain Mode" in the *Hybrid TK/LBIST Flow User's Manual* for details.

- -controller_chain_clock {tck | edt_clock}

  An optional switch and literal pair that specifies whether the tool uses the tck or edt_clock clock during Controller Chain Mode. The default is edt_clock.

- -segment_per_instrument {off | on}

  An optional switch and literal pair that specifies whether, during IP insertion, the tool automatically connects controller scan chain segments within the IP blocks into one scan chain, or leave them as segments. The default is on (leave as segments).

  See "Dofile Flow Usage Details and Architecture for CCM" in the *Hybrid TK/LBIST Flow User's Manual* for details.

  Caution

  In the default mode (on), you are responsible for building the scan chains.

- **-capture_procedures** {***procname*** [*integer%*] | ***count*** | ***integer%***}

  A required switch and repeatable string pair that specifies the Named Capture Procedure (NCP) name for which the NCP should be applied.

  If a percentage is specified, all the NCPs are repeated according to the specified percentage once every 256 patterns. The sum of all the specified percentages should add up to 100.

  You can specify the percentage in addition to the name of the NCP or the percentage on its own without naming the NCP.

  Because the integer percentage may not be known at IP creation time, the integer percentage can be omitted and specified during fault simulation. At that time, you can program the percentage activity for each NCP without having to regenerate the IP. A percentage value of 0 means that the NCP is not active for the pattern set.

  The mutually-exclusive ***count*** option generates the LBIST controller that can support up to the "count" NCPs.

- -shift_counter_resolution {bit | byte}

  An optional switch and literal pair that specifies whether the shift counter considers shift lengths that operate in multiples of 8 or 1. When you specify "byte", the tool rounds up the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

length of the longest scan chain to the next multiple of 8. For example, if the longest scan chain is 67 flops long, then the tool applies 72 shifts.

By default, the tool applies the number of shifts equal to the length of the longest scan chain.

Note

If the shift length specified during IP generation differs from that used during fault simulation, you may be required to use the set_number_shifts command during fault simulation when creating HW default patterns. For example, suppose you have 29 scan chains for fault simulation, but for IP generation you specified a HW default max shift of 38. In this case you would specify the set_number_shifts command to create patterns for HW default mode.

- -pre_post_shift_dead_cycles *integer*

  An optional switch and integer pair that specifies how many shift pause and capture pause dead cycles, at the shift clock period, are applied before and after the capture clocks are pulsed. The specified integer can be between 2 and 8, with the default being 8. For example, specifying 5 means that the shift pause consists of 5 dead cycles and the capture pause consists of 5 dead cycles.

  Reducing the number of pause cycles reduces the test time, but it may also make it harder to achieve timing closure of the test logic. To ensure that the shift and capture signals are staggered an integer value of 1 is not supported.

- -tessent_occ {auto | on | off}

  An optional switch and literal that generates a Tessent OCC-compatible LBIST controller for skeleton designs that do not contain Tessent OCC instances. You specify this switch during pre-synthesis IP creation. Choose one of the following options:

  auto — Infers the Tessent OCC instruments from the TCD and the ICL. This is the default.

  on — Enables generation of Tessent OCC compatible LBIST controller when OCC core instances are not added.

- -burn_in {off | on}

  An optional switch and literal pair that specifies whether to enable or disable support for wafer-level burn-in tests. When enabled, the tool creates additional RTL to enable LogicBIST to run for longer periods of time than supported by the pattern counter hardware. The default is off.

- -single_chain_mode_skip_edt_blocks {off | on}

  An optional switch and literal pair that specifies whether to insert a SIB in each EDT block that enables optionally skipping all scan chains in a given EDT block during single chain unload. The default is on. For details, refer to "Single Chain Mode Diagnosis" in the *Hybrid TK/LBIST Flow User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -warmup_pattern_count *integer*

  An optional switch and integer pair that specifies the number of Logic BIST warm-up patterns to use during fault simulation. If you specify this value, the tool uses the hardware default PRPG seed and low-power mask shift register values as the starting point of the warm-up patterns.

  Note

  A Tessent Core Description (TCD) for the EDT IP creation is required when using this option.

- -initialize_dft_signals_in_child_physical_blocks {on | off}

  An optional switch and literal pair that specifies whether to automatically initialize LBIST-related DFT signal settings in child physical blocks. The default is on. For details, refer to "Initial Static DFT Signal Values" in the *Hybrid TK/LBIST Flow User's Manual*.

Examples

Example 1

The following example configures the LogicBIST controller for a maximum scan chain length of 1000 and a maximum capture width of 8 cycles. The maximum number of patterns is set to 250,000 and the number of patterns run in the hardware default mode is set to 5,000. There are 4 NCPs defined for LogicBIST test such that each of them is active for 25 percent of the patterns. A maximum of 127 warm-up patterns can be used, with 16 warm-up patterns run in the hardware default mode.

```
set_lbist_controller_options -max_shift_cycles 1000 -max_capture_cycles 8
set_lbist_controller_options -max_pattern_count {250000 5000}
set_lbist_controller_options -max_warmup_pattern_count {128 16}
set_lbist_controller_options -capture_procedures \
        {clkseq1 25 clkseq2 25 clkseq3 25 clkseq4 25}
```

Example 2

The following example is the same as the previous example except that the NCP percentage integer values have not been specified with the commands issued for IP creation. In this case, the NCP percentage integer values must be set by issuing the command with the desired values for fault simulation.

```
set_lbist_controller_options -max_shift_cycles 1000 -max_capture_cycles 8
set_lbist_controller_options -max_pattern_count {250000 5000}
set_lbist_controller_options -capture_procedures {clkseq1 clkseq2 clkseq3 clkseq4}
```

Related Topics

report_lbist_configuration

set_clock_controller_pins

set_edt_options

set_lbist_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_lbist_instances

Context: dft -logic_bist

Mode: setup

Specifies the instance in which the LogicBIST controller or single chain mode logic is placed.

Usage

set_lbist_instances [-controller_location *instance_path_name*]
[-single_chain_mode_logic_location *instance_path_name*]

Description

This command is used with the hybrid TK/LBIST flow. Refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

By default, the tool places the LogicBIST controller and single chain mode logic in the top level of the design. Using this command, you can specify a different location for the LogicBIST controller or single chain mode logic.

For example, you can use this command to place the LogicBIST controller or single chain mode logic instance in a specific test logic block or in an always-ON block on a design with multiple power domains.

Arguments

- -controller_location *instance_path_name*

  An optional switch and string pair that specifies the hierarchical instance pathname where the tool instantiates the LogicBIST IP.

- -single_chain_mode_logic_location *instance_path_name*

  An optional switch and string pair that specifies the hierarchical instance pathname where the tool instantiates the single chain mode logic.

Examples

The following example instantiates the LogicBIST controller in an always ON power domain instance */pwr_always/dft_logic*:

```
set_lbist_instances -controller_location /pwr_always/dft_logic
```

The following example instantiates the LogicBIST controller and single chain mode logic in a dedicated test logic block instance */dft_test_logic_i*:

```
set_lbist_instances -controller_location /dft_test_logic_i \
                    -single_chain_mode_logic_location /dft_test_logic_i
```

Related Topics

set_edt_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_clock_controller_pins

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_lbist_pins

Context: dft -logic_bist

Mode: setup

Specifies the connection information for LogicBIST controller pins and enables renaming LogicBIST pins.

Usage

Specify LogicBIST Controller Pin Connections

set_lbist_pins
scan_en {{***top_pin_name*** | **-**} [*internal_pin_name* | **-**]}… |
clock {***top_pin_name*** | **-**} [*internal_pin_name*] |
tck {*top_pin_name* | **-**} [*internal_pin_name*] |
setup_shift_scan_in {*top_pin_name* | **-**} [*internal_pin_name*] |
setup_shift_scan_out {*top_pin_name* | **-**} [*internal_pin_name*] |
tap_instruction_decode {*top_pin_name* | **-**} [*internal_pin_name*] |
test_logic_reset {*top_pin_name* | **-** [*internal_pin_name*]} [-active high | low] |
capture_dr {*top_pin_name* | **-**} [*internal_pin_name*] |
shift_dr {*top_pin_name* | **-**} [*internal_pin_name*] |
update_dr {*top_pin_name* | **-**} [*internal_pin_name*] |
controller_chain_en {{*top_pin_name* | -} [*internal_pin_name* | -]} |
controller_chain_scan_in {{*top_pin_name* | -} [*internal_pin_name* | -]} |
controller_chain_scan_out {{*top_pin_name* | -} [*internal_pin_name* | -]} |
wrapper_chain_input_scan_en {{*top_pin_name* | -} [*internal_pin_name* | -]}…

Rename LogicBIST Pins

set_lbist_pins [***pin_function port_name*** **-on_controller_module**] ...

Description

The set_lbist_pins command specifies connection information for LogicBIST controller pins. You can also use this command to rename pins at the LogicBIST module level in the RTL.

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information. In the flow, pin connections for the enable signals—xbounding_en, control_point_en, observe_point_en, mcp_bounding_en, and test_en—are required; specify them with the add_dft_signals command.

Specifying LogicBIST Pin Connections

When specifying LogicBIST pin connections, the pins are connected to the single top-level LogicBIST controller. This is in a global context and not specific to the current EDT block. The first argument is a literal string that indicates the pin being described in the command followed by one or more pin names.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If you issue this command multiple times with the same pin type, then the last pin or pins you specify are used. To specify multiple pin names corresponding to a pin type, specify all the names in a single issuance of the command.

If the signal to be connected to the LogicBIST controller is internally generated and does not have a corresponding top-level pin, use a "-" literal in place of the *top_pin_name* argument.

Renaming Pins

When renaming pins, the -on_controller_module switch accepts the following pin functions. The default pin names are shown. By default, the done pin is not created inside the LogicBIST controller. When this pin is specified, the port is created on the RTL.

Note

Pin functions shown with asterisks (*) are those for which you can also specify the pin connections.

Table 8-13. Pin Functions That Can Be Renamed

Pin Function	Default Name	Pin Function	Default Name
async_set_reset_en	lbist_async_set_reset_en	controller_chain_scan_out *	ccm_scan_out
clock *	shift_clock_src	controller_chain_scan_en	ccm_scan_en
tck *	ijtag_tck	test_clock	test_clock
setup_shift_scan_in *	ijtag_si	edt_update	edt_update
setup_shift_scan_out *	ijtag_so	clock_out	edt_lbist_clock
tap_instruction_decode *	ijtag_sel	reset_out	lbist_reset
test_logic_reset *	ijtag_reset	enable_out	lbist_en
capture_dr *	ijtag_ce	done	*no default value*
shift_dr *	ijtag_se	prpg_en	lbist_prpg_en
update_dr *	ijtag_ue	misr_en	misr_accumulate_en
scan_en *	scan_en_in scan_en_out	ijtag_select_out	edt_sib_en
controller_chain_en *	ccm_en	xbounding_en	x_bounding_en

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-13. Pin Functions That Can Be Renamed (cont.)

Pin Function	Default Name	Pin Function	Default Name
controller_chain_scan_in *	ccm_scan_in	observe_point_en	observe_point_en
control_point_en	control_point_en	test_en	test_en_in test_en_out
mcp_bounding_en	mcp_bounding_en		

Arguments

Specify LogicBIST Controller Pin Connections

- ***top_pin_name*** | **-**

  A required string or literal that specifies the pin name to be connected to the LogicBIST controller. Choose from the following:

  top_pin_name — A string that specifies the name of the pin to assign to the specific signal.

  - — A literal used for internally-generated pins that do not have a corresponding top-level pin.

- *internal_pin_name*

  An optional string that specifies an internal pin name.

- **scan_en** {{***top_pin_name*** | **-**} [*internal_pin_name* | **-**]} ...

  A required literal and string pair that specifies the scan enable pin and optionally the internal pin corresponding to the pad output. Multiple values can be specified if the design has multiple scan enables.

- **clock** {***top_pin_name*** | **-**} [*internal_pin_name*]

  A required literal and string pair that specifies the LogicBIST clock pin. When a top-level tester controllable clock is available, the same pin can be used for both EDT and LogicBIST clocks. However, when the EDT clock is internally generated, specify a different asynchronous free-running clock as the LogicBIST clock. To indicate that you are not using clock as the LogicBIST clock and to not create a clock pin, specify a dash for the internal_pin_name, as follows:

  ```
  set_lbist_pins clock {- -}
  ```

  In this case, LogicBIST uses the slower test_clock and TCK clocks.

- tck {*top_pin_name* | **-**} [*internal_pin_name*]

  An optional literal and string pair that specifies the JTAG TCK input and optionally the internal pin corresponding to the pad output.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- setup_shift_scan_in {*top_pin_name* | **-**} [*internal_pin_name*]

  An optional literal and string pair that specifies the connection to the seeding register input. In a JTAG based design, this would typically be connected to the TDI pin.

- setup_shift_scan_out {*top_pin_name* | **-**} [*internal_pin_name*]

  An optional literal and string pair that specifies the connection to the seeding register output. In a JTAG based design, this would typically be connected to the TDO pin through the DR-mux.

- tap_instruction_decode {*top_pin_name* | **-**} [*internal_pin_name*]

  An optional literal and string pair that specifies the connection to the instruction decoder output that indicates when the TAP instruction that targets LogicBIST is loaded in the JTAG instruction register.

- test_logic_reset {*top_pin_name* | **-** [*internal_pin_name*]} [-active high | low]

  An optional literal and string pair that specifies the connection to the test_logic_reset output of the TAP controller that indicates when the TAP is in test-logic-reset state. The default active reset state is low. Use the optional -active switch to change the active reset state to high.

- capture_dr {*top_pin_name* | **-**} [*internal_pin_name*]

  An optional literal and string pair that specifies the connection to the capture_dr output of the TAP controller that indicates when the TAP is in the capture-DR state.

- shift_dr {*top_pin_name* | **-**} [*internal_pin_name*]

  An optional literal and string pair that specifies the connection to the shift_dr output of the TAP controller that indicates when the TAP is in the shift-DR state.

- update_dr {*top_pin_name* | **-**} [*internal_pin_name*]

  An optional literal and string pair that specifies the connection to the update_dr output of the TAP controller that indicates when the TAP is in the update-DR state.

- controller_chain_en {{*top_pin_name* | -} [*internal_pin_name* | -]}

  An optional literal and string pair that specifies the controller chain mode enable pin and optionally the internal pin corresponding to the pad output.

  Note

  The controller chain connections for the enable, scan in, and scan out signals cannot be specified when you have enabled the set_lbist_controller_options -segment_per_instrument option. The tool issues an error.

- controller_chain_scan_in {{*top_pin_name* | -} [*internal_pin_name* | -]}

  An optional literal and string pair that specifies the controller chain mode scan input pin and optionally the internal pin corresponding to the pad output.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- controller_chain_scan_out {{*top_pin_name* | -} [*internal_pin_name* | -]}

  An optional literal and string pair that specifies the controller chain mode scan output pin and optionally the internal pin corresponding to the pad output.

- wrapper_chain_input_scan_en {{*top_pin_name* | -} [*internal_pin_name* | -]}

  An optional literal and string pair that specifies the scan enable pin of the input wrapper chains and optionally the internal pin corresponding to the pad output. Multiple values can be specified if the design has multiple input scan enables for the wrapper chain. The wrapper chain input scan enable signals are held at constant 1 (specifically, shift) during LogicBIST test. For ATPG, they remain controllable from specified top or internal pins.

Rename LogicBIST Pins

- ***pin_function***

  A required string for pin renaming that specifies the pin function whose default name you want to change. Refer to Table 8-13.

- ***port_name***

  A required string for pin renaming that specifies the new name for the specified pin function.

- **-on_controller_module**

  A required switch for pin renaming that specifies to rename the specified pin on the LogicBIST module in the LogicBIST RTL. Pin renaming only applies to the specified pin at the LogicBIST module level.

Examples

Example 1

This example specifies the connections between design pins and LogicBIST controller pins, including the TAP interface. A dash is used as the first argument for the TAP generated control signals, since there is no corresponding top-level pin for these signals.

```
set_lbist_pins clock refclk
set_lbist_pins scan_en scan_en
set_lbist_pins setup_shift_scan_in tdi
set_lbist_pins setup_shift_scan_out { tdo jtag/scanCfgReg_so }
set_lbist_pins shift_dr { - jtag/shift_dr }
set_lbist_pins capture_dr { - jtag/capture_dr }
set_lbist_pins update_dr { - jtag/update_dr }
set_lbist_pins test_logic_reset { - jtag/tlr }
set_lbist_pins tap_instruction_decode { - jtag/scanCfgReg_en }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

This example illustrates how to specify multiple scan enable signals without pads. Since the scan enable pin names should be specified in pairs, a dash is used for each of the non-existent internal pin names.

```
set_lbist_pins scan_en {scan_en - wrp_scan_en -}
```

Example 3

The following example renames LogicBIST ports back to their pre-2018.1 release versions for purposes of backwards annotation.

```
set_lbist_pins tck tck -on_controller_module
set_lbist_pins setup_shift_scan_in lbist_scan_in -on_controller_module
set_lbist_pins setup_shift_scan_out lbist_scan_out -on_controller_module
set_lbist_pins tap_instruction_decode sib_en -on_controller_module
set_lbist_pins test_logic_reset sib_reset -on_controller_module
set_lbist_pins capture_dr sib_capture_en -on_controller_module
set_lbist_pins shift_dr sib_shift_en -on_controller_module
set_lbist_pins update_dr sib_update_en -on_controller_module
set_lbist_pins edt_clock edt_clock -on_controller_module
set_lbist_pins test_clock edt_clock -on_controller_module
set_lbist_pins clock_out lbist_clock -on_controller_module
set_lbist_pins prpg_en prpg_en -on_controller_module
set_lbist_pins misr_en misr_en -on_controller_module
set_lbist_pins occ_shift_clock clk_ctrl_shift_clk -on_controller_module
set_lbist_pins occ_scan_en clk_ctrl_scan_en -on_controller_module
set_lbist_pins occ_capture_en clk_ctrl_capture_en -on_controller_module
```

Related Topics

report_lbist_pins

set_edt_options

set_lbist_controller_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_lbist_power_controller_options

Context: dft -logic_bist

Mode: setup

Specifies creation of the low-power shift controller for LogicBIST.

Usage

set_lbist_power_controller_options [**shift** {**none** | **disabled** | **enabled**}
[-min_switching_threshold_percentage ***sw_percentage***]]

Description

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

This command is applicable to the current LogicBIST block.

When you specify this command during IP creation, the tool creates the low-power shift controller for LogicBIST. This command is carried forward in the generated fault simulation dofile to indicate whether the current EDT block has low-power shift logic synthesized (automatically added to the fault simulation dofile).

Arguments

- **shift none** | **disabled** | **enabled**

  A literal pair that indicates the configuration parameters for controlling shift power.

  Choose one of the following:

  - **none** — Indicates that the low-power controller is not synthesized.
  - **disabled** — Forces a LogicBIST run to bypass the low power controller.
  - **enabled** — Enables the low-power controller.

  When you specify either 'enabled' or 'disabled', a low-power controller with the specified switching threshold percentage is synthesized.

- -min_switching_threshold_percentage ***sw_percentage***

  An optional switch and integer that specifies the switching threshold value for the hardware default mode. Valid integer values are between 1 and 50. The default value is 15 percent. During fault simulation, you can specify any switching threshold.

Examples

This example shows three EDT/LogicBIST blocks, two of which are to be synthesized with both EDT and LogicBIST low-power logic, while the third block does not have low power shift logic.

add_edt_block lpblk1

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_edt_power_controller shift enabled -min_switching_threshold 20
set_lbist_power_controller_options shift enabled
add_edt_block lpblk2
set_edt_power_controller shift enabled -min_switching_threshold 8
set_lbist_power_controller_options shift enabled
add_edt_block rgblk
```

Related Topics

set_edt_power_controller

set_power_control

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_learn_report

Context: all contexts

Mode: setup, analysis

Prerequisites: The current design must be set.

Specifies whether the report_gates command can display the learned behavior for a specific gate.

Usage

set_learn_report {OFf | ON}

Description

The set_learn_report command specifies whether the report_gates command should include the information that the tool collects during the static learning process. The application automatically performs the static learning process immediately after it flattens the simulation model, which happens when you leave the Setup mode or issue the create_flat_model command. The static learning process provides general information on the design that the tool can then use in speeding up the ATPG process (such as values that are impossible on other gates if the selected gate is at a specific value.)

Once you enable access to the static learned information with the set_learn_report command, you can specify for the tool to display the learned information on a selected gate by using the report_gates command.

While you can also access the learned information with the report_gates command by using the -Type option, this method displays the information for all the gates of the specified gate type. When you enable access with the set_learn_report command, the tool automatically displays the learned information with all command options. Therefore, you can restrict the report to the learned information on an individual object.

Arguments

- OFf

  An optional literal that disables access to the learned behavior information. This is the default.

- ON

  An optional literal that enables access to the learned behavior information.

Examples

The following example enables access to the learned behavior and then accesses that information:

```
set_learn_report on
report_gates 28
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
/MX3/OR1 (28)  OR
    IO      I  20-/MX3/AN2/OUT
    I1      I  24-/MX3/AN1/OUT
    OUT     O  37-/OUT0
    Learned behavior:  MUX(9,13,17)
```

Related Topics

report_gates

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_lfsr_seed

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Sets the initial value of a PRPG or a MISR.

Usage

set_lfsr_seed ***name seed*** [-Binary | -Hexadecimal]

Description

The set_lfsr_seed command initializes the value of a PRPG or MISR with a seed value. The seed value must be specified by a hexadecimal or binary number greater than zero.

Arguments

- ***name***

  A required string that specifies the name of the PRPG or MISR.

- ***seed***

  A required number, greater than zero, that specifies the initial state of the LFSR.

- -Binary

  An optional switch that specifies to output the value of the *seed* argument in binary format. By default, the seed is output in hexadecimal.

- -Hexadecimal

  An optional switch that specifies to output the value of the *seed* argument in hexadecimal format. This is the default.

Examples

set_lfsr_seed prpg 5

Related Topics

add_lfsr_taps

add_lfsrs

delete_lfsrs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_lfsrs

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Changes the *shift_type* and *tap_type* default settings for the add_lfsrs and add_lfsr_taps commands.

Usage

set_lfsrs [-Both | -Serial | -Parallel] [-Out | -In | -Ca]

Description

The set_lfsrs command controls the default setting for the *shift_type* and *tap_type* switches. You specify the LFSR's shift technique by using one of the following *shift_type* switches: -Both, -Serial, or -Parallel. You specify the placement of the exclusive-OR taps by using one of the following *tap_type* switches: -Out or -In; or you specify the type of LFSR as cellular automata using the *tap_type* switch -Ca. When you change one or both of the default settings, all future add_lfsrs and add_lfsr_taps commands use the new default.

Arguments

The following lists the three *shift_type* switches of which you can choose only one:

- -Both

  An optional switch specifying that the LFSR shifts both serially and in parallel. This is the default behavior upon invocation of the tool.

- -Serial

  An optional switch specifying that a serial shift LFSR shifts a number of times equal to the length of the longest scan chain for each scan pattern.

- -Parallel

  An optional switch specifying that a parallel shift LFSR shifts once for each scan pattern.

  The following lists the three *tap_type* switches of which you can only choose one:

- -Out

  An optional switch that places the exclusive-or taps outside the register path. This is the default upon invocation of the tool.

- -In

  An optional switch specifying that the exclusive-or taps are placed outside the register path.

- -Ca

  An optional switch specifying that the default PRPG type is cellular automata.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example changes the default *shift_type* setting to Serial and the default *tap_type* switch to In:

```
set_lfsrs -serial -in
add_lfsrs lfsr1 prpg 5 13
add_lfsrs lfsr2 prpg 5 11
add_lfsr_taps lfsr1 2 3
add_lfsr_taps lfsr2 3 4
set_system_mode analysis
```

Related Topics

add_lfsrs

add_lfsr_taps

delete_lfsrs

delete_lfsr_taps

report_lfsrs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_license_queue_timeout

Context: all contexts

Mode: all modes

Specifies the amount of time that the tool queues for one or more required licenses when not immediately available.

Usage

set_license_queue_timeout [{*value* | unlimited | no_queue}]
[-release_and_requeue_after *release_and_requeue_after*]

Description

The set_license_queue_timeout command configures the time the tool waits for acquiring one or more required licenses (for example, during set_context). You can specify the same value during tool startup with the command line switch -license_wait (for example the "tessent -license_wait" command).

When the tool requires more than one license or if some licenses have alternatives, the tool queues for all of the possible licenses and acquires the first available. The tool reports progress every minute, showing which licenses are still missing and how long before timeout. If the tool cannot acquire a license during the specified time, the tool reports an error and aborts the operation that requested the license. You can interrupt the license queuing with Ctrl-C, which also aborts the operation that requested the license(s).

When requesting additional licenses for multiprocessing, you must use the "set_multiprocessing_options -license_timeout" command to change the timeout with more than one process. In this specific case, the set_license_queue_timeout command has no affect on the license timeout.

Arguments

- *value*

  A positive integer that specifies the queuing time in minutes.

- unlimited

  An optional string that specifies that there is no time limit and the tool waits until a license is available. This is the default behavior.

- no_queue

  An optional string that specifies that the tool not queue for a license and issue an error if not available immediately.

- -release_and_requeue_after *release_and_requeue_after*

  A switch and positive integer pair that specifies how many minutes the tool queues before releasing already-acquired licenses and performing a re-queue of all required licenses.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When you specify this option, the Tessent Shell tool's license queuing mechanism releases licenses the tool has acquired with every -release_and_requeue_after minutes before re-requesting them. This enables other tools to acquire the licenses under the following circumstances:

- The tool is queuing for more than one license.
- The tool has acquired one license.
- The tool is queuing for one or more licenses that are currently in queue state.

Examples

The following example sets the license queue timeout value to 3 minutes, then displays the timeout value that was set:

```
SETUP> set_license_queue_timeout 3
SETUP> get_license_queue_timeout

3
```

Related Topics

get_license_queue_timeout

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_list_file

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the name of the list file into which the tool places the pins' logic values during simulation.

Usage

set_list_file [*filename*] [-Replace]

Description

The set_list_file command specifies the file in which the tool places the simulation values for the pins which you previously identified with the add_lists command. The default behavior is for the tool to display the simulation values for the pins on standard output.

You can display the list of reported pins by using the report_lists command.

Arguments

- *filename*

  A string that specifies the name of the file in which the tool places the logic values of pins during simulation. If you do not provide a *filename*, the default is standard output.

- -Replace

  An optional switch that replaces the contents of the file if the *filename* already exists.

Examples

The following Tessent FastScan example creates a file to store simulation values that are being reported:

```
set_system_mode analysis
add_lists  /i_1006/o  /i_1007/o
set_list_file listfile
create_patterns
```

Related Topics

add_lists

delete_lists

report_lists

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_load_unload_timing_options

Context: all contexts

Mode: all modes

Sets the scan shift, SSN bus clock period, and other timing aspects of the load/unload procedure during SSN or LogicBIST scan operations for the current physical block.

Usage

SSN Usage

set_load_unload_timing_options
[-usage {ssn}]
[-shift_clock_period *time* [s | ms | us | ns | ps | fs]]
[-ssn_bus_clock_period *time* [s | ms | us | ns | ps | fs]]
[-scan_en_setup_extra_cycles *int*]
[-scan_en_hold_extra_cycles *int*]
[-edt_update_setup_extra_cycles *int*]
[-edt_update_hold_extra_cycles *int*]
[-generated_timeplate_name *timeplate_name*]
[-ssn_bus_inputs_setup_relative_to_active_edge_percentage *int*]
[-ssn_bus_outputs_measure_after_full_period_percentage *int*]

LogicBIST Usage

set_load_unload_timing_options
[-usage {lbist}]
[-scan_en_setup_extra_cycles *int*]
[-scan_en_hold_extra_cycles *int*]
[-edt_update_setup_extra_cycles *int*]
[-edt_update_hold_extra_cycles *int*]
[-prpg_en_setup_extra_cycles *int*]
[-prpg_en_hold_extra_cycles *int*]
[-misr_en_setup_extra_cycles *int*]
[-misr_en_hold_extra_cycles *int*]

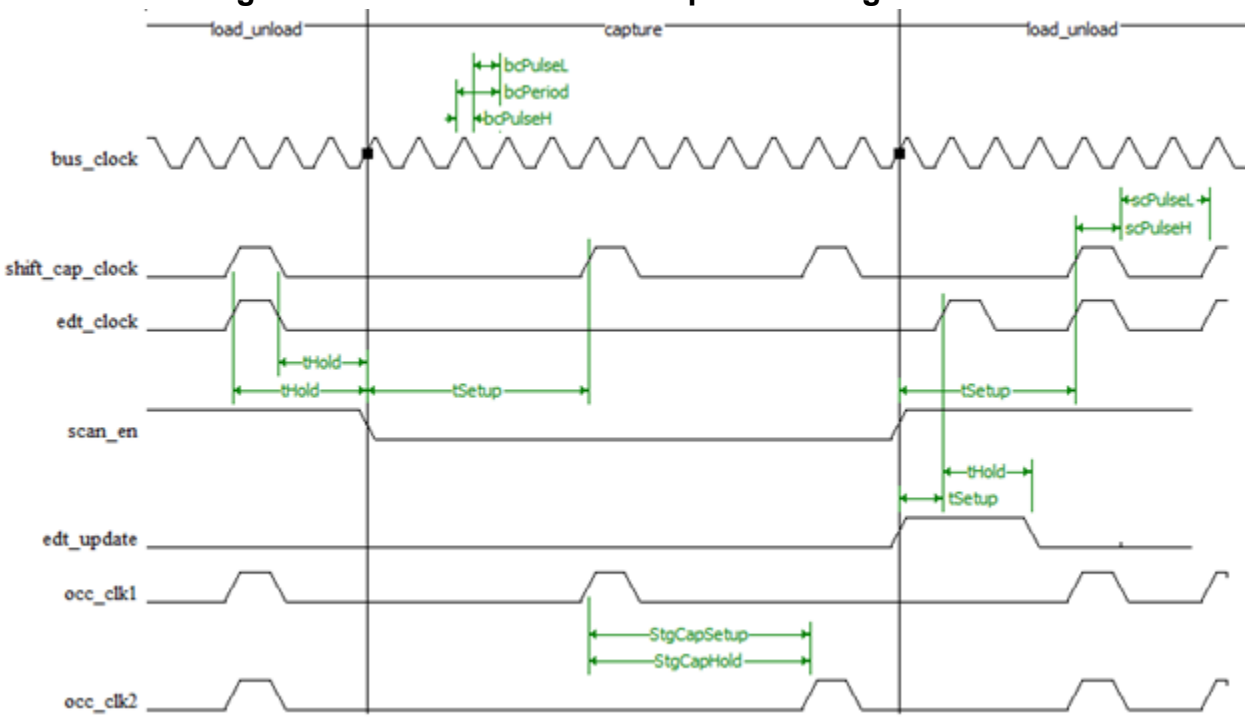
Description

The set_load_unload_timing_options command sets timing aspects for the load/unload procedure during SSN scan operations. These aspects are limited to the scope of the current physical block, as defined by the set_current_physical_block command. By default, the current physical block includes the entire design.

The ScanHost node is fully programmable and capable of generating scan timing that is consistent with the specification of this command. Figure 8-9 shows waveforms separated into three regions: the end of the load/unload region before the capture region, the capture region, and the start of the next load/unload region after the capture region.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 8-9. Load/Unload and Capture Timing With SSN



Typically, the SSN bus clock period runs faster than the maximum frequency of the shift capture clock. By default, the maximum frequency for the SSN bus_clock period is 400 MHz, and the default maximum frequency for the shift_capture_clock is 100 MHz. The set_load_unload_timing_options command enables you to modify aspects shown in the timing diagram, such as the maximum frequency of the clocks, and the amount of setup and hold margin provided between the shift_capture_clock and the controls (scan_en and edt_update).

The required amount of setup and hold timing between scan_en/edt_update and the shift_capture_clock is difficult to predict until you have implemented the clock trees and the high-fanout distribution buffering schemes within the layout implementation step.

After synthesis, those are still ideal nets, making it is impossible to get a precise timing relationship between the clock and the controls. To account for this, the ScanHost node has fully programmable timing, configured with the set_load_unload_timing_options and set_capture_timing_options commands.

Note

When you use the set_load_unload_timing_options command, the measure_sco in the shift procedure is set to 45% of the period.

The extract_icl command automatically invokes extract_sdc to create an SDC file. The SDC includes a Tcl proc named “set_load_unload_timing_options,” which supports the same switches as this command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For each physical block, the recommended use model is:

- In Tessent Shell, create an SDC file using the extract_icl command. This file contains the set_load_unload_timing_options Tcl proc.
- Source this file in your static timing analysis (STA) environment.
- Create a standalone file containing only the call to the set_load_unload_timing_options proc and source it within your STA tool. The values in this file overwrite the default values in the SDC file.

  Note

  The "-usage ssn" option is required for the set_load_unload_timing_options command invocation.

- Adjust the set_load_unload_timing_options switch values in the standalone file until all shift mode timing violations are resolved.
  - Typically, changes to the clock period are not required, but it may be necessary to specify extra setup or hold cycles for scan_en and edt_update.
- In Tessent Shell, apply those timing definitions by issuing the set_current_physical_block command and sourcing the standalone file containing the set_load_unload_timing_options call used during STA.
- Regenerate the post-layout retargeted and ATPG patterns using the precise timing specification that you used to close timing during post-layout STA. (Refer to the section "ScanHost Scan Interface Operation and Timing Requirements" on page 4498 for more information.)

Arguments

- -usage {ssn}

  An optional switch and value pair that specifies the SSN usage. Although this is an optional switch, it is mandatory in most cases.

- -usage {lbist}

  An optional switch and value pair that specifies the LogicBIST usage. Although this is an optional switch, it is mandatory in most cases.

- -shift_clock_period *time* [s | ms | us | ns | ps | fs]

  An optional switch and value pair that specifies the shift clock period. The default value is 10ns. Optionally, choose from the following time units:

  s — seconds.

  ms — milliseconds.

  us — microseconds.

  ns — nanoseconds. This is the default time unit.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ps — picoseconds.

fs — femtoseconds.

- -ssn_bus_clock_period *time* [s | ms | us | ns | ps | fs]

  An optional switch and value pair that specifies the SSN bus clock period. The default is 5 ns (200 MHz) when generating patterns at the chip level (when get_design_level returns "chip") and the SSN clocks are defined without the -freq_multiplier switch. The default is 2.5 ns (400 MHz) when generating simulation ATPG or scan retargeting patterns at the child physical block level or at the chip level when the SSN clocks are defined with a -freq_multiplier value of 2 or more. Optionally, choose from the following time units:

  s — seconds.

  ms — milliseconds.

  us — microseconds.

  ns — nanoseconds. This is the default time unit.

  ps — picoseconds.

  fs — femtoseconds.

  Note

  The differing default values at different design levels is because the recommendation is to use an internal SSN bus frequency of 400 MHz and to use BusFrequencyDivider and BusFrequencyMultiplier nodes to bring the SSN bus frequency down to 200 MHz at the chip boundary.

- -scan_en_setup_extra_cycles *int*

  An optional switch and value pair that adds extra cycles between the transition of the scan_en signal and the rising and falling edge of the shift_capture clock. One extra cycle corresponds to the time taken by an internal shift clock period. Without throttling, this also corresponds to the time to shift in one packet. With throttling, this can correspond to more than a single packet. The default is 1.

  For more information about throttling, see "Yield Statistics on ATE With SSN" in the *Tessent Shell User's Manual*.

- -scan_en_hold_extra_cycles *int*

  An optional switch and value pair that adds extra cycles between the rising and falling edges of the shift_capture clock and the transitions of the scan_en signal. One extra cycle corresponds to the time taken by an internal shift clock period. Without throttling, this also corresponds to the time to shift in one packet. With throttling, this can correspond to more than a single packet. The default is 1.

  For more information about throttling, see "Yield Statistics on ATE With SSN" in the *Tessent Shell User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -edt_update_setup_extra_cycles *int*

  An optional switch and value pair that adds extra cycles between the transition of edt_update signal and the rising edge of the shift_capture clock. One extra cycle corresponds to the time taken by an internal shift clock period. Without throttling, this also corresponds to the time to shift in one packet. With throttling, this can correspond to more than a single packet. The default is 1.

  For more information about throttling, see "Yield Statistics on ATE With SSN" in the *Tessent Shell User's Manual*.

- -edt_update_hold_extra_cycles *int*

  An optional switch and value pair that adds extra cycles between the rising edge of the shift_capture clock and the transitions of the edt_update signal. One extra cycle corresponds to the time taken by an internal shift clock period. Without throttling, this also corresponds to the time to shift in one packet. With throttling, this can correspond to more than a single packet. The default is 1.

  For more information about throttling, see "Yield Statistics on ATE With SSN" in the *Tessent Shell User's Manual*.

- -generated_timeplate_name *timeplate_name*

  An optional switch and string pair that specifies a base name for the timeplate that the tool uses when generating the default timeplate during system mode transition. The base name can consist only of letters, digits, and underscores, and cannot start with a digit. Each command using this switch must specify a unique base name.

- -prpg_en_setup_extra_cycles *int*

  An optional switch and integer pair that specifies the default number of extra cycles from the transition of the prpg_en signal to the rising edge of the edt_lbist_clock. The specified value cannot be different than the -misr_en_setup_extra_cycles value. The default is 0.

- -prpg_en_hold_extra_cycles *int*

  An optional switch and integer pair that specifies the default number of extra cycles from the rising edge of the edt_lbist_clock to the transitions of the prpg_en signal. The specified value cannot be different than the -misr_en_hold_extra_cycles value. The default is 0.

- -misr_en_setup_extra_cycles *int*

  An optional switch and integer pair that specifies the default number of extra cycles from the transition of the misr_accumulate_en signal to the rising edge of the edt_lbist_clock. The specified value cannot be different than the -prpg_en_setup_extra_cycles value. The default is 0.

- -misr_en_hold_extra_cycles *int*

  An optional switch and integer pair that specifies the default number of extra cycles from the rising edge of the edt_lbist_clock to the transitions of the misr_accumulate_en signal. The specified value cannot be different than the -prpg_en_hold_extra_cycles value. The default is 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -ssn_bus_inputs_setup_relative_to_active_edge_percentage *int*

  An optional switch and integer pair that controls the SSN bus input setup time relative to the active edge. The default SSN timing permits a skew of half a cycle between the SSN clock and the SSN input data when using a BusFrequencyDivider or a Receiver1xPipeline node, both of which are capable of strobing data on the trailing edge of the clock.

  The default value for this setting is 50, meaning that SSN datapaths strobed on the first edge (such as datapaths that start with an SSH or a regular Pipeline) have bus inputs forced at 50% of the SSN bus period. SSN datapaths strobed on the other edge (such as those that start with a Receiver1xPipeline or a BusFrequencyMultiplier) have their bus inputs forced at 0% of the SSN bus period.

  Increasing the setup time moves the force time closer to the beginning of the SSN bus cycle. For SSN datapaths strobed on the other edge, the SSN bus input force moves to the end of the previous SSN bus cycle.

  Note

  Changing this value may result in zero-delay simulation mismatches.

- -ssn_bus_outputs_measure_after_full_period_percentage *int*

  An optional switch and integer pair that increases the SSN worst-case loop timing by moving the strobe past the launch edge of the new data by the specified percentage. The default value is 0.

  Note

  Changing this value results in zero-delay simulation mismatches.

Examples

Example 1

The following example shows the content of a file where the load/unload timing parameters are specified. That file is sourced within the STA script of the BlockA physical block and re-sourced within Tessent Shell following the "set_current_physical_block BlockA" command.

The following is content of the *BlockA/load_unload_timing_options.tcl* file:

```
set_load_unload_timing_options      \
 -usage ssn                         \
 -shift_clock_period            10  \
 -ssn_bus_clock_period          2.5 \
 -scan_en_setup_extra_cycles    0   \
 -scan_en_hold_extra_cycles     2   \
 -edt_update_setup_extra_cycles 1   \
 -edt_update_hold_extra_cycles  1
```

The following command is in the STA script of the physical block called BlockA:

source BlockA/load_unload_timing_options.tcl

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following commands are in the Tessent Shell dofile during the post-layout ATPG and scan retargeting runs:

set_current_physical_block -module BlockA
source BlockA/load_unload_timing_options.tcl
set_current_physical_block -module BlockB
source BlockB/load_unload_timing_options.tcl

Example 2

The following is an LBIST example that shows the content of a file where the load/unload timing parameters are specified for the TK/LBIST independent insertion flow. That file is sourced within the STA script of the BlockA physical block and re-sourced within Tessent Shell following the "set_current_physical_block BlockA" command.

The content of *BlockA/load_unload_timing_options.tcl* file is:

```
set_load_unload_timing_options \
 -usage lbist \
 -scan_en_setup_extra_cycles    0 \
 -scan_en_hold_extra_cycles    2 \
 -edt_update_setup_extra_cycles    1 \
 -edt_update_hold_extra_cycles    1
 -prpg_en_setup_extra_cycles    1 \
 -prpg_en_hold_extra_cycles    1
 -misr_en_setup_extra_cycles    1 \
 -misr_en_hold_extra_cycles    1
```

The following figures show waveforms separated into three regions:

- The end of the load/unload region before the capture region.
- The capture region.
- The start of the next load/unload region after the capture region.

Figure 8-10 shows the load/unload timing waveform when the EDT and LBIST are inserted in the same insertion pass.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 8-10. Timing When Independent Insertion Flow is Not Used

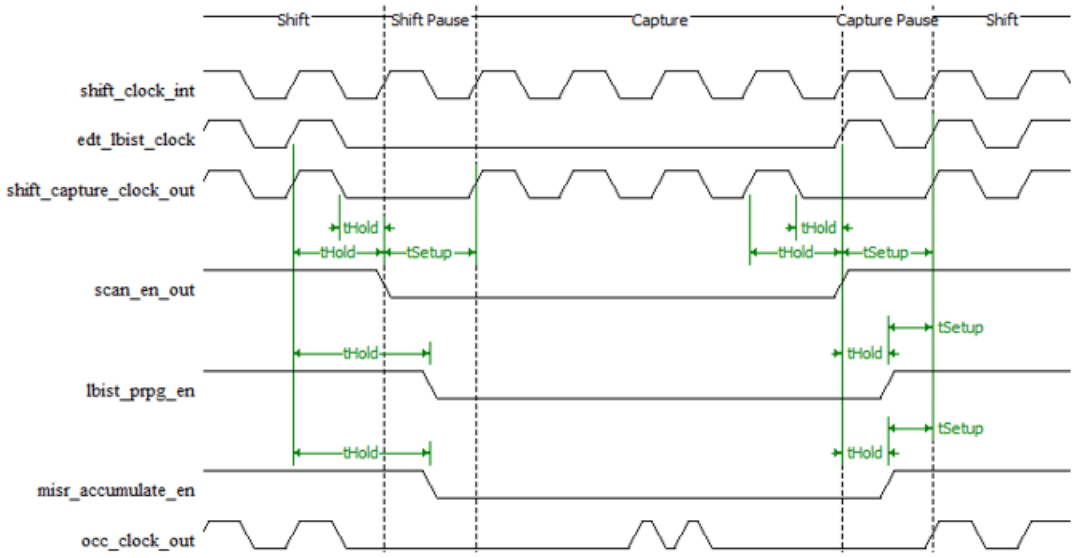
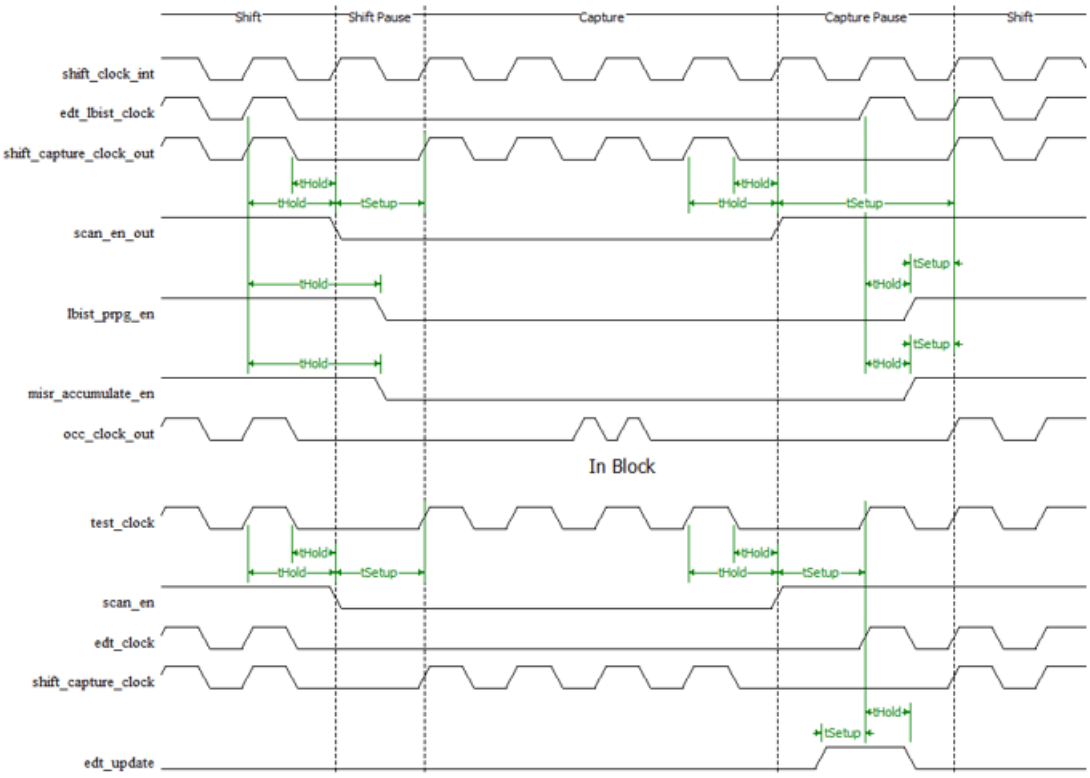


Figure 8-11 shows the base timing waveform for the independent insertion flow where the LBIST-ready controller exists in another block.

Figure 8-11. Timing for Independent Insertion Flow



See "LBIST Load/Unload Timing" in the *TK/LBIST Flow User's Manual* for more information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

get_load_unload_timing_option

report_load_unload_timing_options

reset_load_unload_timing_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_logfile_handling

Context: unspecified, all contexts

Mode: all modes

Specifies for the tool to direct the transcript information to a file.

Usage

set_logfile_handling [-RESume | {*filename* [-Replace | -Append]}]

Description

The set_logfile_handling command causes the tool to write the transcript information, which includes the commands and the corresponding output (if any), into the file you specify. You can execute the set_logfile_handling command at any time, as many times as you need.

In the logfile, the command keyword (or sub-command when a tool command is embedded in a Tcl command) precedes all commands that the tool executes. You can easily search for the executed commands, generate a separate dofile containing those commands, and then execute the dofile, thereby rerunning those commands within the tool.

When you set the logfile handling, the tool still writes the same information to the session transcript window in addition to the logfile. However, you can disable the writing of the information to the transcript window with the set_screen_display command.

If you want to stop writing to a logfile, issue the set_logfile_handling command with no options, which closes the appropriate file.

Arguments

- -Resume

  An optional switch that specifies for the tool to resume writing transcript output to the logfile specified at invocation with the -Logfile invocation switch. You do not need to include the name of the invocation logfile with this switch; the tool remembers the name.

- *filename*

  A string that specifies the name of the file to which you want the tool to write the transcript output. This string can be a full pathname or a leafname. If you only specify a leafname, the tool creates the file in the directory from which you invoked the tool.

  If you do not specify a *filename*, the tool discontinues writing to the logfile and closes it.

- -Replace

  An optional switch that forces the tool to overwrite the file if a file by that name already exists.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Append

  An optional switch that causes the tool to begin writing the transcript at the end of the specified file.

Examples

The following example writes a logfile and disables the display of the transcript information in the transcript window:

```
set_logfile_handling /user/designs/setup_logfile
set_screen_display off
add_clocks 0 clk
add_clocks 1 pre clr
report_clocks
```

The following information shows what the logfile contains after running the preceding set of commands:

```
//  command:  set_screen_display off
//  command:  add_clocks 0 clk
//  command:  add_clocks 1 pre clr
//  command:  report_clocks
User-defined Clocks (2):
========================
  Sync and Async Source Clocks
  ============================
   ------  ---------  -----------  --------
   Name    Off State  Constraints  Internal
   ------  ---------  -----------  --------
   'PRE'       1                      No
   'CLR'       1                      No
   'CLK/       0                      No
```

Related Topics

get_logfile

report_environment

set_transcript_style

set_screen_display

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_logical_design_libraries

Context: all contexts

Mode: setup

Defines the set of available logical libraries that can be used by the read_verilog and read_vhdl -in_library option.

Usage

set_logical_design_libraries **-logical_library_list** ***logical_library_list***
[-default_library *default_library_name*] [-force]

Description

You must use this command before any design modules have been read in unless you specify the -force switch. If you specify -force, all design data that was read before is lost and must be re-read or recreated.

This command is not valid in the no_rtl context. The tool uses a single library named “work.”

As you can see in the example, you can create a Tcl list of library names and use it as the *logical_library_list* value. You can sort this Tcl list from the bottom up to order the VHDL libraries into an order such that the libraries that depend on other libraries are defined after those they depend on. The default library is the last item in the list unless the -default_library is used to specify a different one.

Arguments

- **-logical_library_list** ***logical_library_list***

  A required switch and string pair that specifies the list of logical libraries.

- -default_library *default_library_name*

  An optional switch and string pair that specifies the library name to be used as the default library. If this switch is not specified, the last library name in the logical_library_list is used as the default library.

  Except for the get_modules command, all commands that accept a module name as one of their options look for the module name inside the default library. For example, to create an instance of module M1 from library L1, you use the following command:

  ```
  create_instance instance_name -of_module [get_module M1 -
  in_library L1]
  ```

  Using -of_module M1 creates an instance of module M1 from the default library.

- -force

  An optional switch that suppresses the error message normally displayed when this command is executed after modules have already been read in or created. When the -force option is used, all pre-existing modules are deleted.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

In this example, a Tcl list is defined with all logical library names and is ordered with the libraries that depend on other libraries defined after those they depend on. A Tcl array called vhdl_files is used to specify which files goes in which library. A simple design-independent script at the bottom is used to load those VHDL files into the right libraries. A similar technique can be used for loading Verilog into multiple design libraries.

```
set vhdl_dir design/vhdl/src

set lib_list [list utility lib1 work]

set vhdl_files(utility) [list \
        ${vhdl_dir}/f1.vhd\
        ${vhdl_dir}/f2.vhd\
]

set vhdl_files(lib1) [list \
        ${vhdl_dir}/f3.vhd\
        ${vhdl_dir}/f4.vhd\
]

set vhdl_files(work) [list \
        ${vhdl_dir}/f5.vhd\
        ${vhdl_dir}/f6.vhd\
]

set_logical_design_libraries -logical_library_list $lib_list

foreach lib $lib_list {
        if {[array names vhdl_files -exact $lib] == $lib} {
            read_vhdl $vhdl_files($lib) -in_library $lib
        }
}
```

Related Topics

get_logical_library_list

read_verilog

read_vhdl

set_current_design

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_loop_handling

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup

Specifies how the tool handles feedback networks.

Usage

set_loop_handling [Tiex | Simulation] [-MAX_Stored_loops {number | UNLIMITED}]

Description

The set_loop_handling command lets you perform DRC simulation of circuits containing combinational feedback networks.

The set_loop_handling command specifies the tool's loop handling behavior by either inserting a TIE-X gate or by stabilizing the loop values through an iterative simulation process.

For an iterative simulation, the tool inserts FB_BUF gates to break the combinational loops. It also optimizes the number of iterations performed to both stabilize the circuit and minimize the impact of the iterative process on performance and memory usage.

For another look at combinational feedback loops, refer to "Feedback Loops" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- Tiex

  A literal that specifies that TIE-X gates are used to break combinational loops.

- Simulation

  A literal that specifies for the tool to use a simulation process to stabilize values in the loop. This option gives more accurate simulation results than other options. This is the default.

- -MAX_Stored_loops {number| UNLIMITED}

  An optional switch and integer or keyword pair that specifies the maximum number of loops that can be stored for reporting (using the report_loops command). By default, the tool stores up to1000 loops. Use this switch if you want to store a specific number or all loops for reporting after the flattening process.

Examples

Example 1

The following Tessent FastScan example inserts a TIE-X gate to break any identified combinational or sequential asynchronous loop, then performs ATPG:

```
set_loop_handling tiex
set_system_mode analysis
create_patterns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example shows how the tool indicates a design with more than 1000 combinational loops during the flattening process, and how to store all loops and write the loops to file loops.txt.

```
create_flat_model

// Note: Number of combinational loops exceeds storage limit (1000).

//      Use 'set_loop_handling -max_stored_loops' to modify the number of loops stored and
//      reported by 'report_loops'.

// Flattening process completed,  cell instances=28234, gates=45400,  PIs=110,  POs=103,
// CPU time=93.00 sec.

// ---------------------------------------------------------------------------
// Begin circuit learning analyses.
// ---------------------------------------------------------------------------
// Equivalent gates=54   classes=0   CPU time=9.00 sec.

set_loop_handling  -max_stored_loops unlimited

// Warning: Flat model deleted.

create_flat_model

// Flattening process completed,  cell instances=28234,  gates=45400,  PIs=110,
// POs=103,  CPU time=93.00 sec.

// ---------------------------------------------------------------------------
// Begin circuit learning analyses.
// ---------------------------------------------------------------------------
// Equivalent gates=54   classes=0   CPU time=9.00 sec.

report loops > loops.txt
```

Related Topics

report_loops

set_tla_loop_handling

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_lpct_condition_bits

Context: dft -edt

Mode: setup

Specifies the condition scan cells (condition bits) in the design core to connect to the Type 3 low pin count test (LPCT) controller.

Usage

set_lpct_condition_bits {**-Condition** {**Reset** | **Scan_enable**}
[*pin_name* -on_controller_module] {**-From *connection_name*** | **-No_connection**}

Description

The condition scan cells (condition bits) must already exist in the design core. If no condition bits are specified, the LPCT controller fully controls the reset and scan_enable signals. When LPCT condition bits are specified, the appropriate control bits are added to the test configuration data register in the LPCT controller. For each scan enable condition and each reset condition, one bit is added to the test configuration data register.

During ATPG, the scan cell specified using "set_lpct_condition_bits -from connection_name" is justified to the correct value to ensure the correct value of reset and scan enable.

This command is used in the IP Creation phase; it can only be used to create the Type 3 LPCT controller. For more information, see "Type 3 LPCT Controller" in the *TestKompress User's Manual*.

Arguments

- **-Condition** {**Reset** | **Scan_enable**}

  Required switch and literal pair that specifies the condition bit type. Literal options include the following:

  Reset — Condition bit for controlling design reset during capture.

  Scan_enable — Condition bit for controlling design scan enable during capture.

- *pin_name* -on_controller_module

  Optional string and switch pair that specifies the pin name in the controller RTL for that particular input; this does not specify the connection.

- **-From *connection_name*** | **-No_connection**

  Optional switch and string or switch that specifies the pin corresponding to the condition bits. This argument specifies the input connection to the LPCT controller. The -no_connection switch removes a previously specified condition.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example defines condition bits in the design to control the reset and scan_enable signals for the LPCT controller:

```
// Condition scan cell outputs (to be connected to LPCT logic)
set_lpct_condition_bits -condition reset -from c1/Q
set_lpct_condition_bits -condition scan_en -from c2/Q
```

Related Topics

report_lpct_configuration

report_lpct_pins

set_lpct_controller

set_lpct_instances

set_lpct_pins

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_lpct_controller

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Enables the creation of and specifies the configurations for a low pin count test (LPCT) controller for use with EDT logic.

Usage

set_lpct_controller {**OFF**
| { [ON] [-MAX_SHift_cycles *shift_cycles*] [-MAX_CApture_cycles *capture_cycles*]
[-MAX_SCan_patterns *pattern_count*] [-MAX_CHain_patterns *pattern_count*]
[-TEst_mode_detect {SIgnal | SEquence *binary_seq count*}]
[-Shift_control {Enable | Clock | None}] [-LOad_unload_cycles *count1 count2*]
[-TAp_controller_interface {OFf | ON}] [-Generate_scan_enable {OFf | ON}]
[-TESSENT_Occ {Auto | ON}] [-TESSENT_OCC_Capture_cycle_width {1 | 2 | 3}]}}

Description

Use this command during EDT logic creation to create the necessary LPCT controller logic for a test application and during pattern generation to verify the compatibility between the LPCT controller configuration and the design core. The tool also carries this command forward into the dofile output for ATPG.

Arguments

- **OFF**

  Required literal that disables the creation of an LPCT controller. Default setting.

- On

  Optional literal that enables the creation of an LPCT controller.

- -MAX_SHift_cycles *shift_cycles*

  Optional switch and integer pair that specifies the number of shift cycles in the longest scan chain. This number can be an estimate but it should take into consideration both the EDT and bypass modes. This number affects the size of the shift cycle counter.

  Note

  This option can only be used for the Type 3 LPCT controller. For more information, see "Type 3 LPCT Controller" in the *Tessent TestKompress User's Manual*.

- -MAX_CApture_cycles *capture_cycles*

  Required switch and integer pair that specifies the maximum number of capture cycles for all test patterns. This number should take into consideration all of the types of test patterns generated for the test application.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

This option can only be used for the Type 3 LPCT controller. For more information, see "Type 3 LPCT Controller" in the *Tessent TestKompress User's Manual*.

- -MAX_SCan_patterns *pattern_count*

  Optional switch and integer pair that specifies the maximum number of scan test patterns generated for any test pattern type in the test application. You must consider all the test pattern types used when determining this number.

  This number determines the size of the scan test pattern counter created for the LPCT controller. By default, a counter of size 20 bits is used which implies a maximum of 1048576 scan test patterns.

  Note

  This option can only be used for the Type 3 LPCT controller. For more information, see "Type 3 LPCT Controller" in the *Tessent TestKompress User's Manual*.

- -MAX_CHain_patterns *pattern_count*

  Optional switch that specifies the maximum number of chain test patterns generated for any test pattern type in your test application. You must consider the maximum number of test patterns generated for any chain test type when determining this number. The default size of the chain test pattern counter is 10 bits which implies no more than 1024 chain test patterns.

  Note

  This option can only be used for the Type 3 LPCT controller. For more information, see "Type 3 LPCT Controller" in the *Tessent TestKompress User's Manual*.

- -TEst_mode_detect {signal | sequence *binary_seq count*}

  Optional switch and literal that configures how the LPCT controller enters test mode. By default, test mode is enabled when the test_mode signal is asserted.

  signal — The Set LPCT Pin test_mode command specifies the name of the signal to be used to enable test mode.

  sequence <*binary_seq*> <*count*> —The LPCT controller enables test mode if the specified binary sequence, specified by *binary_seq*, is detected within the first *count* cycles after reset.

  Note

  This option can only be used for the Type 3 LPCT controller. For more information, see "Type 3 LPCT Controller" in the *Tessent TestKompress User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -SHIFT_Control {Enable | Clock | None}

  Optional switch and literal pair that specifies how the LPCT controller generates the shift clock control signal. Options include:

  Enable — A lpct_shift_en enable signal is output to generate the shift clock waveform. This is the default setting.

  Clock — A lpct_shift_clock signal is generated when an internal clock is pulsed during shift. Note, if all shift clocks are top-level clocks, they are controllable from top-pins during both shift and capture cycles; in this case, an additional lpct_shift_clock is not needed and is therefore not generated.

  None — No output signal. This option is used when clocks are available at the top-level.

- -LOAd_unload_cycles *count1 count2*

  Optional switch and double integer that specifies the number of cycles that precede and succeed the shift cycles in the load_unload procedure.

  During the IP creation phase, this argument has no impact on the hardware of Type 1 and Type 2 controllers, because they are fixed by design. The Type 1 controller has two pre-shift and two post-shift load/unload cycles, and the Type 2 controller has three pre-shift and two post-shift load/unload cycles. The tool issues an error if you specify any other value.

  Use this argument to generate Type 3 controller hardware that generates more pre-shift and post-shift cycles than the default values of zero and two, respectively. The tool issues an error if you specify values less than the minimum.

  For all controller types, if there is an incoming test procedure, the tool adds extra cycles to that test procedure so that the pre-shift and post-shift cycles in the load_unload procedure for EDT pattern generation match the -load_unload_cycles value count. For example, if the incoming test procedure already has one post-shift cycle, and this argument specifies two post-shift cycles, then one new post-shift cycle is added to the test procedure, so that there are two post-shift cycles during the load_unload procedure.

  During pattern generation, this argument is not meant for specifying the needed load/unload cycles and has no impact on pre-shift and post-shift load/unload cycles of any type of LPCT controllers. The tool generates patterns based on the LPCT hardware generated earlier in the IP creation phase. (You can use this command in the patterns -scan context for legacy support reasons only.) In the patterns -scan context, the tool simply rule-checks that the number of load/unload cycles in the test procedure file match the LPCT hardware.

- -TAp_controller_interface {OFf| ON}

  Optional switch and literal that specifies to use a TAP-generated scan enable version of the controller.

- -Generate_scan_enable {OFf | ON}

  Optional switch and literal that specifies whether the scan_en signal is generated internally by the LPCT controller.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

OFf — If the -TAp_controller_interface argument is set to Off, the LPCT controller has a top-level scan enable signal. This is the default configuration. If the -TAp_controller_interface argument is set to On, an error is issued.

ON — If the -TAp_controller_interface argument is set to Off, the scan enable signal is internally generated by the LPCT controller. If the -TAp_controller_interface argument is set to On, the LPCT the scan enable is generated using the TAP state machine.

- -TESSENT_Occ {Auto | ON}

  An optional switch and literal that generates a Tessent OCC-compatible LPCT controller for skeleton designs that do not contain Tessent OCC instances. You specify this switch during pre-synthesis IP creation. Choose one of the following options:

  auto — Infers the Tessent OCC instruments from the TCD and the ICL. This is the default.

  on — Enables generation of Tessent OCC compatible LPCT controller when OCC core instances are not added.

- -TESSENT_OCC_Capture_cycle_width {1 | 2 | 3}

  An optional switch and literal that specifies how many bits are present in the OCC. This must be a number between 1 and 3 (corresponding to capture_window_size 2 through 6 in the OCC DftSpecification). The default is 2.

Examples

The following example enables an LPCT controller and configures it to use a sequence to enter test mode:

```
//define parameters for test controller
```

set_lpct_controller on -max_shift_cycles 400 -max_capture_cycles 2
set_lpct_controller -test_mode_detect sequence 1100100101001110 64

Related Topics

report_lpct_configuration

report_lpct_pins

set_lpct_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_lpct_instances

Context: dft -edt

Mode: setup

Inserts the low pin count test (LPCT) controller and clock gater in a specified hierarchical instance within the design core.

Usage

set_lpct_instances -Block_location instance_pathname
-Clock_gater_location *instance_pathname*

Description

The set_lpct_instances command inserts the LPCT controller in a specified hierarchical instance within the design core. It also adds a clock gater, but only if "set_lpct_controller -shift_control *clock*" is specified.

Arguments

- -Block_location instance_pathname

  Optional switch and string pair that specifies an instance in the design core in which to insert the LPCT controller. By default, the LPCT controller is inserted at the top-level of the design core.

- -Clock_gater_location *instance_pathname*

  Optional switch and string pair that specifies an instance in which to insert the LPCT clock gater. By default, the clock gater is inserted at the top-level of the design core.

Examples

The following example places the LPCT controller inside the *my_design/xyz/foo* instance in the design core:

```
//define parameters for LPCT controller

set_lpct_controller on -max_shift_cycles 5000 -max_capture_cycles 4 -max_chain_patterns
1000
set_lpct_controller -shift_control clock
set_lpct_instances -block_location my_design/xyz/foo
```

Related Topics

report_lpct_configuration

report_lpct_pins

set_lpct_controller

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_lpct_pins

Context: dft -edt

Mode: setup

Specifies control pins and connections between the low pin count test (LPCT) controller and the core design.

Usage

Specify for all configuration

```
set_lpct_pins
    [CLock pin_name [internal_node_name] [-ON_controller_module] [-NOInv | -INV]] |
    [CLOCK_Mux_select [pin_name -ON_controller_module]
        {connection_name | -No_connection}]
    [CAPTURE_Enable [pin_name -ON_controller_module]
        {connection_name | -No_connection}]
    [SHIFT_Enable [pin_name -ON_controller_module]
        {connection_name | -No_connection}]
    [SHIFT_Clock [pin_name -ON_controller_module]
        {connection_name | -No_connection}]
```

Specify for Type 1 LPCT controllers (external scan_en)

```
set_lpct_pins
    [INPUT_Scan_enable pin_name [internal_node_name]
        [-ON_controller_module] [-NOInv | -INV]]
    [TEST_Clock_connection [pin_name -ON_controller_module]
        {connection_name | -No_connection}]
```

Specify for Type 2 or Type 3 LPCT controllers (internally-generated scan_en TAP or LPCT generated)

```
set_lpct_pins
    [Reset pin_name [internal_node_name] [-ON_controller_module]
        [-Active High | Low] [-NOInv | -INV]]
    [OUTPUT_Scan_enable [pin_name -ON_controller_module]
        {connection_name | -No_connection}]
```

Specify for Type 3 LPCT controller (scan_en generated by LPCT controller only)

```
set_lpct_pins
    [DATA_In pin_name [internal_node_name] [-ON_controller_module] [-NOInv | -INV]]
    [TEST_Mode pin_name [internal_node_name] [-ON_controller_module] [-NOInv | -INV]]
    [RESET_Out [pin_name -ON_controller_module] {connection_name | -No_connection}]
    [TEST_End [pin_name -ON_controller_module] {connection_name | -No_connection}]
    [TEST_Active [pin_name -ON_controller_module]
        {connection_name | -No_connection}]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Specify for Type 2 LPCT controller (scan_en generated using TAP state machine only)

set_lpct_pins
[SHIFT_Dr *pin_name* [*internal_node_name*] [-ON_controller_module]
[-NOInv | -INV]]
[CAPTURE_Dr *pin_name* [*internal_node_name*] [-ON_controller_module]
[-NOInv | -INV]]
[Update_dr *pin_name* [*internal_node_name*] [-ON_controller_module] [-NOInv | -INV]]
[-ON_controller_module] [-NOInv | -INV]]
[TMs *pin_name* [*internal_node_name*] [-ON_controller_module] [-NOInv | -INV]]
[TEST_Clock_connection [*pin_name* -ON_controller_module]
{*connection_name* | -No_connection}]

Description

Specifies control pins and connections between the low pin count test (LPCT) controller and the core design.

When using the LPCT controller in which scan_en is generated using TAP states, the clock and reset inputs of the controller should be connected to TCK and the test_logic_reset pins on the TAP controller.

For hybrid TK/LBIST IP generation, the dft -edt -lbist context supports this command.

Arguments

- CLock

  Required argument that specifies as always-pulse clock input for a flip-flop based design.

  This option can be used for Type 1, Type 2, and Type 3 LPCT controllers. For more information, see "Reduced Pin Count Requirements" in the *Tessent TestKompress User's Manual*.

- *pin_name*

  Optional string and switch pair that specify the pin name in the controller RTL for that particular input; this does not specify the connection. Depending on the application, it may be a top-level pin, an input pin, or an output pin as follows. Use this string with the -on_controller_module switch to name the output pin when scan cell conditions are used to reset the LPCT controller.

- *internal_node_name*

  Optional string used together with the pin_name argument to specify the top-level pin name and corresponding internal connection for controller input pins. You can use a literal dash "-" for the *pin_name* argument when no top-level pin exists in the design.

- -ON_controller_module

  Optional switch used with the *pin_name* argument that specifies the pin name in the controller RTL for each input; this does not specify the connection.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- *connection_name*

  Optional switch used to specify the names of one or more internal pins to connect to the controller output pins.

- -No_connection

  Optional switch used to specify that no connection is required for a controller output pin.

- -NOInv| -INV

  Optional switch that specifies a LPCT signal is inverted between the point where it is connected during EDT logic creation and the chip boundary in the final netlist used for pattern generation. The default is no inversion.

  You should specify -INV when there is an inversion between the internal connection name for the LPCT pin and the chip boundary such as when specifying an inverting pad.

  Specifying this inversion information during the EDT logic creation phase generates the correct dofiles and test procedure files for the pattern generation phase but has no effect on the EDT logic.

- CLOCK_Mux_select

  Optional argument that specifies the connection to be made to the clock_mux_select output of the controller. This pin is a MUX select signal that selects between the shift clock generated by the LPCT controller and the capture clock. If clock_mux_select is high, the shift clock is selected; if clock_mux_select is low, the capture clock is selected.

  This option can be used for Type 1, Type 2, and Type 3 LPCT controllers. For more information, see "Reduced Pin Count Requirements" in the *Tessent TestKompress User's Manual*.

- CAPTURE_Enable

  Optional argument that specifies the controller output signal that indicates when the controller is in capture mode. This option specifies the pin to which this capture_enable signal is to be connected.

  This option can be used for Type 1, Type 2, and Type 3 LPCT controllers. For more information, see "Reduced Pin Count Requirements" in the *Tessent TestKompress User's Manual*.

- SHIFT_Enable

  Optional argument that specifies a controller output signal that indicates when the controller is in shift. The shift_enable option specifies the pins to which the shift_enable signal should be connected. If you provide clock gating using "set_lpct_controller -shift_control enable", this option is required; it is used as the clock gater enable signal to produce the shift clock from a pulse-always clock.

  This option can be used for Type 1, Type 2, and Type 3 LPCT controllers. For more information, see "Reduced Pin Count Requirements" in the *Tessent TestKompress User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- SHIFT_Clock

  Argument that specifies the pin is a shift clock signal.

  This option can be used for Type 1, Type 2, and Type 3 LPCT controllers. For more information, see "Reduced Pin Count Requirements" in the *Tessent TestKompress User's Manual*.

- INPUT_Scan_enable

  Argument that specifies the top-level scan enable port.

  This is used as an input only to the Type 1 LPCT controller. This option can only be used for the Type 1 LPCT controller. For more information, see "Other LPCT Controller Types (Not Recommended)" in the *Tessent TestKompress User's Manual*.

  Note

  Low Pin Count EDT With DFT Signals is recommended as a superior alternative to the Type 1 LPCT controller.

- Reset

  Required argument that specifies the reset input signal for the controller. The reset pin can be shared with the design's functional reset. This can be either an external pin or internal power-on-reset.

  This option can only be used for the Type 3 LPCT controller. For more information, see "Type 3 LPCT Controller" in the *Tessent TestKompress User's Manual*.

- OUTPUT_Scan_enable

  Argument that specifies the list of internal pins to which the LPCT output scan enable pin is connected.

  This option is required when generating a Type 2 or Type 3 LPCT controller.

- DATA_In

  Optional argument that specifies the pin is a data input to the test controller. For the Type 3 controller, a test sequence and configuration is scanned in through this port. This port is usually shared with the edt_channel input.

  This option can only be used for the Type 3 LPCT controller. For more information, see "Type 3 LPCT Controller" in the *Tessent TestKompress User's Manual*.

- TEST_Mode

  Optional argument that specifies the pin is a test mode signal used to enable test mode. This signal is not used when sequence detection is used.

  This option can only be used for the Type 3 LPCT controller. For more information, see "Type 3 LPCT Controller" in the *Tessent TestKompress User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- RESET_Out

  Optional argument that specifies the reset signal to the design. This signal is generated by combining the functional reset signal (specified using "set_lpct_pins reset") with the reset condition or reset control configuration inside the controller. This enables the tool to detect faults connected to the reset signal even while the top-level reset pin is constrained to OFF during capture. If this option is not specified, all the pins driven by the reset signal in the input netlist are changed to be driven by the lpct_reset_out output of the controller. If this option is specified, only the pins explicitly specified are driven by the lpct_reset_out output of the controller.

  This option can only be used for the Type 3 LPCT controller. For more information, see "Type 3 LPCT Controller" in the *Tessent TestKompress User's Manual*.

- TEST_End

  Optional argument that specifies the pin is an end of test signal.

  This option can only be used for the Type 3 LPCT controller. For more information, see "Type 3 LPCT Controller in the *Tessent TestKompress User's Manual*.

- TEST_Active

  Optional argument that specifies an output signal from that LPCT controller that indicates the LPCT controller is active and controlling the scan test.

  This option is only valid for the Type 3 controller. For more information, see "Type 3 LPCT Controller in the *Tessent TestKompress User's Manual*.

- SHIFT_Dr

  Required argument that specifies the pin is a Shift_Dr signal on the TAP controller.

  This option can only be used for the Type 2 LPCT controller. For more information, see "Other LPCT Controller Types (Not Recommended)" in the *Tessent TestKompress User's Manual*.

  Note

  Streaming-Through-IJTAG Scan Data is recommended as a superior alternative to the Type 2 LPCT controller.

- CAPTURE_Dr

  Required argument that specifies the pin is a Capture_Dr signal on the TAP controller.

  This option can only be used for the Type 2 LPCT controller. For more information, see "Other LPCT Controller Types (Not Recommended) in the *Tessent TestKompress User's Manual*.

  Note

  Streaming-Through-IJTAG Scan Data is recommended as a superior alternative to the Type 2 LPCT controller.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Update_dr

  Required argument that specifies the pin is an Update_Dr signal on the TAP controller.

  This option can only be used for the Type 2 LPCT controller. For more information, see "Other LPCT Controller Types (Not Recommended)" in the *Tessent TestKompress User's Manual*.

  Note

  Streaming-Through-IJTAG Scan Data is recommended as a superior alternative to the Type 2 LPCT controller.

- TMs

  Required argument that specifies the pin is a test mode select tms input pin of the TAP.

  This option can only be used for the Type 2 LPCT controller. For more information, see "Other LPCT Controller Types (Not Recommended)" in the *Tessent TestKompress User's Manual*.

  Note

  Streaming-Through-IJTAG Scan Data is recommended as a superior alternative to the Type 2 LPCT controller.

- -Active {High|Low}

  Optional switch and literal pair that specifies the active state for the input reset signal. By default, the reset is active high.

  This option can only be used for the Type 3 LPCT controller. For more information, see "Type 3 LPCT Controller" in the *Tessent TestKompress User's Manual*.

- TEST_Clock_connection

  Optional switch that specifies the test clock input pin for the user-added test_mode mux. The tool connects the output of the clock gater, added to facilitate sharing of the LPCT clock and the scan clock, to the specified pin. The user must connect the other input of the mux to the functional clock and the select pin to the test mode signal.

Examples

Example 1

The following example defines the connections between the LPCT controller and the design core.

```
// Temporary top-level scan IO pins (removed by TK during IP creation

add_scan_chains chain1 grp1 scan_in1 scan_out1
add_scan_chains chain2 grp1 scan_in2 scan_out2

// LPCT controller input pins
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_lpct_pins clock ref_clk clk_pad/Z
set_lpct_pins reset_out - pwr_up/rst
set_lpct_pins data_in fnl_input

// Connections from LPCT logic outputs to design

set_lpct_pins output_scan_enable scan_en
set_lpct_controller -shift_control enable
set_lpct_pins shift_en clk_mux/shift_en
set_lpct_pins capture_en clk_mux/cap_en
```

Example 2

The following example defines the connections between the LPCT controller and the design core when the LPCT controller internally generates the scan_en signal.

```
set_lpct_controller -generate_scan_enable on -max_shift 1000 -max_capture 3
set_lpct_pins output_scan_enable scan_mode
```

Example 3

The following example defines the connections between the LPCT controller and the design core when a top-level scan_en signal exists.

```
set_lpct_controller on
set_lpct_pins input_scan_enable scan_en
```

Example 4

The following example defines the connections between the LPCT controller and the design core when a top-level scan_en signal exists and the user has implemented clock control.

```
set_lpct_controller -shift_control enable
set_lpct_pins capture_en pll/capture_start
set_lpct_pins clock_mux_select pll/select
set_lpct_pins shift_en pll/shift_en
```

Example 5

The following example defines the connections between the LPCT controller and the design core when a TAP generated scan_en signal exists.

```
set_lpct_controller -tap_controller on -generate_scan_enable on
set_lpct_pins output_scan_enable scan_mode
set_lpct_pins shift_dr - tap/shiftEn
set_lpct_pins capture_dr - tap/captureEn
set_lpct_pins update_dr - tap/updateEn
set_lpct_pins test_mode - tap/atpgInstruction
set_lpct_pins reset - tap/resetl -active low
```

Example 6

The following example defines the connections between the LPCT controller and design core when a TAP is used to generate scan_en signal

```
set_lpct_pins clock refclk -on_controller_module          // input pin
set_lpct_pins capture_en pll_start -on_controller_module // output pin
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 7

The following example specifies to use a top-level scan clock as the LPCT clock for a Type 1 controller.

```
set_context dft -edt
add_clocks 0 clk                    // There is a single clock in the design
add_scan_chains ...
set_lpct_controller on -shift_control clock
set_lpct_pin clock clk              // LPCT clock is shared with scan clock
set_lpct_pin input_scan_enable scan_en
set_lpct_pin test_clock_connection test_mode_mux/B
set_system_mode analysis
report_lpct_pins
write_edt_files created -replace
```

Example 8

The following example specifies to use tck as the scan shift clock in a Type 2 LPCT controller. Note, the test mode mux that selects between the LPCT-generated scan clock and the functional clock already exists in the design. The test clock connection pin on the mux is specified with the test_clock_connection pin type.

```
set_context dft -edt
add_clocks 0 tck
add_scan_chains ...
set_lpct_controller -tap_controller_interface on
set_lpct_controller -shift_control clock
set_lpct_pins output_scan_enable scan_en
set_lpct_pins clock tck pad_tck/Z      // LPCT clock is tck
set_lpct_pins tms tms pad_tms/Z
set_lpct_pins reset - tap_i/tlr
set_lpct_pins capture_dr - tap_i/capturedr
set_lpct_pins shift_dr - tap_i/shiftdr
set_lpct_pins update_dr - tap_i/updatedr
set_lpct_pins test_mode - tap_i/edt_scan_inst
set_lpct_pins test_clock_connection test_mode_mux/B
set_edt_options -channel 1
set_edt_pins input 1 tdi pad_tdi/Z
set_edt_pins output 1 tdo tap_i/tap_edt_channel_reg_in
set_system_mode analysis
report_lpct_pins
report_lpct_configuration
write_edt_files created -replace
```

Related Topics

report_lpct_configuration

report_lpct_pins

set_lpct_controller

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_lvx_options

Context: patterns -scan, patterns -retargeting

Mode: analysis

Configures Tessent LVX (Laser Voltage Imaging/Probing) settings.

Usage

set_lvx_options [-ssh_icl_instances *icl_instances*] [-sequence *string*]
[-repetitions {load_chains | *int*}] [-payload_loop_count {off | *int*}]
[-sync_pulse_port *port*] [-sync_pulse_cycles *cycle_count*]

Description

The set_lvx_options command provides optional configuration for creating lvx_loop and lvx_verification patterns.

Although the options set by this command affect both lvx_loop and lvx_verification patterns, the tool ignores the -repetitions, -sync_pulse_port, and -sync_pulse_cycles arguments when writing lvx_verification patterns.

Arguments

- -ssh_icl_instances *icl_instances*

  An optional switch and list of ICL instances that specifies a subset of the active SSHs to target. By default, LVX targets all active SSHs in the current mode.

- -sequence *string*

  An optional switch and string that specifies the LVX loop sequence to write into the pattern set. The default pattern is “1100”.

- -repetitions {load_chains | *int*}

  An optional switch and integer that specifies the minimum number of times to repeat the sequence in the payload. For tester patterns and a Verilog test bench the default value is 1. When you use the “load_chains” value, the tool automatically calculates a number of repetitions that is sufficient to generate activity on all EDTs. This is useful to observe design activity during simulation.

  Note

  The tool does not measure datapath output in the Verilog test bench. If you use the test bench, you must monitor values shifted into the internal scan chains.

  Note

  The “load_chains” option generates enough repetitions to observe at least some activity in all scan chains. However, the number of repetitions generated may not necessarily fill up the entire scan chain. If you need more simulation cycles, manually specify a larger number of repetitions.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -payload_loop_count {off | *int*}

  An optional switch and literal or integer pair that controls whether the LVX payload is in a loop with the specified count. The default value is "off", which does not use an explicit loop and loop count around the payload. This option is specifically intended for patterns that are intended to be run on the tester.

  Note

  The lvx_verification patterns support only a single application of the payload. If you generate lvx_verification patterns while this switch is set to an integer value, the tool disregards that value and uses a value of "off".

- -sync_pulse_port *port*

  An optional switch and port name pair that specifies a port to pulse at the start of each lvx_loop pattern set for LVP. Afterwards, the port is forced low for the remainder of the lvx_loop payload. Use the -sync_pulse_cycles switch to specify the duration of the pulse. If you do not specify -sync_pulse_cycles, it defaults to a pulse of one cycle.

  The pattern models this port as an additional scan chain, so it can be pulsed using NRZ timing in parallel with scan shifting. This results in no interruption in periodicity.

- -sync_pulse_cycles *cycle_count*

  An optional switch and integer pair, used in conjunction with -sync_pulse_port, that specifies the number of shift cycles to force the port high at the start of the pattern set. The length of the payload must be at least double the cycle_count in order to have equal numbers of pulses high and low. The number of repetitions is automatically increased to satisfy NRZ timing for the -sync_pulse_port port. If you do not specify -sync_pulse_port, either concurrently with this switch or in a separate set_lvx_options command, the tool reports a warning and ignores this setting.

Examples

The following example applies the default sequence, "1100", 200 times. For LVP, it forces the port GPIO9 high for the first four shift cycles of the lvx_loop pattern. This creates a wide pulse that is easier to detect. During LVP, you loop over the entire lvx_loop payload.

```
set_lvx_options -repetitions 200 -sync_pulse_port GPIO9 -sync_pulse_cycles 4
```

The following example applies the custom sequence 101010 on a specific SSH ICL instance. The lvx_verification pattern calculates the expected response to the custom sequence and the tool verifies the response at the SSN bus output ports.

```
set_lvx_options -ssh_icl_instances piccpu_rtl_tessent_ssn_scan_host_1_inst \
  -sequence 101010
```

Related Topics

get_lvx_option

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_macrotest_options

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Prepares a black box macro to enable MacroTest conversion of patterns at the boundaries of that instance.

Usage

set_macrotest_options **{OFf | ON}**

Description

The set_macrotest_options command changes two normal behaviors of the tool's DRC. To obtain a pulse from a macro, a PI of the circuit that has been declared as a clock, read control, or write control must drive the pin of the macro. If the black box is an empty instantiation of a RAM, it is likely that the tool needs to pulse the write control to write into the RAM. Normally, if a write control does not connect to a _ram or _cram primitive, DRC issues a G5 violation and does not take you out of Setup until the write control is deleted. The set_macrotest_options command prevents this and treats any write control as valid (as if it were connected to a _ram or _cram).

Another DRC aspect that the tool modifies occurs if a transparent latch (TLA) is on the control (input) side of a black box macro, and is unobservable due to the macro. Tessent FastScan converts a black box macro to a set of TieX gates, one per output. As a result, if the TLA only fans out to the macro, the tool cannot observe it (it has no sensitizable path to some PO or scan cell). Normally, DRC converts such a TLA to a TieX gate, preventing MacroTest from justifying values on the input side of the black box macro backward through the TLA. This is because it thinks the TieX means the primitive cannot be controlled. The set_macrotest_options command prevents this and keeps unobservable TLAs as TLAs.

If in doubt, issue the set_macrotest_options command in setup mode before proceeding with macro testing. No errors occur if DRC does not encounter the situations resolved by this command. Therefore, you can issue the set_macrotest_options command before every MacroTest run if you desire.

Arguments

- **{OFf| ON}**

  A required Boolean that specifies whether to modify DRC as described in the preceding description.

  OFf — Disables DRC modification (normal DRC is performed). This is the default at invocation.

  ON — Enables modified DRC analysis to avoid potential macrotest-specific ATPG issues.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example shows the command issued to enable black box macro testing:

```
set_macrotest_options on
```

Related Topics

macrotest

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_marker_file_options

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies settings for the marker file generated by the extract_fault_sites and extract_inter_cell_data commands.

Usage

set_marker_file_options
[-top_name *name*]
[-max_bridges {*max_bridges_value* | unlimited}]
[-max_opens {*max_opens_value* | unlimited}]
[-max_cell_pairs {*max_cell_pairs_value* | unlimited}]
[-max_defects {*max_defects_value* | unlimited}]

Description

This command defines the marker file settings for the extract_fault_sites and the extract_inter_cell_data commands.

The -top_name, -max_bridges, -max_opens, and -max_defects switches apply to the marker file generation of all subsequent extract_fault_sites commands. Typically, the generated marker file contains less bridges/opens than the generated UDFM file. The marker file generated for interconnect bridges and opens is always sorted such that the entries for the bridges/opens with the largest total critical area are written at the top of the file and the entries for the bridges/opens with the smallest total critical area at the end of the file.

For example, if you issue the “set_marker_file_options -max_bridges 20” command and option, the subsequent extract_fault_sites commands write the following to the marker file:

- The ten bridges with the largest total critical area.
- The ten bridges with the smallest total critical area.

As result any bridges with a mid-level total critical area in the generated UDFM file are excluded from the marker file.

The -max_cell_pairs switch applies to the marker file generation of all subsequent extract_inter_cell_data commands. The marker file generated for the cell neighborhood combinations is always sorted by the total bridging length (tbl).

For example, if you issue the “set_marker_file_options -max_cell_pairs 20” command and option, the subsequent extract_inter_cell_data commands write the 20 cell pairs with the largest tbl to the marker file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- -top_name *name*

  An optional switch and string that specifies the design name of the top module for the marker file.

- -max_bridges *max_bridges_value* | unlimited

  An optional switch and value or literal pair that specifies the maximum number of bridges to write to the marker file.

  The default value is 1000.

- -max_opens *max_opens_value* | unlimited

  An optional switch and value or literal pair that specifies the maximum number of opens to write to the marker file.

  The default value is 1000.

- -max_cell_pairs *max_cell_pairs_value* | unlimited

  An optional switch and value or literal pair that specifies the maximum number of cell pairs to write to the marker file when using the extract_inter_cell_data command.

  The default value is 1000.

- -max_defects *max_defects_value* | unlimited

  An optional switch and value or literal pair that specifies the maximum number of defects to write to the marker file. The *max_defects_value* includes all defects types that are specified with the -defect_types switch of the extract_fault_sites command.

  The default value is 1000.

Examples

Example 1

In this example, the extract_fault_sites command writes all interconnect bridges with a distance up to three times the technology length (default value) to a UDFM file called *bridges.udfm*. The command also generates a marker file called *bridges.marker* containing 100 bridges as specified by the preceding set_marker_file_options command. In the marker file, 50 of the bridges have the largest total critical area in the generated UDFM file. The other 50 bridges in the marker file have the smallest total critical area in the generated UDFM file. The top module name written to the marker file is *my_chip*.

```
set_marker_file_options -top_name my_chip -max_bridges 100

extract_fault_sites -output_file bridges.udfm -defect_types { bridges } \
                    -marker_file bridges.marker
```

Example 2

In this example, the extract_inter_cell_data command writes the 12 most important cell pairs to the output file named *top_12_cells.data*. The command also generates a marker file called

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

top_3.marker containing the entries related to the three most important cell pairs as specified by the preceding set_marker_file_options command.

```
set_marker_file_options -max_cell_pairs 3

extract_inter_cell_data -output_file top_12_cells.data -max_cell_pairs 12 \
                        -marker_file top_3.marker
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_memory_cluster_library_generation_options

Context: dft

Mode: setup

Configures the automation of Shared Bus physical to logical (P2L) mapping and logical memory TCD file generation.

Usage

set_memory_cluster_library_generation_options
{**-generate_physical_to_logical_info**
[-cluster_instances *cluster_instances*]
[-init_file_name *init_file_name*]
{[-output_directory *dir_path*]
| [-original_location]}
[-output_file_extension *extension*]
[-replace]}
| -off

Description

The set_memory_cluster_library_generation_options command enables automatic mapping of physical memories that are arranged within the logical memories of a shared bus memory cluster. The logical memory TCD file is modified to include the generated MemoryGroupAddressDecoding and PhysicalToLogicalMapping wrappers. These wrappers reference the appropriate physical memory TCD, define the selection codes, and incorporate the physical memory to logical memory interface mapping information.

The set_memory_cluster_library_generation_options command requires that the cluster and memories are within the current design, and the memory cluster TCD, physical memory TCDs, and logical memory TCDs without physical to logical information are loaded. Siemens EDA recommends that you run the mapping process on a current design containing only the memory cluster and its memories to retain the certification scope and help ensure a successful generation.

When set_memory_cluster_library_generation_options runs, a memory cluster initialization dofile named <*design_name*>_<*design_id*>.memory_cluster_mbist_mode_init is created in the current directory. Normally, the initialization file requires no modification. The initialization file automatically runs when the check_design_rules command executes, at which point the physical to logical mapping extraction and analysis occurs. For additional usage information, including how to modify the initialization file, refer to the "Shared Bus Learning" topic in *Tessent MemoryBIST User's Manual for use With Tessent Shell*.

By default, the generated logical memory TCD files retain the original file names and are written into the *memory_tcd_outdir* folder found in the current directory.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- -generate_physical_to_logical_info

  A required switch that enables the physical to logical memory mapping and generation of the logical memory TCD files.

- -cluster_instances *cluster_instances*

  An optional switch that specifies the cluster instance for which logical memory mapping and generation of the logical memory TCD files will be done. By default, the tool picks one instance of each cluster module present in the design. The tool reports the cluster instances on which mapping occurred when the tool transitions from the setup system mode to the analysis mode.

- -init_file_name *init_file_name*

  An optional switch and value pair that specifies the name of a memory cluster initialization file the tool will use. If you do not specify the switch, the tool will default to using an automatically generated initialization file. For further information about the memory cluster initialization file, refer to the "Enabling MemoryBIST Mode for a Cluster" topic in *Tessent MemoryBIST User's Manual for use With Tessent Shell*.

- -output_directory *dir_path*

  An optional switch and string pair that specifies the directory in which to write all the modified logical memory TCD files. By default, the tool will write the files into the automatically created *memory_tcd_outdir* folder, found in the current working directory. This switch is mutually exclusive with the -original_location switch.

- -original_location

  An optional switch that directs the tool to write all modified logical memory TCD files into the location where the original memory TCD files are stored. By default, the tool writes the files into the automatically created *memory_tcd_outdir* folder, found in the current working directory. This switch is mutually exclusive with the -output_directory switch. To avoid overwriting the original memory TCD files, you must also specify the -output_file_extension switch with a different file extension, otherwise the tool generates an error message. You can use the -replace switch to intentionally overwrite the existing files. Siemens EDA does not recommend overwriting original files and recommends keeping the original and modified files in separate locations.

- -output_file_extension *extension*

  An optional switch and string pair that specifies the extension to use when the tool writes the modified logical memory TCD files. By default, the tool writes the modified files with the same extension used by the original files.

- -replace

  An optional switch that directs the tool to overwrite any existing files without reporting an error message.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -off

  An optional switch that turns off a previously enabled physical to logical memory mapping and TCD generation. This switch is mutually exclusive with all other arguments.

Examples

If you want to change the suffix of the modified logical memory TCD filenames, use the "-output_file_extension" switch to replace the original extension with one you specify. For example, if the original file is called "Logical.tcd" and the following command is used:

```
set_memory_cluster_library_generation_options \
 -generate_physical_to_logical_info -output_file_extension modified
```

The output file will have a name of "Logical.modified".

If you want to append a suffix to the filename instead of replacing the original extension, using the following command will result in an output file named "Logical.tcd.modified":

```
set_memory_cluster_library_generation_options \
 -generate_physical_to_logical_info -output_file_extension tcd.modified
```

Related Topics

set_memory_cluster_validation_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_memory_cluster_validation_options

Context: dft

Mode: setup

Enables Shared Bus memory cluster library validation.

Usage

set_memory_cluster_validation_options
{***name_patterns***
[-regexp] [-nocase] [-logical_to_interface_only]
[-cluster_instances *cluster_instances*]
[-init_file_name *init_file_name*]} |
[-off]

Description

The set_memory_cluster_validation_options command configures and enables a tracing-based validation of the specified memory cluster TCD and logical memory TCDs against the structure of the memory cluster in the current design. The validation verifies the following:

- No memory is missing memory BIST coverage.
- Mappings and connectivity specified in the LogicalMemoryToInterfaceMapping and PhysicalToLogicalMapping TCD wrappers are correct.
- That the specified PipelineDepth property for the logical memories is correct.

When you run set_memory_cluster_validation_options, the tool creates a memory cluster initialization dofile named *<design_name>_<design_id>.memory_cluster_mbist_mode_init* in the current directory. Normally, the initialization file requires no modification. The initialization file is automatically run when the check_design_rules command is executed, at which point the validation of the Shared Bus libraries and circuit analysis occurs.

For further usage information, refer to the "Shared Bus Learning" topic in *Tessent MemoryBIST User's Manual for use With Tessent Shell*.

Arguments

- ***name_patterns***

  A required string or Tcl list of strings that specifies one or more patterns the tool uses to identify memory module names. An error is issued if *name_patterns* is not specified.

- -regexp

  An optional switch that directs the tool to interpret the *name_patterns* argument as a regular expression rather than a simple wildcard pattern. Refer to the "Glob and Regular Expression Pattern Matching Syntax" section for a description of the regular expression syntax.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when searching for modules matching the specified *name_pattern*.

- -logical_to_interface_only

  An optional switch that directs the tool to validate mappings extracted from the design against the memory cluster TCD LogicalMemoryToInterfaceMapping wrappers only. This option is useful when performing early validation, for example, prior to physical RAM being integrated in the design within the logical memory and PhysicalToLogicalMapping wrappers are not present in the logical memory TCD. For this usage, the *name_patterns* argument should correspond to the design modules associated with logical memories activated by individual selection codes specified in the memory cluster TCD.

- -cluster_instances *cluster_instances*

  An optional switch and value pair that specifies the cluster instance or instances to validate. The *cluster_instances* value can be an object name, a Tcl list of object names, or a collection of objects. By default, the tool will validate one instance of each cluster module present in the design and reports the cluster instance being validated.

- -init_file_name *init_file_name*

  An optional switch and value pair that specifies the name of a memory cluster initialization file the tool will use. If you do not specify the switch, the tool will default to using an automatically generated initialization file. For further information about the memory cluster initialization file, refer to the "Enabling MemoryBIST Mode for a Cluster" topic in *Tessent MemoryBIST User's Manual for use With Tessent Shell*.

- -off

  An optional switch that turns off module matching validation that was previously enabled with a set_memory_cluster_validation_options command and a valid *name_patterns* expression. This switch is mutually exclusive with all other arguments

Examples

Example 1

Assume the following memory modules exist in a design and are connected to a Shared Bus Interface:

```
SYNC_1R1W_16x8, SYNC_1R1W_32x4, RAM_32x4, RAM_128x128
```

If you want to identify them as memory modules, use the following command to match the memory modules and perform a memory cluster validation:

```
set_memory_cluster_validation_options {SYNC.* RAM.*} -regexp
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

Assume the following logical memory modules exist in a design and are connected to a Shared Bus Interface:

```
LM_0, LM_1, LM_2, LM_3
```

If you want to identify them as memory modules, use the following command to match the memory modules and perform a memory cluster validation at the logical level:

```
set_memory_cluster_validation_options {LM.*} -regexp \
-logical_to_interface_only
```

Each module, which is considered as a logical memory, is required to be activated by a unique selection code from the memory cluster TCD. Refer to the "Memory Cluster TCD Validation at the Logical Level" topic in *Tessent MemoryBIST User's Manual for use With Tessent Shell* for further information.

Related Topics

set_memory_cluster_library_generation_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_memory_identification_options

Context: dft

Mode: setup

Sets options and pattern expressions to validate memory TCD are associated with memory modules to be tested by MemoryBIST.

Usage

set_memory_identification_options {***name_patterns*** [-regexp] [-nocase]} | [-off]

Description

Sets regular expression or wildcard naming patterns to identify memory module names in the design technology library. When check_design_rules is run, Tessent MemoryBIST uses this information to identify the memories instantiated in the design that have not been declared as memories and are not tested by MemoryBIST. This condition can be caused by a missing memory TCD for a design's memory module. When a memory library file is read in, the tool automatically associates the memory TCD to the design module and declares it as a memory to be tested by MemoryBIST.

The options specific to the set_memory_identificaton_options command are used by the check_design_rules command to ensure that all modules that should be declared as memories are properly recognized after a successful system mode transition. An error is issued if a design module matches the provided *name_patterns* argument, but no associated memory TCD is found. A warning is issued if there are design modules recognized as memories, but they do not match the *name_patterns* argument. In cases where errors or warnings are issued, the tool automatically displays a tabular report from the validation. The validation report can always be displayed with the report_memory_identification command.

Arguments

- ***name_patterns***

  A required string or Tcl list of strings that specifies one or more patterns to be used to identify memories in the design. A non-empty string must be provided or an error is issued.

- -regexp

  An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a regular expression instead of a simple wildcard pattern. Refer to the "Glob and Regular Expression Pattern Matching Syntax" section for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when searching for modules matching the *name_patterns*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -off

  An optional switch that is mutually exclusive with all other arguments. Specifying the -off switch eliminates module matching validation that was previously enabled with a set_memory_identification_options command executed with a valid *name_patterns* expression.

Examples

Assume the following memory modules exist in a design and it is desired to identify them as memory modules:

```
SYNC_1R1W_16x8, SYNC_1R1W_32x4, RAM_32x4, RAM_128x128
```

The following use of set_memory_identification_options can be used to match these memory modules:

set_memory_identification_options {SYNC.* RAM.*} -regexp

Related Topics

report_memory_identification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_memory_instance_options

Context: dft

Mode: setup

Prerequisite: The current design must be set with the set_current_design command.

Set memory instance options for design rules and design specifications.

Usage

set_memory_instance_options {***memory_instances***
[-bist_data_in_pipelining {off | on | auto | *int*}]
[-physical_cluster_override *string*]
[-test_clock_override *domain_label*]
[-repair_sharing {off | on | auto}]
[-use_in_memory_bist_dft_specification {off | on | auto}]
[-use_in_memory_bisr_dft_specification {off | on | auto}]
[-partitioning_group *group_label*]} |
[-physical_cluster_size_ratio *int*] |
[-generate_external_repair_logic {off | on}]

Description

Sets options on or for memory instances to be used by the check_design_rules and create_dft_specification command, when the set_dft_specification_requirements command was used to enable the memory BIST and or memory BISR feature.

This command is used to set options on memory instances as returned by the get_memory_instances command. Most options can be set differently for each memory instance. The -physical_cluster_size_ratio option applies to all instances.

Arguments

- ***memory_instances***

  A string that contains a Tcl list of one or more names of memory instances, or a collection of one or more memory instances. The memory instances are the instances on which the tcd_memory_name Instance attribute is defined as a result of a TCD Memory description having been matched to its module name—see "Tessent Core Description" on page 4793. You can use the get_memory_instances command to get a collection of all memory instances.

  Tip

  If your ***memory_instances*** string contains a space, see "Escaped Hierarchical Names in Command Tcl Lists" in the *Tessent Shell User's Guide* for details on how to specify it correctly.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -bist_data_in_pipelining {auto | on | off | *int*}

  An optional switch and value pair that specifies the number of pipeline stages to be added on all BIST signals from the controller to the memory. This affects the memory BIST partitioning since only memories with equal data-in pipelining can be tested in the same controller step. The value off is equivalent to 0 and the value on is equivalent to 1. When the value is set to auto, the value specified in the DefaultsSpecification MemoryInterfaceOptions/bist_data_in_pipelining property is applied to the specified memory instances.

- -physical_cluster_override *string*

  An optional switch and value pair that associates one or more memory instances to a given cluster ID. The string may contain any number of letters, numbers, and underscores; it cannot start with the word "def" and be followed by an integer because those strings are reserved for the clusters computed from the DEF file. If you read a DEF file using the read_def command, physical clusters are created for all memory instances for which a physical cluster override string was not specified. See the description of the -physical_cluster_size_ratio option below for more information about the clustering process. The automatic cluster generation computed from DEF is performed when running the check_design_rules commands.

- -test_clock_override *domain_label*

  An optional switch and value pair that associates one or more memory instances to a clock domain label. By default, the clock domain label associated with the memory is one of the clock domains connected to the clock ports of the memory. When the clocks have different frequencies, the fastest one is selected. You can use this option when:

  - the memory instance has no clocks (asynchronous memory)
  - to force the selection of a given clock when multiple clocks have the same frequency
  - to select a slower clock from the available clocks

  Clock ports receiving a clock that is different than the selected clock domain have a multiplexer inserted on it to inject a common clock during memory BIST.

- -repair_sharing {off | on | auto}

  An optional switch and literal pair that specifies whether the memory instance has memory repair sharing enabled. By default, the value of "auto" uses the global enable value specified in the DefaultsSpecification/DftSpecification/MemoryBist/RepairOptions/repair_sharing property. A value of "auto" also uses the global repair group scope specified by the repair_group_scope property in the DefaultsSpecification/DftSpecification MemoryInterfaceOptions or MemoryClusterOptions wrappers. Specifying "off" or "on" overrides the global values specified in the DefaultsSpecification for the memory instance.

- -use_in_memory_bist_dft_specification {off | on | auto}

  An optional switch and literal pair that specifies whether the memory instance is to be considered by DRC and as a part of the DftSpecification/MemoryBist wrapper that the create_dft_specification command creates. When set to auto, the memory instance is

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

considered if it is not seen as already connected to a memory BIST interface module. If it is already connected to a memory BIST interface module, it is excluded from the new DftSpecification/MemoryBist wrapper.

This feature is useful when doing multi-pass insertion in the case where you initially added memory BIST for a set of memories and then, later, edited the RTL and inserted new memories. When you specify the default value of auto, re-running the memory BIST insertion process on this RTL only considers the new memory. You can force the exclusion of a memory instance even if it is not already connected to a memory BIST interface module by specifying the off value.

- -use_in_memory_bisr_dft_specification {off | on | auto}

  An optional switch and literal pair that specifies whether the memory instance is to be part of the DftSpecification/MemoryBisr wrapper that the create_dft_specification command creates. When set to auto, the memory instance is considered if it is not seen as already connected to a memory BISR register module. If it is already connected to a memory BISR register module, it is excluded from the new DftSpecification/MemoryBisr wrapper.

  This feature is useful when doing multi-pass insertion in the case where you initially added memory BISR for a set of memories and then, later, edited the RTL and inserted new repairable memories. When you specify the default value of auto, re-running the memory BISR insertion process on this RTL only considers the new repairable memory. You can force the exclusion of a memory instance even if it is not already connected to a memory BISR register module by specifying the off value.

- -partitioning_group *group_label*

  An optional switch and string value pair that associates a custom partitioning group label for one or more memories. Memories associated to different *group_label* partitioning groups do not share the same controller. Memories in the same partitioning group may be assigned to separate controllers based on other MemoryBIST Partitioning Rules. When this option is unspecified, the memory belongs to the default unnamed group.

- -physical_cluster_size_ratio *int*

  An optional switch and value pair that specifies the parameter used to create memory clusters for Memory BIST partitioning. If you read a DEF file using the read_def command, memory clusters are created and memories not belonging to the same cluster are assigned to different memory BIST controllers when the create_dft_specification command creates the DftSpecification/MemoryBist wrapper.

  A rectangle is created to fully enclose the current design and the length of its diagonal is extracted. The ratio is the percentage of a cluster's diameter with respect to this diagonal distance. A memory instance is added to a given cluster if it does not cause the diameter of the cluster to exceed the threshold ratio. Otherwise a new cluster is created. The default ratio is 20 percent but you can change this percentage value by specifying an integer between 0 and 100 using the -physical_cluster_size_ratio option. A value of 0 causes each memory instance to be in its own cluster, and a value of 100 results in all memory instances being in the same (single) cluster.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -generate_external_repair_logic {off | on}

  An optional switch and value pair that specifies that external IO repair logic is to be added to the specified memory instances when set to on. The default value is off. When set to on, the Step/MemoryInterface(*id*)/generate_external_repair_logic property is automatically populated and set to on for the associated memory interface when the create_dft_specification command is run. For further information, refer to "External Repair" in *Tessent MemoryBIST User's Manual for use With Tessent Shell*.

Examples

Example 1

The following example sets the number of data in the pipelining stages to 3 for the memory instances found below instance u1/u2. It also sets the ***-physical_cluster_size_ratio option to*** 30 percent.

```
set_memory_instance_option [get_memory_instances -below_instance u1/u2] \
    -bist_data_in_pipelining 3

set_memory_instance_option -physical_cluster_size_ratio 30
```

Related Topics

get_memory_instances

get_memory_instance_option

report_ijtag_instances

set_dft_specification_requirements

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_misr_connections

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Describes the connections between the scan chain outputs and the MISR.

Usage

set_misr_connections ***misr_name*** **-tap** ***integer*** **-chain** ***chain_name***

Description

Use the set_misr_connections command with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

If you map multiple scan chains to the same MISR position, then the tool infers a XOR spatial compactor that maps from those multiple scan chain outputs to the specified MISR feedback position.

Arguments

- ***misr_name***

  A required string that specifies the name of the MISR specified with the add_misrs command.

- **-tap** ***integer***

  A required switch and required and repeatable integer that specifies the tap points for the MISR feedback network. The tap index starts from 1, to be consistent with the decompressor commands.

- **-chain** ***chain_name***

  A required switch and string that specifies the name of the scan chain pin that you want to connect to the MISR specified by ***misr_name***.

Examples

This example describes the hardware connection from scan chain outputs to MISR inputs through a 4-to-1 XOR compactor.

```
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[0] -tap 1
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[1] -tap 1
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[2] -tap 1
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[3] -tap 1
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[4] -tap 2
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[5] -tap 2
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[6] -tap 2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[7] -tap 2
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[8] -tap 3
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[9] -tap 3
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[10] -tap 3
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[11] -tap 3
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[12] -tap 4
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[13] -tap 4
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[14] -tap 4
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[15] -tap 4
```

Related Topics

add_misrs

delete_misrs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_module_matching_options

Context: all contexts

Mode: all modes

Defines the acceptable prefixes and suffixes or regular expressions to use when matching an ICL or a TCD module to a design module within ICL extraction.

Usage

set_module_matching_options [-prefix_pattern_list *prefix_pattern_list*]
[-suffix_pattern_list *suffix_pattern_list*]
[-module_name_map_list *list_forming_mapping_pairs*]
[-regexp] [-nocase] [-append]
[-format {all | icl | tcd_bscan | tcd_scan | tcd_fusebox | tcd_memory | tcd_memory_cluster}]
[-excluded_modules_from_module_matching *excluded_modules_from_module_matching*]

Description

The module matching is performed in those situations as follows:

1. With a set_current_design command call. This forces the tool to perform a complete elaboration, which resets other settings such as already defined black boxes.
2. With a set_context command call, and the design is already elaborated.
3. With an extract_icl command call.

Note

Defining new settings with the set_module_matching_options command does not trigger a new module matching. A module matching is triggered by the above mentioned situations and uses the settings defined at that point of time.

During module matching, the list of all design modules found below the current design is used to search for corresponding ICL and TCD modules. The ICL and TCD modules are then attached to the design modules.

Note

An ICL or TCD description of the top module deactivates the module matching in most cases. In this case, only ICL elaboration and related tasks are executed, but no module matching is performed.

If the ICL or TCD modules refer to the RTL names of the design modules and module matching is done on the synthesized view, the design modules that are used more than once are likely to have been uniquified with an integer suffix. You may also have instructed the synthesis tool to prefix every synthesized module with a common prefix. The default suffix list handles the unification suffix inserted by synthesis automatically. If you instructed the synthesis tool to

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

prefix every synthesized module with a common prefix, you need to describe the used prefix using the -prefix_pattern_list switch.

See the instrument_name, tcd_bscan_name, tcd_fusebox_name, tcd_memory_cluster_name, and tcd_memory_name Instance attribute descriptions. Those attributes are set on the design instances and modules when they are matched to an ICL or TCD module. See also the get_ijtag_instances and get_memory_instances command descriptions.

Arguments

- -prefix_pattern_list *prefix_pattern_list*

  An optional switch and string pair that specifies the prefix patterns taken into account when matching ICL or TCD modules to design modules. The *prefix_pattern_list* is a Tcl list of prefix strings or regular expressions to be used as a prefix to the ICL or TCD module name when matching it against the design module names. The *prefix_pattern_list* value is always augmented with the specified "set_insertion_options -edited_module_prefix" value list when matching a module of any format.

  By default, the tool matches any module name with a prefix that matches the "*ts_graybox_" regular expression. The tool also extends user-specified prefixes with this prefix. This means that the tool matches all names of the format <*any_name*>ts_graybox_<*module_name*>. The additional and default prefixes enable the automatic recognition of Tessent Shell-generated graybox modules using module matching.

  Note

  By default, the "set_insertion_options -edited_module_prefix" command and switch inserts an underscore character between the prefix and the original module name. When using the -prefix_pattern_list switch, you must explicitly specify this underscore.

- -suffix_pattern_list *suffix_pattern_list*

  An optional switch and string pair that specifies the suffix strings or regular expressions to be used for uniquified design module mapping. This switch can only be used once. The *suffix_pattern_list* string or Tcl list of strings lists one or more patterns to be used for uniquified design module mapping. By default, the tool matches any module name with a suffix that matches the regular expression "{[_]+[0-9]+}". This means that, by default, the tool matches all names of the format <module_name>_<number>.

- -module_name_map_list *list_forming_mapping_pairs*

  An optional switch and string pair that specifies module name mapping where the -prefix_pattern_list and -suffix_pattern_list switches do not suffice. The *list_forming_mapping_pairs* string lists the module name and the uniquified module name in pairs. You can match the same module name to different uniquifications by repeating the original module name with different pairs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -regexp

  An optional switch that directs the matching patterns as a regular expression instead of as a simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" on page 3834 for a description of the regular expression syntax.

- -nocase

  An optional switch that forces the pattern matching to be case insensitive. The presence of the -nocase switch in the most recent set_module_matching_options command determines the behavior of the module matching. If the -nocase switch is used in the most recent set_module_matching_options command, then the pattern matching is case insensitive. If the -nocase switch is omitted, the case sensitivity depends on the hardware description language (Verilog or VHDL) that has been used to describe the design module.

- -append

  An optional switch that specifies to append the newly specified pattern list to the existing list or to replace the existing list.

- -format {all | icl | tcd_bscan | tcd_scan | tcd_fusebox | tcd_memory | tcd_memory_cluster}

  A switch and literal pair used in conjunction with the -excluded_modules_from_module_matching switch that defines modules not to be considered during module matching for the specific format or all formats. Choose one from the following:

  all

  icl

  tcd_bscan

  tcd_fusebox

  tcd_memory

  tcd_memory_cluster

  tcd_scan

  The default is "all", which means the tool does not consider the excluded modules during module matching in all of the formats above.

  Refer to the "Tessent Core Description" on page 4793 for more information on these TCD macro module types.

- -excluded_modules_from_module_matching *excluded_modules_from_module_matching*

  A switch and string pair that specifies a module collection to exclude from module matching for the format you specify with the -format switch or for all formats. Specifying a new collection for a specific format deletes previously defined exclude modules for that format.

Return Values

None

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

In the following example, this command is used to enable matching ICL modules with an optional prefix “mycore_” and an optional suffix matching _ts[0-9]+. In this example, an ICL module InstrumentA matches mycore_InstrumentA_ts1 and mycore_InstrumentA_ts2.

```
// default
get_module_matching_options -prefix_pattern_list


*ts_graybox_

set_module_matching_options -prefix_pattern_list {mycore_} \
  -suffix_pattern_list {_ts[0-9]+} -regexp
get_module_matching_options -prefix_pattern_list


*ts_graybox_  mycore_  .*ts_graybox_mycore_
```

Example 2

The following example demonstrates using the -module_name_map_list switch to map module names, with some modules repeated for multiple uniquifications of the original name:

```
set_module_matching_options \
        -module_name_map_list {cpu_a cpu_1_a cpu_a cpu_2_a \
                gpu_x gpu_1_x gpu_x gpu_2_x gpu_x gpu_3_x gpu_x gpu_4_x}
```

Example 3

The following example demonstrates using the -format and -excluded_modules_from_module_matching switches:

```
set_module_matching_options -format icl \
         -excluded_modules_from_module_matching $mods_to_ignore
```

Related Topics

get_module_matching_options

report_module_matching

report_module_matching_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_multiple_detection

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables the multiple detection of faults.

Usage

set_multiple_detection
[-Guaranteed_atpg_detections *guaranteed_atpg_detections*]
[-Desired_atpg_detections *desired_atpg_detections*]
[-Simulation_drop_limit *simulation_drop_limit*]
[-REPort_incremental_statistics {ON | OFF}] [OFF]

UDFM Usage:

set_multiple_detection
[-Simulation_drop_limit *simulation_drop_limit*] [OFF]

Description

Multiple detection is a statistical way to gain bridge fault coverage. By default, the tool generates a single detection pattern set, where a fault is detected once, then dropped from the simulation.

When multiple detection is enabled, the DS fault category represents faults detected one or more times, not just faults detected the threshold number of times. The create_patterns command includes a cumulative Bridge Coverage Estimate (BCE) in the statistics transcription for each batch of patterns. Use the *simulation_drop_limit* argument to increase the accuracy of the BCE, being aware that the improved accuracy entails more run time.

The BCE, which is based on simulation results, is calculated as follows:

$$\mathrm{BCE} = \left[\sum_{i=1}^{n} \frac{f_i}{TE} \cdot (1 - 2^{-i}) \right] + \frac{N_{\mathrm{DI}}}{TE}$$

where:

f_i = number of faults detected *i* times by the test pattern set

TE = number of total testable faults in the target fault list

N_{DI} = number of DI faults in the target fault list*n* = *atpg_detections* (or *fault_drop_detections* if used)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Enabling multiple detections changes the behavior of the following commands:

- analyze_compression — If the -Preserve_faults_from_analysis switch is specified, preserves the N-detection information for all the detected faults during the analyze_compression operation.
- create_patterns — Performs multiple iterations (loops) as needed to ensure every testable fault is detected *guaranteed_atpg_detections* times and includes a cumulative bridge coverage estimate (BCE) in the statistics transcription for each batch of patterns simulated. Each pattern counts at most only one detection for any particular fault.
- report_statistics — Lists the final BCE in reports and additionally includes a histogram that shows the profile of the number of detections for multiple detection DS faults.
- report_environment — Lists a "multiple detection = …" line that shows the current multiple detection settings. If this line is absent, it means multiple detection is disabled.
- report_faults — Displays a "num_detections..." line for each DS fault that shows the number of detections for that fault.
- write_faults — Writes a "num_detections..." line for each multiple detection DS fault that shows the number of detections for that fault.

Issue this command without any arguments to report the current setting. If you issue this command with arguments more than once, the last command overwrites the previous settings.

Note that when the fault type is UDFM, the set_multiple_detection command works only for simulation and not for ATPG.

Arguments

- -Guaranteed_atpg_detections *guaranteed_atpg_detections*

  An optional switch and integer pair that specifies the number of detections required for each testable fault. The *guaranteed_atpg_detections* can be any integer between 1 and 255 and determines the number of detections for each fault that the create_patterns command should target during pattern generation. The specified number of detections for each fault are guaranteed during fault simulation before the tool drops the fault from the fault list.

  The default value of *guaranteed_atpg_detections* is 1. Increasing the number of guaranteed detections can significantly increase the pattern count and ATPG run time.

- -Desired_atpg_detections *desired_atpg_detections*

  An optional switch and integer pair that specifies the number of detections desired for each testable fault. These faults are detected using Embedded Multi-Detect (EMD) functionality. The *desired_atpg_detections* can be any integer between 1 and 255.

  Setting *desired_atpg_detections* to 1 disables EMD.

  By default, *desired_atpg_detections* is equal to *guaranteed_atpg_detections*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The *desired_atpg_detections* cannot be less than the *guaranteed_atpg_detections* except when EMD is disabled (*desired_atpg_detections* set to 1).

Unlike the number of guaranteed detections, increasing the number of desired detections does not increase the pattern count. However, it may increase the ATPG run time. For optimal performance, it is not recommended to set the *desired_atpg_detections* to greater than 5.

- -Simulation_drop_limit *simulation_drop_limit*

  An optional switch and integer pair that specifies the maximum number of detections for a fault. The tool drops the fault from the simulation after the *simulation_drop_limit*.

  When the fault type is UDFM, you can set the *simulation_drop_limit* to any positive integer. By default, the *simulation_drop_limit* is 1.

  For all other fault types, the *simulation_drop_limit* cannot be less than the *guaranteed_atpg_detections* or *desired_atpg_detections* values. If you do not specify a value, the *simulation_drop_limit* is automatically set to one of the following:

  - **guaranteed_atpg_detections = desired_atpg_detections = 1**

    The *simulation_drop_limit* is set to 1.

  - **guaranteed_atpg_detections <= 10** AND **2 <= desired_atpg_detections <= 10**

    The *simulation_drop_limit* is set to 10.

  - **guaranteed_atpg_detections > 10** OR **desired_atpg_detections > 10**

    The *simulation_drop_limit* is set to the greater of the *guaranteed_atpg_detections* or *desired_atpg_detections* values.

- -REPort_incremental_statistics {ON | OFF}

  An optional switch and literal pair that enables the tool to report incremental multiple-detect test coverage statistics at the end of each multiple-detect ATPG loop. The default is for the tool to not report such statistics.

- OFF

  An optional literal that disables the multiple detection of faults.

Examples

Example 1

The following example generates a 3-detection pattern set without EMD, where a fault is dropped from the simulation after 5 detections:

```
set_multiple_detection -guaranteed_atpg_detections 3 -desired_atpg_detections 1
        -simulation_drop_limit 5
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example generates a pattern set where each testable fault must be detected twice and should be detected at least 8 times:

```
set_multiple_detection -guaranteed_atpg_detections 2 -desired_atpg_detections 8
```

The simulation drop limit is automatically set to 10.

Example 3

The following example disables the multiple detection of faults:

```
set_multiple_detection off
```

Related Topics

create_patterns

read_faults

report_environment

report_faults

report_statistics

set_fault_type

write_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_multiprocessing_options

Context: unspecified, dft -edt, patterns -scan

Mode: setup, analysis, insertion

Sets the values of several variables that affect multiprocessing commands for ATPG or simulation.

Usage

set_multiprocessing_options [-generic_delete *string*] [-generic_scheduler *string*]
[-license_timeout {*integer* | unlimited | no_queue}] [-lsf_heuristics {on | off}]
[-lsf_learning {on | off}] [-lsf_options *string*] [-multithreading {on | off}]
[-optimize_secondaries { pattern_count | runtime | balanced }]
[-processors_per_grid_request *integer*] [-remote_shell {rsh | ssh}]
[-result_time_limit *integer*] [-scheduler_timeout {*integer* | unlimited}]
[-sge_options *string*]

Description

Note

Take care to exclude any unsupported platforms that exist in your grid environment by specifying appropriate grid job submission constraints. For a list of supported platforms, refer to "Supported Hardware and Operating Systems" in *Managing Tessent Software*.

For more information about setting up for multiprocessing, refer to "Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual* and "The Tessent Tcl Interface" in the *Tessent Shell User's Manual*.

Arguments

- -generic_delete *string*

  An optional switch and string that specify a command to delete a secondary process. You create a secondary process via the generic job scheduler when you issue commands such as "add_processors generic". The string consists of a command that the tool uses to delete a job with the generic job scheduler. It also contains a placeholder which has the format %x, where "x" is a single-word string. The tool looks for the single-word string in the output of the command specified for the generic jobs submission via the -generic_scheduler switch and expects to find a number following the string that represents the job number used by the generic scheduling system. The command specified by the -generic_scheduler switch is issued when each secondary process is added, and the number parsed by the tool is then stored and substituted for the placeholder in the generic_delete commands issued to the generic scheduling system to delete each secondary process.

  If you specify, for example:

```
set_multiprocessing_options -generic_delete \
  { $SGE_ROOT/bin/lx24-x86/qdel %job }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Then running the command specified by the -generic_schedule switch should produce output that contains a job number following the word “job”, such as:

```
Your job 32335 ("tessenttask") has been submitted
```

The tool parses the number after “job” in the output and substitutes it for the placeholder %job, when it deletes the secondary process. In this case, the tool forms the command:

```
$SGE_ROOT/bin/lx24-x86/qdel 32335
```

The placeholder can be any word, as long as it matches the word that is output prior to the job number in the output of the command specified by the -generic_scheduler switch. If that word is “task” instead of “job” as in the following:

```
Your task 33435 ("tessenttask") has been submitted
```

Then you would specify a generic_delete command containing %task as in the following:

```
set_processing_options -generic_delete \
  { $SGE_ROOT/bin/lx24-x86/qdel %task }
```

The tool parses the number after ‘task’ in the output and substitutes it for the placeholder %task, when it deletes the secondary process. In this case, the tool forms the command:

```
$SGE_ROOT/bin/lx24-x86/qdel 33435
```

Please note that the string that the tool looks for in the output is case sensitive. For example, if the scheduler command output was:

```
Task 3343 (“tessenttask”) has been submitted
```

The command specified by the -generic_delete switch would have to contain the string “%Task” rather than “%task”.

The tool uses this command to delete jobs when they are no longer needed, such as when you issue a delete_processors command, when the tool shuts down, or when the scheduler exceeds its timeout setting or is interrupted by Ctrl-C. This explicitly notifies the generic scheduling system that the job can be terminated or has already completed. If you use the generic scheduler without specifying a generic_delete command, the tool simply tells the running secondary processes to terminate and leave it up to the generic scheduling system to detect the termination of the secondary processes.

- -generic_scheduler *string*

  An optional switch and string that specifies a command script. The script requests a secondary process using the generic job scheduler. (The generic job scheduler is used for grid systems other than LSF or SGE). The *string* value replaces the current value. Setting a correct value for the string requires a knowledge of the scheduler configuration at your site and the job control submission syntax. An incorrect specification can prevent the correct operation of the add_processors command.

  The specified script applies only to subsequent add_processors commands, not currently running processes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

To specify the number of processors per request for a generic grid scheduler, use the -processors_per_grid_request switch.

- -license_timeout {*integer* | unlimited | no_queue }

  An optional switch and non-negative integer that specifies the number of minutes the tool waits when attempting to acquire a license. The default value is 5 minutes.

  integer— A non-negative integer that specifies the queuing time in minutes. A zero value causes the tool to not wait for a license, the same behavior as specifying "no_queue". The tool reports when it is unable to acquire a license in the specified amount of time.

  unlimited — An optional string that specifies no time limit. The tool waits until a license becomes available.

  no_queue —An optional string that specifies that the tool not wait for a license and reports when a license is not available immediately.

- -lsf_heuristics {on | off}

  An optional switch and keyword that enables or disables LSF heuristics learning when you add a processor by LSF grid request. The tool uses LSF heuristics learning in an attempt to automatically choose machines on the LSF grid that are compatible with the tool if you did not specify an LSF "type" constraint with -lsf_options and if either LSF scheduler learning has failed or you disabled that feature using "-lsf_learning off."

  LSF heuristics learning may be able to determine tool compatibility based on the names of the types and models available and successfully identify some type/model combinations that the tool can use; however, this may also result in specifying incompatible combinations or miss some compatible combinations. If LSF heuristics learning is turned off or fails, then the tool submits the job anyway without specifying any type/model restrictions.

  LSF heuristics learning is enabled by default. For more information about using this switch, refer to "Processor Addition to the LSF Grid" in the in the *Tessent Scan and ATPG User's Manual*.

- -lsf_learning {on | off}

  An optional switch and keyword that enables or disables LSF scheduler learning when you add a processor by LSF grid request. When this switch is on and you have not specified an LSF "type" constraint with -lsf_options, the tool asks the LSF system what machines and type/model combinations are available, after which the tool runs the lsrun command to log onto likely machines to determine which are compatible with the tool.

  LSF scheduler learning cannot succeed if your LSF system is configured to prevent use of lsrun. If LSF scheduler learning is turned off or fails, the tool then uses LSF heuristics learning unless you have disabled this feature with "-lsf_heuristics off."

  LSF scheduler learning is enabled by default. For more information about using this switch, refer to "Processor Addition to the LSF Grid" in the *Tessent Scan and ATPG User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -lsf_options *string*

  An optional switch and string that specifies command options for the tool to append to the job submission command for the LSF job scheduler. The string value you set for this switch replaces the current value. Setting a correct value for the string requires a knowledge of the scheduler configuration at your site and the job control submission syntax. Applies only to subsequent add_processors commands, not to currently running processes. An incorrect specification can prevent the correct operation of the add_processors command.

  When you add the "verbatim" key symbol at the beginning of the string, the tool interprets the rest of the line as the submission specification for each grid request it makes, overriding any memory and swap availability requirements it would otherwise attempt to add (see the note for the add_processors ***hostname*** argument). When you use this keyword, be sure to include any needed memory and swap requirements in the rest of the -lsf_options string. For more information, refer to "Example 8" on page 2751.

- -multithreading {on | off}

  An optional switch and keyword that disables multithreading functionality. Multithreading is enabled by default.

- -optimize_secondaries { pattern_count | runtime | balanced }

  An optional switch and keyword that specifies an optimization when using additional secondary processes.

  pattern_count — Focuses on maintaining consistent pattern count across multiple tool runs when the design and tool settings do not change. Pattern count remains the same regardless of the number of CPUs or the distribution of those processors between the primary and secondary processes. Scalability of distributed processing may be more limited at higher processor counts than with the other two options. This is the default.

  runtime — Focuses on speed of pattern generation. However, it can result in more patterns.

  Note

  Pattern generation will run with the *-compaction_effort* switch set to *low* to further optimize run time unless this switch is changed when creating patterns.

  balanced — A compromise between pattern count and run time. Tool optimizations attempt to keep pattern count within ten percent of pattern_count. Run time performance improves with additional secondary processes.

  Without multiple secondary processes, all options behave the same as pattern_count. To benefit from improved scalability with either the runtime or balanced options, you must use multiple secondary processes, not just additional local threads on the primary process.

  If you add processors to the primary host with multithreading enabled, the primary process increases its thread count instead of adding additional secondaries. For more information, refer to "Multiprocessing for ATPG and Simulation" in the *Tessent Scan and ATPG User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -processors_per_grid_request *integer*

  An optional switch and integer value that specifies the maximum number of processors (slots) to group into one request for the grid system, either for SGE, LSF, or generic requests. The default value is "unspecified" (-1) because there are different default values for SGE, LSF, and generic. The default for LSF requests is 8. The tool can use LSF job submission switches to ensure that this number of processors is requested per grid request. For SGE and generic processor requests, the syntax for requesting a specific number of processor slots per grid request is site-specific, so the tool defaults to one processor per request.

  If you want to specify the number of processors per request for an SGE or a generic grid scheduler, use the -sge_options or -generic_scheduler switches. You can do this either by hardcoding a value in the -sge_options or -generic_scheduler string or by using the pseudo-variable *%processors_per_grid_request*.

  The pseudo-variable enables you to change the processors_per_grid_request value later without changing the corresponding grid command variable value. When using the pseudo-variable, the default value is 8 for SGE, LSF, and generic requests. For an example using the variable, refer to "Example 3" on page 2749.

  The requested number of processors (for example, add_processors sge:20) results in the following number of grid requests: the requested number of processors divided by processors_per_grid_request. The result of the division is rounded up, and the last grid request can request fewer processors than the number of "processors_per_grid_request."

  Note

  The use of job schedulers requires certain environment prerequisites to be satisfied before you invoke the tool. For more information, refer to "Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual*.

- -remote_shell {rsh | ssh}

  An optional switch and keyword that specifies the shell command that the primary host should use to create processes on secondary hosts. Applies only when you specify hosts manually rather than using a job scheduler. By default, the tool uses the rsh command.

- -result_time_limit *integer*

  An optional switch that specifies a time limit in minutes for detecting non-responsive secondary processes while waiting for certain types of job results. If the tool detects that the time limit has been exceeded for a specific result needed to make further progress, the tool issues a warning about the unresponsive secondary process and waits again for the same result. (This gives you the opportunity to investigate and correct the problem.) If the result_time_limit is again exceeded, the tool kills the secondary process and reassigns the job to another processor, and the cycle starts over. In most cases, you do not have to change the default value. Note that if you set the value too low, the tool could possibly kill all of its secondary processes needlessly. A value of 0 disables the time limit, but otherwise the command does not accept a value of less than 15. The default value is 45.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Messages similar to the following in your log file may indicate that processes are being killed as a result of a time limit that is too low:

```
//  Warning: Secondary process 1 has not returned a result in a long time.
//           If this process continues to be non-responsive it will be killed.
//           The timing used to determine non-responsiveness can be adjusted
//           with "set_multiprocessing_options -result_time_limit _value_".

//  Warning: Terminating non-responsive process 1 (<hostname>)  [<timestamp>]
//           The timing used to determine non-responsiveness can be adjusted
//           with "set_multiprocessing_options -result_time_limit _value_".
//           3 secondary processes running, 1 process dying
```

Note

In certain cases involving extremely complex and high-effort ATPG, the tool automatically increases the secondary process timeout limit. The limit increase is proportionate based on the longest-running jobs that have completed on other hosts. After increasing the limit, the tool continues to monitor progress on the processes.

- -scheduler_timeout {*integer* | unlimited}

  An optional switch and non-negative integer value or the string “unlimited” that specifies a time limit, in minutes, for the tool to attempt to add processors using a job scheduler. If the tool does not receive all the requested processors within the specified time, it proceeds with the processors it has received.

  integer— A non-negative integer that specifies the time limit, in minutes, for the tool to attempt to add processors using a job scheduler. A zero value indicates the same as specifying the “unlimited” string. The default value is 10 minutes.

  unlimited — An optional string that specifies that the tool waits until all requested processors are received from the grid.

- -sge_options *string*

  An optional switch and string that specifies command options for the tool to append to the job submission command for the SGE or OGE job scheduler. The *string* value replaces the current value. Setting a correct value for the string requires a knowledge of the scheduler configuration at your site and the job control submission syntax. An incorrect specification can prevent the correct operation of the add_processors command.

  The specified script applies only to subsequent add_processors commands, not currently running processes. To specify the number of processors per request, use the -processors_per_grid_request switch.

Command Variables

The following variables are available for use in this command. You can define these variables in your dofile for convenience.

%command — Represents a substring you use with the -generic_scheduler switch to specify the location to substitute the Tessent command script that launches a secondary process. For more information, refer to “Example 5” on page 2750.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

%job — A placeholder, rather than a variable. For more information on this placeholder, refer to the description of -generic_delete *string*.

%processors_per_grid_request — Specifies the maximum number of processors (slots) to group into one single request for the grid system, either for SGE, LSF, or generic requests. When using this pseudo-variable, the default value is 8 for SGE, LSF, and generic requests. For more information, refer to "Example 3" on page 2749.

Tcl Notes

When using Tcl constructs with the set_multiprocessing_options command, the tool's logfile reports the evaluated command and strips any Tcl special characters (quotes, braces, and brackets) from the logfile. If you need to replay a session from the logfile, you must add the Tcl syntax.

Enclosing any expression in braces ({ }) prevents the tool from evaluating the expression. Enclosing any expression in double quotation marks (" ") causes the tool to resolve any variables in the expression.

Examples

Example 1

The following example controls type and memory constraints that would normally be supplied automatically:

```
set_multiprocessing_options -lsf_options { -R \
    select[type==LINUX64 && mem> 200 && swp > 200 ] }
get_multiprocessing_option -lsf_options
-R select[type==LINUX64 && mem> 200 && swp > 200]
```

This setting would constrain LSF to select hosts meeting the specified requirements. The "type" specification also inhibits the automatic learning that would normally take place to discover all the type/model names in an LSF cluster. (This is useful if your site prevents the use of the lsrun command or if the learning hits non-responsive hosts.) You can specify any legal LSF constraints for the installation, which overrides any related specifications the tool would have provided otherwise. It is up to you to supply syntactically and semantically correct LSF options.

Example 2

The following example specifies that LSF use queue *myqueue*:

```
set_multiprocessing_options -lsf_options "-q myqueue"
```

Example 3

The following example specifies that LSF reserve *%processors_per_grid_request* processors or slots:

```
set_multiprocessing_options -lsf_options "-n  %processors_per_grid_request"
```

Note, however, that the -n specification would probably be only a subset of the bsub options assigned to the -lsf_options variable, and that the above example command line serves only as an example of how to use the *%processors_per_grid_request* variable. If you don't use the -n

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

switch in the -lsf_options string, the tool still specifies -n using the processors_per_grid_request default value to the LSF job scheduler. Note than when requesting additional processors per grid request, the number of grid requests decreases accordingly.

Example 4

The following example specifies that SGE impose a 6 hour time limit on secondary processes:

set_multiprocessing_options -sge_options {-l h_rt=6:00:00}

Example 5

The following example shows how to set up a generic_scheduler command that uses SGE to request additional processors:

**set_multiprocessing_options -generic_scheduler **
{ $SGE_ROOT/bin/lx*-amd64/qsub *%command* }

%command is a substring you must include as part of the -generic_scheduler switch to indicate where the tool should place the path to the command script that is used to launch additional processors. Variables can be defined in the dofile to make this more convenient, as in the following example dofile excerpt:

```
set SUBMIT { $SGE_ROOT/bin/lx*-amd64/qsub }
set_multiprocessing_options -generic_scheduler "${SUBMIT} %command"
```

Example 6

The following example shows how to use an alternative interface to SGE for terminating secondary processes:

add_processors generic

set_multiprocessing_options -generic_delete { $SGE_ROOT/bin/lx24-x86/qdel *%job* }

The *%job* is a placeholder that specifies a word, in this case “job”, that the tool uses to locate the job number in the generic scheduler’s output. The job number is substituted for the placeholder and used with the command $SGE_ROOT/bin/lx24-x86/qdel when deleting processors.

The place holder does not need to be *%job*. It is whatever was used when the secondary process was started. This is a variation of this example using a different placeholder, in this case, *%task*:

add_processors generic

set_multiprocessing_options -generic_delete { $SGE_ROOT/bin/lx24-x86/qdel *%task* }

Example 7

The following example specifies that the primary host should use the ssh shell command to create processes on secondary hosts (manual specification of hosts only):

set_multiprocessing_options -remote_shell ssh

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tip

If you encounter errors trying to use SSH, refer to "SSH Environment and Passphrase Errors" in the *Tessent Scan and ATPG User's Manual*.

Example 8

The following example specifies that LSF job requests should specify the use of hosts with type LINUX64 and should not specify any additional job constraints:

```
set_multiprocessing_options -lsf_options  { verbatim -R "select[ type==LINUX64 ]" }
```

Related Topics

add_processors

delete_processors

get_multiprocessing_option

report_multiprocessing_options

report_processors

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_net_dominance

Context: all contexts

Mode: setup, analysis (dft -edt and patterns -scan contexts only)

Specifies the fault effect of bus contention on tri-state nets.

Usage

set_net_dominance **Wire** | **And** | **Or**

Description

The set_net_dominance command specifies the fault effect of bus contention on tri-state nets. This provides the capability to detect some faults on tri-state driver enable lines when those drivers connect to a tri-state bus. These faults would be ATPG-untestable unless the tool can use the Z-state for detection.

When using Tessent FastScan or Tessent TestKompress, the Wire behavior is the same where any different binary value results in an X-state.

The truth tables for each type of bus contention fault effect are given in Tables 8-14, 8-15, and 8-16. This command does not affect Good machine behavior.

Table 8-14. WIRE Bus Contention Truth Table

	X	**0**	**1**	**Z**
X	X	X	X	X
0	X	0	X	0
1	X	X	1	1
Z	X	0	1	Z

Table 8-15. AND Bus Contention Truth Table

	X	**0**	**1**	**Z**
X	X	0	X	X
0	0	0	0	0
1	X	0	1	1
Z	X	0	1	Z

Table 8-16. OR Bus Contention Truth Table

	X	**0**	**1**	**Z**
X	X	X	1	X
0	X	0	1	0

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-16. OR Bus Contention Truth Table (cont.)

	X	0	1	Z
1	1	1	1	1
Z	X	0	1	Z

Arguments

- **Wire**

  A literal that specifies for the tool to use unknown behavior for the fault effect of bus contention on tri-state nets. This is the default behavior upon invocation of the tool.

- **And**

  A literal that specifies for the tool to use wired-AND behavior for the fault effect of bus contention on tri-state nets.

- **Or**

  A literal that specifies for the tool to use wired-OR behavior for the fault effect of bus contention on tri-state nets.

Examples

The following Tessent FastScan example specifies that the fault effect on tri-state nets is wired-AND during ATPG:

```
set_net_dominance and
set_system_mode analysis
create_patterns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_net_resolution

Context: all contexts

Mode: setup

Specifies the behavior of multi-driver nets.

Usage

set_net_resolution **Wire** | **And** | **Or**

Description

The set_net_resolution command specifies the behavior of non-tri-state, multi-driver nets. The default upon invocation of the tool is Wire, which requires all inputs be at the same value to achieve a value. If you can model your nets using the And or Or option, you can improve your test coverage results.

Arguments

- **Wire**

  A literal that specifies for the tool to use unknown behavior for non-tri-state, multi-driver nets. This requires all inputs to be at the same value to achieve a value other than X. This is the default upon invocation of the tool.

  If you are using Tessent FastScan or Tessent TestKompress with E9 rule checking enabled, the tools do not perform checking on wire gates if you use the set_net_resolution command to change their behavior to AND or OR.

- **And**

  A literal that specifies for the tool to use wired-AND behavior.

- **Or**

  A literal that specifies for the tool to use wired-OR behavior.

Examples

The following Tessent FastScan example specifies that the behavior of non-tri-state, multi-driver nets is wired-AND during ATPG:

```
set_net_resolution and
set_system_mode analysis
create_patterns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_number_shifts

Context: dft -edt, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Sets the number of shifts for loading or unloading the scan chains.

Usage

set_number_shifts ***shift_number***

Description

Note

For diagnosis, you can only use the set_number_shifts command to modify the number of shifts for an EDT design. There is no support in the diagnosis tool for non-EDT designs.

By default, the number of shifts used for loading or unloading the scan chains in a scan group is the same as the largest number of scan cells in any scan chain in that scan group. The tool determines this number once it traces all scan chains. However, you can use the set_number_shifts command to increase the default number.

You normally issue this command in setup mode. If the specified number of cycles is more than the number determined during DRC, the tool adds the extra cycles as overshift cycles. If the specified number of cycles is less than required, the tool issues an error.

If EDT is enabled, you can issue this command in either setup or analysis mode. The number of shift cycles specified in analysis mode does not impact the pattern generation process, which uses the number determined during DRC. The generated external patterns are padded with the additional cycles when they are saved using the "write_patterns -edt_external" command.

For edt_internal patterns, you must use the parameter file keyword ALL_EDT_INTERNAL_USE_NUMBER_SHIFTS in order to apply the number of shift cycles specified by set_number_shifts. For more information, refer to "ALL_EDT_INTERNAL_USE_NUMBER_SHIFTS" on page 5006.

Here are some considerations for using the set_number_shifts command:

- The set_number_shifts command is mutually exclusive with the "set_edt_options -OVERShift_cycles *integer*" command. If you attempt to issue both commands in the same session, the tool issues an error and resets the number of shifts to zero.
- The set_number_shifts command does not affect parallel testbenches.
- The set_number_shifts command has no effect on WGL patterns if both the WGL_FULL_CHAIN and WGL_FULL_SCANGROUP parameter file keywords are set to 0. By default, WGL_FULL_CHAIN is set to 1 and WGL_FULL_SCANGROUP is set to 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***shift_number***

  Specifies the number of shifts. If the number specified is smaller than the default number determined by the tool, the tool issues an error message.

Examples

Example 1

The following example sets the number of shifts for loading or unloading the scan chains to 2.

```
set_number_shifts 2
```

Example 2

The following example adjusts the number of shifts for a given pattern set and runs diagnosis:

```
set_number_shifts 100
read_patterns my_patterns/pat.wgl
diagnose_failures flogs/fail_1.log
```

Related Topics

set_edt_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_observation_point

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the observation point for random pattern fault simulation.

Usage

set_observation_point **Master** | **SLave** | **SHadow** | **Clockpo**

Description

The set_observation_point command specifies whether Tessent FastScan observes master latches, slave latches, shadow latches, or clock primary outputs during random pattern fault simulation. If you select Master, Slave, or Shadow, Tessent FastScan also observes the primary outputs that do not connect to clock lines. The default behavior upon invocation of Tessent FastScan is Master.

Arguments

- **Master**

  A literal that specifies observation of master latches and normal primary outputs. This is the default behavior upon invocation of Tessent FastScan.

- **SLave**

  A literal that specifies observation of slave latches and normal primary outputs.

- **SHadow**

  A literal that specifies observation of observable shadow latches and normal primary outputs.

- **Clockpo**

  A literal that specifies observation of only primary outputs directly connected to clocks.

Examples

The following Tessent FastScan example specifies slave latches as the observation point for random pattern fault simulation:

```
set_system_mode analysis
set_observation_point slave
add_faults -all
simulate_patterns -source random
```

Related Topics

set_capture_clock

set_random_patterns

simulate_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_output_masks

Context: dft, dft -edt, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Ignores any fault effects that propagate to the primary output pins you select.

Usage

set_output_masks **OFf** | **ON** [-Bidi_exclude] [-Exclude *pin_pathname*…]

Description

The tool uses primary output pins as the observe points during the fault detection process. When you mask a primary output pin, you inform the tool to mark that pin as an invalid observation point during the fault detection process. This command provides the ability to flag all or a portion of the primary output pins that do not have strobe capability (are not observable). The tool classifies the faults whose effects only propagate to that observation point as Atpg_Untestable (AU).

You can use set_output_masks in setup mode or analysis mode, but not both. The tool behaves as follows:

- When output masks are enabled in setup mode and you disable them in analysis mode, the tool reports, “Error: Output masks enabled in setup mode cannot be disabled in analysis mode.”
- When output masks are enabled in setup mode and you enable them in analysis mode, the tool reports, “Note: Output masking was already enabled in setup mode.”
- When output masks are enabled in analysis mode and you return to setup mode, the tool disables the output masks.

You can use the report_output_masks command to display the current setting of the output masks.

Arguments

- **OFf**

  A required literal that specifies for the tool to use primary output pins as the observe points during the fault detection process. In addition, the tool gives possible credit for fault effects that reach a pin that the pattern set specifies as being at an unknown value. This is the default behavior upon invocation.

- **ON**

  A required literal that specifies for the tool to mask the simulated primary output pin values with Xs for all patterns. As a result, the tool does not give any credit for fault effects that reach those masked primary output.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-**Bidi_exclude

  An optional switch that modifies how the command treats bidirectional pins.

  Note

  The meaning of the -Bidi_exclude switch is different for each setting (“off” or “on”) of the command. It basically specifies to do the opposite, for bidirectional pins, of what is done for other pins.

  When used with the Off literal, the -Bidi_exclude switch specifies for the tool to set_output_masks “on” for bidirectional pins. Thus, the command “set_output_masks off -bidi_exclude” results in the tool masking bidirectional pins and no others.

  Conversely, if you use the On literal, then adding the -Bidi_exclude switch causes the tool to set_output_masks “off” for bidirectional pins. The command “set_output_masks on -bidi_exclude”, therefore, results in the tool masking all pins except bidirectional pins.

- **-**Exclude *pin_pathname…*

  An optional switch and repeatable string that modifies how the command treats a specific pin.

  Note

  Similar to the -Bidi_exclude switch, the -Exclude switch specifies for the tool to do the opposite, for each specified pin, of what is done for other pins.

  When used with the Off literal, the -Exclude switch causes the tool to set_output_masks “on”, but only for the specified pin(s). The command “set_output_masks off -exclude my_pin”, for example, results in the tool masking the pin whose pathname is “my_pin” and not masking any others.

  Conversely, when used with the On literal, the -Exclude switch causes the tool to set_output_masks “off” for the specified pin(s). In this case, “set_output_masks on -exclude my_pin” applies masking to all pins except the pin “my_pin”.

Examples

The following example reports all the primary outputs of a design having one primary output, /y_inout, that is bidirectional. It then specifies to mask all the primary outputs except any that are bidirectional, and the three pins, y_out[3], y_out[2], and x_out[1]. The example then displays the masked primary output pins. As you would expect, none of the pins excluded from masking are listed.

report_primary_outputs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
SYSTEM:  /x_out[3]
SYSTEM:  /x_out[2]
SYSTEM:  /x_out[1]
SYSTEM:  /x_out[0]
SYSTEM:  /y_out[3]
SYSTEM:  /y_out[2]
SYSTEM:  /y_out[1]
SYSTEM:  /y_out[0]
SYSTEM:  /my_out
SYSTEM:  /y_inout
```

set_output_masks on -bidi_exclude -exclude y_out[3] y_out[2] x_out[1]

report_output_masks

```
TIEX   /x_out[3]
TIEX   /x_out[2]
TIEX   /x_out[0]
TIEX   /y_out[1]
TIEX   /y_out[0]
TIEX   /my_out
```

Related Topics

add_output_masks

delete_output_masks

report_environment

report_output_masks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_parallel_load_subchains

Context: dft -edt, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the internal parallel access functionality is enabled or disabled.

Usage

set_parallel_load_subchains [OFf | ON]

Description

By default, internal parallel access is disabled. In this case, when a netlist contains a subchain in a library model, the parallel testbench can only apply serial shifts to load the scan data and compare the capture values because internal pin names (where the parallel value is forced and observed) of the library models cannot be accessed. This limitation can increase parallel simulation runtime and significantly increase the size of the parallel testbench file.

If this command is enabled in Setup mode, the parallel testbench is written to access internal pin names of the library models. When you exit Setup mode, a DRC warning is issued if the netlist contains a subchain. In this case, you should enable this command. You can use the "report_scan_chains -subchains" command to display detailed information about the subchains in the netlist.

Arguments

- OFf

  An optional literal that disables internal parallel access. This is the default.

- ON

  An optional literal that enables internal parallel access. This command can be disabled or re-enabled in analysis mode to write out different parallel testbenches if necessary, but only if it was initially enabled in Setup mode. If this command was not enabled in Setup mode, you cannot change the setting in analysis mode because internal access points can only be learned during DRC when switching from Setup mode to a non-Setup mode.

  Note

  When this command is enabled, extra memory is required for storing the internal names of the library cells. Also, internal pin names of the Tessent Cell library models must match the internal pin names of the Verilog library model; otherwise, testbench compile errors or incorrect results from a Verilog simulation can occur. If you used LibComp to translate the Verilog libraries to ATPG libraries, then internal pin names should match because LibComp preserves these internal names.

Examples

The following example enables internal parallel access in Setup mode, sets the system mode to ATPG, creates test patterns, and saves the test patterns with parallel loading of subchains.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_parallel_load_subchains on
set_system_mode analysis
create_patterns
write_patterns trans_pat.gz -verilog -parallel
```

If the testbench is not compiled because the internal pin names of the Verilog library do not match those of the Tessent Cell library, parallel loading can be disabled. Then the testbench loads the subchain cells serially.

```
set_parallel_load_subchains off
write_patterns trans_pat2.gz -verilog -parallel
```

Related Topics

report_environment

report_scan_chains

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_pathdelay_holdpi

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the ATPG keeps non-clock primary inputs at a constant state after the first force.

Usage

set_pathdelay_holdpi **OFf** | **ON**

Description

The set_pathdelay_holdpi command lets you specify for the tools to ignore non-clock primary input changes after the first force in each pattern.

Arguments

- **OFf**

  A literal specifying that the tool can change non-clock primary input values at any time. This is the default behavior upon invocation of the tool.

- **ON**

  A literal that specifies for the tool not to change non-clock primary input values after the launch of the transition into the path.

Examples

The following example enables Tessent FastScan to ignore changes to non-clock primary inputs:

```
set_pathdelay_holdpi on
```

Related Topics

set_transition_holdpi

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_pattern_buffer

Context: unspecified, dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Stores run-time pattern data in temporary files in the directory specified.

Usage

set_pattern_buffer {***directory_name...***} | -**Off**

Description

The set_pattern_buffer command enables you to specify a directory where the tool can store temporary run-time pattern data. When the tool accesses a pattern, it is then automatically retrieved from the specified directory, as opposed to virtual memory, conserving the memory for other tool processing tasks. This provides a run-time processing advantage when there is not ample swap space.

You can specify multiple directories. As individual buffer files approach their file size limit or volume space runs out, the tool searches the specified directory list for available space. The tool deletes these buffer files when you exit the application.

Note

Buffer files may be multiple gigabytes large. Choose directories on volumes that have enough disk space to handle expected amount of pattern data.

Arguments

- ***directory_name***

  A required repeatable string that specifies the full path directory names in which the tool saves buffer files, thus enabling the usage of temporary buffer files for pattern data. Re-issuing this command with additional directories causes the tool to add the directories to the current list of directories available for buffer files.

- -**Off**

  A required switch that disables the use of temporary buffer files, causing the tool to save run-time data in virtual memory. This is the default upon invocation. Disabling the usage of pattern buffers can only be done when there is no pattern data, such as after issuing the reset_state or set_system_mode commands.

Examples

The following example places buffer files in the specified directories.

```
set_pattern_buffer /usr/jdoe/buffers /usr2/jdoe/morebuffers
```

Related Topics

report_resources

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_pattern_classification

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the order in which to store test patterns.

Usage

set_pattern_classification **ON** | **OFf**

Description

When set to "on," each pattern is stored by its classification, according to the following ordered list:

1. basic
2. read only (a type of basic pattern)
3. clock sequential and multiple load

When set to "off," test patterns are stored in the order in which they were generated. Note that when simulating the external patterns (by issuing the simulate_patterns command), the tool automatically turns off pattern classification.

Arguments

- **ON**

  A literal that enables pattern classification. If the current internal pattern set is not classified, the pattern is automatically classified after the command is executed.

- **OFf**

  A literal that turns off pattern classification. When pattern classification is disabled, patterns are stored in the same order in which they were generated.

Examples

The following example stores patterns in the order in which the tool generates them:

```
set_pattern_classification off
```

Related Topics

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_pattern_filtering

Context: patterns -scan, patterns -scan_diagnosis

Mode: analysis

Creates a temporary set of sampled scan patterns.

Usage

set_pattern_filtering [-EXternal] {**-Off**
| [{-PATtern_type *keyword*}… | {-NOPAttern_type *keyword*}…]
[{-RANge *begin_int end_int*}…] [{-NORange *begin_int end_int*}…]
[-SAMple_per_type *integer*]
[-CAPture | -CLOCK_Sequences]
[{-NAmed_capture_procedures {*procedure_name* ... | ALL}}
| {{[-CLocks *clock_name*…] [-NO_CLocks *clock_name* …]}
| [-RESTRICT_CLocks *clock_name*…]}]
[-LIst *pattern_index*…]
[{-CAPTURE_Power *expression* [ST_Peak | ST_Average | WSA_Peak |
WSA_Average] [-INSTance *instance_name*...] [-MODUle *module_name*...]}...]
[{[-SHIFT_Power {LOAD | RESPonse} *expression*] [-INSTance *object_name*...]
[-MODUle *module_name*...]}...]}

Description

Saves a sample of patterns for each pattern type instead of saving all the patterns. You can use this for serial pattern simulation to shorten the simulation time. In Tessent FastScan, the different types are those reported by the report_statistics command. In Tessent TestKompress, the types are additionally based on whether or not the tool used masking in a pattern. For example, Tessent TestKompress considers a clock-sequential pattern with scan chain masking a different type than a clock-sequential pattern without scan chain masking.

You can save sampled patterns in all supported formats with the write_patterns command. To restore the original non-sampled patterns, use the “set_pattern_filtering -off” command and option. If the content of the original pattern set changes while pattern filtering is on, pattern filtering is disabled automatically.

By default, this command preserves the original pattern order. When you use the -list switch, the user-specified pattern order is enforced. Also, the filtering and ordering only applies to scan (ATPG) patterns, and is not performed for chain test patterns.

Note

Because MacroTest patterns are order dependent, they are sampled sequentially, beginning at the first pattern. Non-MacroTest patterns are re-ordered.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- -EXternal

  An optional switch that applies sampling to external patterns. Before using this switch, you must specify the name of the external pattern file with the read_patterns command. The tool also re-orders the patterns.

- -Off

  An optional switch that restores the original non-sampled patterns.

- **-**PATtern_type *keyword*

  An optional switch and keyword pair that applies sampling based on the keyword pattern type. You can use multiple "-Pattern_type keyword" pairs; however, you cannot combine this switch with the -Nopattern_type switch in the same command. Keywords are listed in Table 8-17. If you specify neither this switch nor the -Nopattern_type switch, the default is to sample all pattern types.

  Valid pattern type keywords are:

Table 8-17. Keywords for Pattern Types

Pattern Type	Keyword
Basic	BAsic
Clock Sequential	CLOCK_Sequential
MacroTest	MacroTest
Multiple load	MUltiple_load

- **-**NOPattern_type *keyword*

  An optional switch and keyword pair that specifies to not sample the keyword pattern type. You can use multiple "-Nopattern_type keyword" pairs; however, you cannot combine this switch with the -Pattern_type switch in the same command. Keywords are listed in Table 8-17.

- -RANge *beg_int end_int*

  An optional, repeatable switch and pair of integers that creates a sample of patterns beginning with pattern beg_int and ending with pattern end_int. Patterns are numbered beginning with 0; for example, to restrict sampling to the first 100 patterns, use "-range 0 99."

- -NORange *beg_int end_int*

  An optional, repeatable switch and pair of integers that specifies to not apply sampling to the range of patterns beginning with pattern beg_int and ending with pattern end_int. Patterns are numbered beginning with 0; for example, to exclude patterns 51 through 60 from sampling, use "-norange 51 60." You can use this switch to exclude sub-ranges within sampled ranges specified with the -Range switch.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -SAMple_per_type *integer*

  An optional switch and positive integer pair that determines how many patterns of the specified filtering to sample.The default is 1.

- -CAPture

  An optional switch that samples the patterns by capture clock rather than by pattern type. It samples patterns based upon the capture clock of each capture cycle so that each clock pulse in different capture cycles is sampled <*n*> times, where <*n*> is selected by the "-sample_per_type <*n*>" option (default is 1). See Example 9.

  If you specify the -Capture switch with the -Pattern_ type switch, the command only samples patterns by capture clock within the specified pattern types.

- -CLOCK_Sequences

  An optional switch that samples the patterns by unique clock sequences rather than by pattern type. Each clock sequence is sampled <*n*> times, where <*n*> is selected by the "-sample_per_type <*n*>" option (default is 1).

- -NAmed_capture_procedures *procedure_name...* | ALL

  An optional switch and repeatable string or literal pair that samples patterns for either the specified named capture procedures or all named capture procedures.

  The number of patterns sampled depends on the number of samples specified using the -sample switch. If the -sample switch is not used, all the patterns that use the specified named capture procedures are sampled.

- -CLocks *clock_name…*

  An optional switch and repeatable string pair that samples only patterns that use one or more of the specified clocks. This includes patterns that use unspecified clocks if they also use at least one of the specified clocks. For example, "set_pattern_filtering -clocks clk1 clk2 clk3" samples a pattern that pulses clk1, clk2, or clk3. Patterns that do not pulse any clock are not sampled.

  You cannot combine this switch with the -Restrict_clocks switch in the same command.

- -NO_CLocks *clock_name…*

  An optional switch and optional, repeatable, string pair that samples only patterns that do not use any of the specified clocks. To obtain a sample that includes all the patterns that do not pulse any clock, omit the *clock_name* argument.

  Note

  If you issue "set_pattern_filtering -no_clocks clk1 clk2," where clk1 and clk2 are the only clocks pulsed by any pattern, the filtering result is the same as omitting the clock names and simply issuing "set_pattern_filtering -no_clocks."

  You cannot combine this switch with the -Restrict_clocks switch in the same command. If you omit the *clock_name* argument, you cannot combine this switch with the -Clocks switch in the same command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -RESTRICT_CLocks *clock_name…*

  An optional switch and repeatable string pair that samples only patterns that use the specified clocks, or a subset of these clocks, exclusively. Patterns that use any unspecified clocks are not sampled, even if they also use some or all of the clocks specified. For example, "set_pattern_filtering -restrict_clocks clk1 clk2 clk3" does not sample a pattern that pulses clk1, clk2 and clk4. Patterns that do not pulse any clock are not sampled.

  You cannot combine this switch with the -Clocks or -No_clocks switch in the same command.

- -LIst *pattern_index…*

  An optional switch and repeatable, non-negative, integer pattern indexes that creates a sample consisting of patterns with the specified pattern indexes, and orders the sampled patterns in the same order the indexes appear (left to right) in the command. The *pattern_index* refers to original pattern indexes in the pattern set before filtering by the set_pattern_filtering command. This switch cannot be combined with any other pattern filtering options.

  This switch enables you to arrange patterns in any order. For example, during IDDQ test, if the current of every pattern is measured to sort them in order of increasing current, the result is a list of pattern indices (for example, 13, 2, 24, and so on) that does not match any of the tool's built-in ordering criteria. You could use this switch to order the patterns according to the IDDQ sort.

- -CAPTURE_Power

  Optional switch that filters test patterns based on switching activity during capture.

- ***expression***

  Optional literal expression that contains two literals, an operator followed by a percentage: { <*operator*> <*percentage*> }. The expression must be supplied as a Tcl list. Use braces ({}) to indicate a list.

  <*operator*> — Optional literal that specifies an operation for test pattern filtering based on switching activity during shift or capture. The operator can be one of the following:

  - < — Less than
  - > — Greater than
  - <= — Less than or equal to
  - >= — Greater than or equal to

  <*percentage*> — Optional float number that specifies a switching threshold percentage that applies to filtering test patterns based on capture or shift power. The specified float number must be positive.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ST_Peak | ST_Average | WSA_Peak | WSA_Average

  Optional literal that filters the test patterns based on a specified metric. These options can only be used for switching activity during capture. Literals include:

  ST_Peak — Peak of state transitions. Default.

  ST_Average — Average of state transitions.

  WSA_Peak — Peak of weighted switching activity.

  WSA_Average — Average of weighted switching activity.

  Weighted Switching Activity is defined in the *Tessent Scan and ATPG User's Manual*.

- -INSTance *instance_name*...

  An optional switch and repeatable string that specifies one or more target instance(s) for test pattern sampling. This option is only available for sampling local switching activity during shift and/or capture. The instance_pathname may include any number of asterisk (*) and/or question mark (?) wildcard characters in a name.

- -MODULe *module_name*...

  An optional switch and repeatable string that specifies one or more target module(s) for test pattern sampling. This option is only available for sampling local switching activity during shift and/or capture. The module_pathname may include any number of asterisk (*) and/or question mark (?) wildcard characters in a name.

- -SHIFT_Power {LOAD | RESPonse}

  An optional switch and literal that filters test patterns based on switching activity during shift. Literal options include:

  LOAD — filters based on switching during test pattern loading.

  RESPonse — filters based on switching during the circuit response to test patterns.

Examples

Example 1

This example assumes ATPG has just been run, resulting in 125 patterns of three different types, including clock sequential. There are two capture clocks. The example first saves a sample consisting of four clock sequential patterns to a separate pattern file in TITDL format. The example then samples two patterns for each pattern type. Next, the same command is repeated with the addition of -Capture. This causes the tool to sample two patterns per type as before, and then add as necessary, additional patterns to ensure the sampled pattern set contains at least two samples per capture clock.

report_pattern_filtering

```
// Pattern filtering is off.
```

set_pattern_filtering -pattern_type clock_sequential -sample_per_type 4

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Pattern filtering is on: 1 pattern type(s),
    4 selected pattern(s), 125 original pattern(s).
```

write_patterns clock_sequential.tdl -titdl -replace

```
// Note: pattern filtering is on, #-saved-pattern(s) = 4,
    #-total-pattern(s) = 125
```

set_pattern_filtering -sample_per_type 2

```
// WARNING: The previous filter option has been reset.
// Internal pattern filtering is on: 3 pattern type(s),
   6 selected pattern(s), 125 original pattern(s).
```

set_pattern_filtering -capture -sample_per_type 2

```
// WARNING: The previous filter option has been reset.
// Internal pattern filtering is on: 3 pattern type(s),
   7 selected pattern(s), 125 original pattern(s).
```

report_pattern_filtering

```
   Current pattern filtering configuration
-----------------------------------------------
   original_patterns (internal)  = 125
   current_filtered_pattern(s)   = 7
   sampled_patterns_per_type     = 2
   sample_capture_clock          = on
   sample_pattern_type(s)        = all
-----------------------------------------------
```

Example 2

This example creates a sample of patterns 0 through 4, 6 through 10, and 20 through 29.

set_pattern_filtering -range 0 10 -norange 5 5 -range 20 29

```
// WARNING: The previous filter option has been reset.
// Internal pattern filtering is on: 20 selected pattern(s),
   125 original pattern(s).
```

report_pattern_filtering

```
   Current pattern filtering configuration
-----------------------------------------------
   original_patterns (internal)  = 125
   current_filtered_pattern(s)   = 20
   sampled_patterns_per_type     = all
   sample_capture_clock          = off
   sample_pattern_type(s)        = all
   include_range(s)              = [20, 29], [0, 10]
   exclude_range(s)              = [5, 5]
-----------------------------------------------
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

This example creates a sample of patterns that use one or more of the clocks clk1, clk2, and clk3 and other clocks except for clk4; patterns that use clk4 are not sampled.

set_pattern_filtering -clocks clk1 clk2 clk3 -no_clocks clk4

```
// Warning: The previous filter options have been reset.
// Internal pattern filtering is on: 6 selected pattern(s),
// 125 original pattern(s).
```

report_patterns

```
//        pre-
//        filter
// patt # patt #   type   cycles loads obs_proc cap_proc capture_clk_seq
// ------ ------ -------- ------ ----- -------- -------- ---------------
//    0      0   basic       1     1       -        -    [clk3]
//    1      3   basic       1     1       -        -    [clk1]
//    2      5   clk_seq     2     1       -        -    [clk1] [clk2]
//    3      6   clk_seq     2     1       -        -    [clk2] [clk3]
//    4      9   clk_seq     2     1       -        -    [clk2] [clk5]
//    5     18   clk_seq     2     1       -        -    [clk2] [clk6]
```

Notice that when pattern filtering is in effect, the report_patterns command displays the index numbers of the patterns prior to filtering as well as after filtering.

Example 4

This example creates a sample of patterns that do not pulse any clock.

set_pattern_filtering -no_clocks

```
// Warning: The previous filter options have been reset.
// Internal pattern filtering is on: 2 selected pattern(s),
// 125 original pattern(s).
```

report_patterns

```
//        pre-
//        filter
// patt # patt #   type   cycles loads obs_proc cap_proc capture_clk_seq
// ------ ------ -------- ------ ----- -------- -------- ---------------
//    0      1   basic       1     1       -        -    [-]
//    1      2   basic       1     1       -        -    [-]
```

Example 5

This example creates a sample of patterns 54, 106, 34 and 125, in that order.

set_pattern_filtering -list 54 106 34 125

```
// Warning: The previous filter options have been reset.
// Internal pattern filtering is on: 4 selected pattern(s),
// 125 original pattern(s).
```

report_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//        pre-
//       filter
// patt # patt #   type   cycles loads obs_proc cap_proc capture_clk_seq
// ------ ------ -------- ------ ----- -------- -------- ---------------
//    0     54   basic     1       1       -        -    [clk2,clk4]
//    1    106   clk_seq   3       1       -        -    [clk1] [clk3] [c
//    2     34   mult_ld   4       3       -        -    [clk1]
//    3    125   clk_seq   2       1       -        -    [clk1] [clk2]
```

Example 6

This example creates a sample of patterns that use the specified named capture procedures TranPtrn01 and TranPtrn03.

set_pattern_filtering -named_capture_procedures TranPtrn01 TranPtrn03

```
//  Warning: The previous filter options have been reset.
//  Internal pattern filtering is on: 17 selected pattern(s), 32 original
pattern(s).
```

report_patterns

```
//        pre-
//       filter
// patt # patt #   type   cycles loads obs_proc cap_proc capture_clk_seq
// ------ ------ -------- ------ ----- -------- -------- ---------------
//    0      0   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    1      1   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    2      2   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    3      3   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    4      4   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    5      5   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    6      6   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    7      7   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    8      8   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    9      9   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//   10     19   clk_seq   2       1       -  TranPtrn03 [clk] [hclk]
//   11     20   clk_seq   2       1       -  TranPtrn03 [clk] [hclk]
//   12     21   clk_seq   2       1       -  TranPtrn03 [clk] [hclk]
//   13     22   clk_seq   2       1       -  TranPtrn03 [clk] [hclk]
//   14     23   clk_seq   2       1       -  TranPtrn03 [clk] [hclk]
//   16     25   clk_seq   2       1       -  TranPtrn03 [clk] [hclk]
```

report_pattern_filtering

```
    Current pattern filtering configuration
-----------------------------------------------
    original_patterns (internal) = 32
    current_filtered_pattern(s)  = 17
    sampled_patterns_per_type    = all
    sample_capture_clock         = off
    selected named capture proc  = TranPtrn01, TranPtrn03
    sample_pattern_type(s)       = all
-----------------------------------------------
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 7

This example creates a sample of two patterns for each of the specified named capture procedures TranPtrn01 and TranPtrn03 and then saves the sampled patterns.

set_pattern_filtering -sample 2 -named TranPtrn01 TranPtrn03
write_patterns pat_TranPtrn01_TranPtrn03.ascii

Example 8

This example creates a sample of two patterns for all the named capture procedures and then saves the sampled patterns.

set_pattern_filtering -sample 2 -named ALL
write_patterns pat_all.ascii

Example 9

This example shows a pattern report before and after filtering the patterns using the -capture option.

report_patterns

```
//
//  pattern #  type          cycles load  observe_proc  capture_proc  capture_clock_sequence
//  -------  -----------  ----- ---- -----------  ----------- --------------------
//    0   basic             1     1        -        -          [-]
//    1   basic             1     1        -        -          [-]
//    2   basic             1     1        -        -          [-]
//    3   basic             1     1        -        -          [-]
//    4   basic             1     1        -        -          [sclk]
//    5   basic             1     1        -        -          [sclk]
//    6   basic             1     1        -        -          [sclk]
//    7   basic             1     1        -        -          [sclk]
//    8   basic             1     1        -        -          [sclk]
//    9   basic             1     1        -        -          [reset_]
//    10  basic             1     1        -        -          [sclk]
//    11  basic             1     1        -        -          [sclk]
//    12  basic             1     1        -        -          [sclk]
//    13  basic             1     1        -        -          [sclk]
//    14  basic             1     1        -        -          [sclk]
//    15  basic             1     1        -        -          [reset_]
//    16  basic             1     1        -        -          [sclk]
//    17  basic             1     1        -        -          [sclk]
//    18  clock_sequential  2     1        -        -          [sclk]  [clk12]
//    19  clock_sequential  2     1        -        -          [sclk]  [clk12]
//    20  clock_sequential  2     1        -        -          [reset_]  [clk12]
```

set_pattern_filtering -capture

```
//  Internal pattern filtering is on: 4 selected pattern(s), 21 original
pattern(s).
```

report_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//
//            pre-filtering
// pattern # pattern # type  cycles  load  observe_proc capture_proc  capture_clock_sequence
// ------ ------- ---   ----- ---- ------- -----------   -----------------
// 0       0        basic   1     1         -        -         [-]
// 1       4        basic   1     1         -        -         [sclk]
// 2       9        basic   1     1         -        -         [reset_]
// 3       18       clock_sequential 2     1  -       -       [sclk] [clk12]
```

Notice that each clock pulsed in different capture cycles is sampled once, since the default sample per type is one.

Example 10

This example shows a pattern report before and after filtering the patterns using the -clock_sequence option.

report_patterns

```
//
//  pattern #  type          cycles load  observe_proc  capture_proc  capture_clock_sequence
//  -------  -----------  ----- ---- -----------  ----------- -------------------
//    0   basic             1     1        -       -          [-]
//    1   basic             1     1        -       -          [-]
//    2   basic             1     1        -       -          [-]
//    3   basic             1     1        -       -          [-]
//    4   basic             1     1        -       -          [sclk]
//    5   basic             1     1        -       -          [sclk]
//    6   basic             1     1        -       -          [sclk]
//    7   basic             1     1        -       -          [sclk]
//    8   basic             1     1        -       -          [sclk]
//    9   basic             1     1        -       -          [reset_]
//    10  basic             1     1        -       -          [sclk]
//    11  basic             1     1        -       -          [sclk]
//    12  basic             1     1        -       -          [sclk]
//    13  basic             1     1        -       -          [sclk]
//    14  basic             1     1        -       -          [sclk]
//    15  basic             1     1        -       -          [reset_]
//    16  basic             1     1        -       -          [sclk]
//    17  basic             1     1        -       -          [sclk]
//    18  clock_sequential  2     1        -       -          [sclk]   [clk12]
//    19  clock_sequential  2     1        -       -          [sclk]   [clk12]
//    20  clock_sequential  2     1        -       -          [reset_]   [clk12]
```

set_pattern_filtering -clock_sequence -sample_per_type 2

```
//  Internal pattern filtering is on: 9 selected pattern(s), 21 original
pattern(s).
```

report_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//
//            pre-filtering
// pattern # pattern # type  cycles  load  observe_proc capture_proc  capture_clock_sequence
// ------ ------- ---  ----- ---- ------- -----------   -----------------
// 0       0         basic   1    1        -       -          [-]
// 1       1         basic   1    1        -       -          [-]
// 2       4         basic   1    1        -       -          [sclk]
// 3       5         basic   1    1        -       -          [sclk]
// 4       9         basic   1    1        -       -          [reset_]
// 5       15        basic   1    1        -       -          [reset_]
// 6       18        clock_sequential  2    1   -  -          [sclk]  [clk12]
// 7       19        clock_sequential  2    1   -  -          [sclk]  [clk12]
// 8       20        clock_sequential  2    1   -  -          [reset_]  [clk12]
```

Two patterns are sampled per unique clock sequence of the internal patterns because of the specified value of the -sample_per_type switch. For the [reset_] [clk12] sequence, only one pattern is available for sampling.

Example 11

This example shows a pattern report after filtering the patterns using the -shift_power option for test patterns with switching activity greater than 25% during the circuit response to test patterns.

ANALYSIS> set_pattern_filtering -shift_power response { > 25 }

```
//  Internal pattern filtering is on: 2 selected pattern(s), 3 original
//  pattern(s).
```

ANALYSIS> report_patterns

```
//
//            pre-filtering
// pattern #  pattern #  type cycles loads observe_proc capture_proc capture_clock_sequence
// ---------  ---------  ---- ------ ----- ------------ ------------ ----------------------
//        0          1   basic   1     1         -            -        [CLK]
//        1          2   basic   1     1         -            -        [CLK]
//
```

Example 12

This example shows a pattern report after filtering the patterns using the -capture_power option for test patterns with switching activity greater than 9%, using the peak of weighted switching activity.

ANALYSIS> set_pattern_filtering -capture_power { > 9 } wsa_peak

```
//  Internal pattern filtering is on: 1 selected pattern(s), 3 original
//  pattern(s).
```

ANALYSIS> report_patterns

```
//
//            pre-filtering
// pattern #  pattern #  type cycles loads observe_proc capture_proc capture_clock_sequence
// ---------  ---------  ---- ------ ----- ------------ ------------ ----------------------
//        0          2   basic   1     1         -            -        [CLK]
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

order_patterns

read_patterns

report_environment

report_pattern_filtering

report_patterns

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_pattern_source

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: analysis

Specifies the source of the test patterns.

Usage

When EDT is Off

set_pattern_source **Internal** | {**Random** | **Bist** | {{**External** ***filename***}
[-APpend | -Replace] [-Binary | -AScii | -STil | -WGl]}
[-MAsk *mask_filename* [-MASKFILE_pattern_offset *number*]]
[-PREserve_masked_observe_points] [-NOPadding] [-Store_patterns] [-ALl_patterns]}

When EDT is On

set_pattern_source **Internal** | {{**External** ***filename***}
[-APpend | -Replace] [-Binary | -AScii | -STil | -WGl]
[-MAsk *mask_filename* [-MASKFILE_pattern_offset *number*]]
[-PREserve_masked_observe_points] [-Store_patterns] [-ALl_patterns]}

Description

Note

The set_pattern_source command followed by the “run” command has been mostly superseded by the read_patterns and simulate_patterns commands. For “set_pattern_source internal” and “run”, the replacement command is create_patterns. Siemens EDA recommends using the newer commands.

Arguments

- **Internal**

  A literal that specifies the internal set of patterns. This is the default upon invocation of the tool.

- **Random**

  A literal that specifies the random patterns, capture clock, and observation point specified previously.

- **Bist**

  A required literal that specifies BIST patterns.

  There must be at least one defined LFSR or an error displays. The existence of such an LFSR means that the design successfully passed the BIST rules checking when leaving Setup.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **External *filename***

  A literal and string pair that specifies the set of patterns contained in ***filename*** when performing a simulation run. For simulation, the specified patterns are loaded into an external pattern set database.

  In analysis system mode, the tool adds effective patterns from a simulation run to the internal pattern set, if there is at least one undetected fault in the fault list. When you use the -All_patterns switch, the tool adds all simulated patterns from the external pattern set to the internal pattern set, not just the effective patterns.

  In analysis mode, a simulation run does not alter the internal pattern set.

  The external patterns can be in ASCII, binary, STIL, or WGL test pattern format.

  Note

  You may notice a drop in test coverage when using an external pattern set as compared to using generated patterns. This is an artificial drop.

  There is an issue regarding artificial drops in test coverage due to redundant faults. When you run ATPG, the tool classifies some faults as redundant (RE). The tool knows these faults are untestable when calculating test coverage. If you restart the tool, or go back to Setup mode before reading the patterns back in, the faults that were previously classified as RE are now classified as UC or UO. Now, the tool does not know these faults are untestable when calculating test coverage because faults cannot be proven redundant by fault simulation. The tool only fault simulates the external patterns. It does not target undetected faults and try to prove them redundant as it does when the pattern source is internal. Therefore, you may see a slight drop in test coverage. This drop is an artificial drop and coverage still remains the same.

  The external pattern formats are described in the *Tessent Scan and ATPG User's Manual*.

- -APpend

  An optional switch that specifies for the tool to append the patterns contained in filename to the patterns in the tool's current external pattern set.

- -Replace

  An optional switch that specifies for the tool to replace its current external pattern set with the patterns contained in filename. This is the default.

- -Binary

  An optional switch that specifies the external test pattern set is in binary format. This is used when reading in a file saved with the write_patterns -BInary command.

  If neither -Binary or -Ascii options are specified, the tool tries to open and process the file as a binary file. If this is unsuccessful, the tool then tries to open and process the file as an ASCII file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -AScii

  An optional switch that specifies the external test pattern set is in ASCII format. You cannot use this option with the Internal pattern source.

- -WGl

  An optional switch specifying that the external test pattern set is in WGL format. You must use this option if you specify external patterns that are in WGL format.

  Restrictions:

  - For Tessent FastScan and Tessent TestKompress, you must load parallel scan patterns, not serial. Patterns produced by MacroTest cannot be read back in. For more information, see the note under the External argument.
  - If you re-simulate scan patterns, you can only use serial patterns.

- -STil

  An optional switch specifying that the external test pattern set is in STIL format. You must use this option if you are loading STIL functional patterns into the tool.

  Restriction:

  - You must load parallel scan patterns, not serial.
  - The patterns must be STIL patterns originally created in these tools.

  Analysis of the STIL patterns conforms to the IEEE 1450.0 specification. The following features from the IEEE 1450.0 specification are not supported:

  - Multiple data elements per test cycle (See section 5.7 of IEEE 1450.0)
  - Non-cyclized test data (See section 5.9 of IEEE 1450.0)
  - Goto statement referring to a previous pattern label (See section 22.8 of IEEE 1450.0)
  - MatchLoop statement (See section 22.7 of IEEE 1450.0)

  Analysis of the STIL patterns supports only waveform descriptions in the WaveformTable that Tessent tools currently support. Therefore, a waveform description should not have more than two event edges, unless it is describing a double pulse. No other waveforms that have more than two event edges can be used. Also, timing expressions are limited to using only the +, -, *, /, (, and) operators on constants or variables that have time-based units.

- -MAsk

  An optional switch that masks certain observation points in external patterns added to the internal pattern set after simulation. This switch enables you to salvage patterns that result in unreliable failures on certain cells at the tester, or that fail verification due to simulation mismatches at a few observation points. Rather than discard failing patterns altogether and accept the accompanying significant reduction in test coverage, you can keep the patterns,

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

use this switch to mask the few problem observation points, and typically see only minimal impact on test coverage.

The tool obtains the masking information from a text file you create (see the mask_filename argument).

This switch affects the capture values of only those patterns from the external pattern set that the tool has added to the internal pattern set after simulation. Therefore, if you write patterns using "write_patterns -external," you do not see the masking information from the mask file reflected in the saved patterns; the saved patterns are exactly the same as the original patterns.

To save a pattern set that incorporates the masking information, use the run command to perform a simulation of the external pattern set you specified with "read_patterns -preserve_masked_observation_points." Then use the write_patterns command without the -External switch to write the resulting internal pattern set to a file.

Tip

When you use the -Mask switch, the tool automatically executes the -Store_patterns switch with it; you do not need to include an explicit -Store_patterns in your command line.

- *mask_filename*

  An optional string used with the -Mask switch to specify a mask file. The mask file is a text file you create that lists the patterns and locations you want masked. Use the statements and syntax described in Table 8-18 to create a mask file. Use one statement per line. Precede comment text with a pair of forward slashes (//). An example mask file follows the table.

  Tessent TestKompress names scan chains internally. Depending on the design architecture, you must use this internal naming format to identify Tessent TestKompress scan chains in the mask file as follows:

  Modular EDT — edt_<block_name>_channel<number>

  Top-level EDT — edt_channel<number>

  Use the report_edt_pins command to determine the <number> associated with a scan chain.

  The mask file must be in a pattern based format. Most ATE's return accumulated cycle-based failure data. Use the read_failures command to convert cycle-based failure data to pattern based data for the mask file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-18. Mask File Statements

Statement	Usage Rules
<location>	Required. Specifies a location to mask or observe. Use as many location statements as you need. List all location statements that specify a pattern index in ascending order. Syntax format depends on the type statement that precedes the location statement, as follows: **EDT Off:** • If "type mask" or "type observe" precedes the location statement, use one of these two formats: <pattern_index> <primary_output_name> <pattern_index> <chain_name> <cell_ID> • If "type pin_mask" precedes the location statement, use one of the following three formats: <primary_output_name> <chain_name> <chain_output_pin_name> **EDT On:** • If "type mask" or "type observe" precedes the location statement, use one of these three formats: <pattern_index> <primary_output_name> <pattern_index> <chain_name> <cell_ID> <pattern_index> <channel_name> <channel_cell_index> Specify the *channel_name* argument as follows: For non-modular flows: *edt_channel<number>*. For example, *edt_channel3*. For modular flows: *edt_<block_name>_channel<number>*. For example, *edt_block1_channel3*. Note, the *channel_cell_index* argument refers to the unload cycle. • If "type pin_mask" precedes the location statement, use one of the following three formats: <primary_output_name> <channel_name> <channel_output_pin_name>

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-18. Mask File Statements (cont.)

Statement	Usage Rules
type <value>	Optional. Indicates if the locations specified by subsequent location statements are to be masked (forced to X) or observed. Specify one of the following values: • mask — specifies to mask the locations, for a specific pattern • observe — specifies to observe the locations and mask all others, for a specific pattern • pin_mask — specifies to mask the locations, for all patterns. Overrides other type statements that cover the same location. Use as many type statements as you like. By default, the tools assume "type mask" for locations listed before the first type statement. A type statement inserted between two location statements for the same pattern does not take effect until the next different pattern.

Following is an example mask file:

```
//file: my_pattern_mask.ascii
//
// The next 3 lines mask 3 locations in pattern 20.
20 primary_output_12
20 chain2 31
20 chain3 101
// The next line masks 1 location in pattern 25.
25 chain1 120
type observe
// The following 2 lines mask all but the 2 specified locations in
//   pattern 40.
40 chain5 3
40 primary_output_1
// The following type statement takes effect with pattern 61.
type mask
// The next line defines a third observe location for pattern 40.
40 primary_output_2
61 chain4 120
// The next line masks chain5 for all patterns, overriding previous
//   "type observe" for cell 3 of chain 5 for pattern 40, and
//   subsequent "type observe" for cell 99 of chain5 for pattern 65.
type pin_mask
chain5
type observe
// The following line masks all cells in chain5 except cell 99 in
//    pattern 65.
65 chain5 99
```

- -MASKFILE_pattern_offset *number*

  An optional switch and integer that specifies the pattern offset number for the mask file. By default (when no offset is specified), the pattern in the mask file starts with pattern 0. This switch is useful for reducing the number of pattern files that must be loaded when only a subset need to be masked.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -PREserve_masked_observe_points

  An optional switch, used with external, that enables you to preserve existing masked observation points (wherever Xs appear in a pattern's unload and measure_po values) in external patterns loaded into the tool. When you use this switch to preserve existing masked observation points in the external patterns, the internal patterns stored after simulating the external patterns preserve these masked observation points. You can optionally specify "-Mask *mask_filename*" with additional observation points to be masked.

- -Store_patterns

  An optional switch that directs the tool to place patterns it simulates during the Good or Fault system mode into the internal pattern set. You can then use the write_patterns command to save these patterns to an external file.

  You cannot use this option with the Internal pattern source.

- -ALl_patterns

  An optional switch that, in analysis mode only, specifies for the tool to add all simulated patterns to the internal pattern set, provided there is at least one undetected fault in the fault list. The simulated patterns can be either randomly generated or can be from the specified external pattern file. Patterns that cause bus conflicts, which the tool ordinarily discards, are included when you use this switch. Without this switch, the tool adds only the effective patterns from a simulation run to the internal pattern set.

- -NOPadding

  An optional switch specifying that the source test pattern set contains ASCII patterns that are not padded for the scan load and unload data. For example, the source pattern set may be one that you wrote with the write_patterns command using its -NOPadding switch.

  You cannot use this option with the Internal pattern source.

Examples

Example 1

The following example performs fault simulation on an external pattern file:

```
set_system_mode analysis
set_pattern_source external file1
add_faults -all
simulate_patterns
report_statistics
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example fault grades transition faults using a set of path delay patterns. Assume path delay patterns generated with "set_fault_type path_delay -mask_nonobservation_points" are stored in the file, path_patterns_mask.ascii:

```
set_system_mode analysis
set_pattern_source external path_patterns_mask.ascii -mask
set_fault_type transition
add_faults -clock all
simulate_patterns
report_statistics
```

Example 3

The following example shows how to combine patterns from several different files into one pattern set, which you can save in any supported format. First, the example loads patterns from three separate pattern files into the tool's external pattern set. The patterns are in the same order in which they are loaded. The example then writes the contents of the external pattern set into a new file in the WGL format. Notice the patterns loaded are in mixed formats (ASCII and binary); also, that the tool automatically knows to use the GNU gzip utility when processing the input file with the ".gz" extension.

```
set_pattern_source external my_patterns1.ascii
set_pattern_source external my_patterns2.binary -append
set_pattern_source external my_patterns3.ascii.gz -append
write_patterns my_patterns_all.wgl -wgl -external
```

Example 4

The following commands create a new pattern set using the existing pattern set pat.ascii and keep the mask observation points plus the additional masked bit specified in maskfile my_mask:

```
set_pattern_source external external pat.ascii -store \
        -preserve_masked_observation_points -maskfile my_mask
create_patterns
write_patterns pat_new_masks.ascii
```

Example 5

The following example demonstrates how to specify the offset to speed up the pattern mask flow. Assume that a group of pattern files are created as follows:

Pat1.stil & Pat1.v contain pattern index 0 to 999

Pat2.stil & Pat2.v contain pattern index 1000 to 1999

Pat3.stil & Pat3.v contain pattern index 2000 to 2999

Pat4.stil & Pat4.v contain pattern index 3000 to 3999

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

After Verilog simulation, only pattern file Pat3.v fails simulation with the following mismatch bit in maskfile Pat3.fail:

```
// This File is simulation generated (Pat3.v)
//format pattern
//failure_buffer_limit_reached none
//data_reference cell
//pattern_id  chain/PO_name  cell_number  expected_value  simulated_value
scan_test
   2001          chain1          513            L              H
// reg132/Q
last_pattern_applied          2999
// failing_patterns= 1 simulated_patterns= 1000  simulation_time= 2600000;
failure_file_end
```

The following command enables you to load only the mismatch pattern file and mask the mismatch bits, then save the pattern file without mismatches:

```
set_pattern_source external Pat3.stil -stil -store \
        -mask Pat3.fail -maskfile_pattern_offset 2000
create_patterns
write_patterns Pat3_masked.v -verilog
write_patterns Pat3_masked.stil -stil
```

Related Topics

set_abort_limit

set_capture_clock

set_checkpointing_options

set_observation_point

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_pattern_type

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the type of pattern(s) used by ATPG simulation.

Usage

set_pattern_type [-MAX_SEQuential {OFf | *depth*}] [-SEQuential {OFf | *depth*}] [-MULtiple_load {OFf | ON}]

Description

Note

Switch settings for this command are cumulative; that is, switches retain their settings over multiple set_pattern_type commands until you specify a different setting.

Tip

If you use ATPG Expert (create_patterns), you do not need to issue this command. ATPG Expert determines the optimal settings for this command for most designs.

The set_pattern_type command determines the type of patterns the tool uses during ATPG simulation. The switches for this command retain their settings over multiple commands, until you change them.

The -Sequential switch provides the ability to use clock sequential cells. For example, "-sequential 2" enables the tool to create clock sequential patterns with up to two clock pulses between scan loads. Try to use the smallest possible depth, because the depth affects both memory requirements and performance. Use the report_sequential_fault_depth command to obtain estimates of the maximum test coverage possible with different sequential depth settings.

When you increase the sequential depth, the tool places all current ATPG untestable faults in the untested fault class. However, you can improve multiple-clock pattern compression by increasing the sequential depth. You cannot decrease the sequential depth when there are any active patterns. You cannot set the depth greater than one if a clock procedure contains a restore_bidis statement, because the procedure cannot use sequential ATPG.

Arguments

- -MAX_SEQuential OFf | *depth*

  An optional switch and argument pair that specifies the maximum clock sequential depth if the tool automatically increases the depth during test generation. (When using the create_patterns command to generate test patterns, the tool may increase sequential depth automatically in order to achieve the maximal test coverage.)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When you use -max_sequential, ATPG expert can start at a lower depth and then later increase to the max value. The argument indicates the upper limit, but the tool starts from the optimal depth. For example:

If the optimal depth is two and you use "-max_sequential 3", the tool overrides the setting and start with two.

The off option (the default) disables the restriction for clock sequential depth so that the tool can increase the clock sequential depth to any number. When you use this switch to set maximum clock sequential depth, the depth set by the -Sequential switch cannot be greater than the maximum clock sequential depth. The maximum clock sequential depth can be an integer value from 1 to 256.

- -SEQuential OFf | *depth*

  An optional switch and argument pair that specifies the depth of the design's non-scan sequential elements (its sequential depth). The integer can be a value between 0 and 255, but the recommended practice is to set the depth as low as possible, because it affects both memory requirements and performance. The default sequential depth is off, which is equivalent to depth zero.

  You use the -sequential switch most often to specifically set the depth to a specific number. It indicates the sequential start depth, even if it is not the optimal number. ATPG Expert does not modify it.

  You can regard the difference between -sequential and -max_sequential as a pattern generation order strategy. For example:

  If the optimal depth is two and you use "-sequential 3", the tool starts with a depth of three.

  A setting of off or 0 enables the tool to create only non-sequential patterns, while a setting of 1 enables the tool to also create basic sequential patterns; either setting results in patterns having a maximum sequential depth of 1. A setting of 2 or more enables clock sequential pattern generation.

  For -SEQuential, if two numbers are displayed in the settings summary, the first number is the clock sequential depth and the second number is the maximal clock sequential depth for clock sequential test cube generation.

  Note

  You must not enable clock sequential patterns if a clock procedure contains a restore_bidis statement; the procedure cannot use clock sequential ATPG.

- -MULtiple_load OFf | ON

  An optional switch and literal pair that enables the tool to create multiple load patterns when you have enabled clock sequential pattern generation (the sequential depth is set to two or more). A depth of four is required if the RAM has data hold, H or H1, in its _cram_read port definition. Although it is generally not recommended to set depth higher than required, setting depth to four is acceptable if you are targeting RAMs with the -Sequential option. This switch is off by default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For -MULtiple_load, “ON” means that the clock sequential simulation engine enables you to simulate test patterns with multiple loads, and “OFF” means that the multiple loads for clock sequential test cube generation is disabled.

Refer to “Multiple Load Patterns” in the Tessent Scan and ATPG User’s Manual for additional details about the multiple load pattern type.

Examples

Example 1

The following example displays the current pattern type settings:

set_pattern_type

```
//  Settings : -SEQuential     0
//             -MAX_SEQuential 5
//             -MULtiple_load  OFF
```

Example 2

The following example disables the restriction for clock sequential depth so that the tool can increase the clock sequential depth to any number:

set_pattern_type -max_sequential off

```
//  Settings : -SEQuential     0
//             -MULtiple_load  OFF
```

Example 3

The following example creates patterns with up to two clock pulses between scan loads:

set_pattern_type -sequential 2

```
//  Settings : -SEQuential     2
//             -MULtiple_load  OFF
```

Example 4

The following example resets sequential depth to its default setting (0).

set_pattern_type -sequential 0

```
//  Settings : -SEQuential     0
//             -MULtiple_load  OFF
```

Example 5

The following example specifies to create multiple load patterns for testing RAMs.

set_pattern_type -sequential 4 -multiple_load on

```
//  Settings : -SEQuential     4
//             -MULtiple_load  ON
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

report_sequential_fault_depth

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_physical_translation

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Translates special Verilog characters to SPICE-readable characters that Calibre applications can read.

Usage

set_physical_translation [-Path_divider *path_divider*] [-Substitute *substitute*] [-Replacement *replacement*] [-Global_prefix *global_prefix*] [-Hier_prefix *hier_prefix*] [-STrip_prefix *strip_prefix*] [-Array_delimiters *array_delim*]

Description

You can restore any translation setting back to its default setting by running the command without the corresponding argument. For specifics for when you use this command, see "Net Pair Identification with Calibre for Bridge Fault Test Patterns" in the *Tessent Scan and ATPG User's Manual* for complete information.

Arguments

- -Path_divider *path_divider*

  An optional switch and string specifying the instance path divider. The default is "/X".

- -Substitute *substitute*

  An optional switch and string for substituting a "#" (pound sign) for a leading character or set of characters for pins, wires, and instance names. By default, the tool substitutes leading "$"s (dollar signs) with "#"s (pound signs). You can specify a maximum of 10 characters for substitution. If you specify the -Replacement switch and string, then the tool uses this character for the substitution character.

- -Replacement *replacement*

  An optional switch and string replacing the default substitution character. The default is a "#" (pound sign). If you specify a character, then the tool uses this character instead of the default when substituting a leading character or characters for pins, wires, and instance names.

- -Global_prefix *global_prefix*

  An optional switch and string for changing the default global prefix character the tool inserts for pins, wires, and instance names. The default is a "#" (pound sign).

- -Hier_prefix *hier_prefix*

  An optional switch and string changing the hierarchy prefix. The default is " " (space).

- -STrip_prefix *strip_prefix*

  An optional switch and string stripping the prefix. This argument is in Calibre form (for example, to strip Xbogus_top/Xmodule/net, you specify -strip_prefix Xbogus_top).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Array_delimiters *array_delim*

  An optional switch and string changing the array delimiter. The default is “[ ]” (square brackets).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_possible_credit

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the percentage of credit that the tool assigns possible-detected faults.

Usage

set_possible_credit ***percentage***

Description

The set_possible_credit command specifies the percentage of possible-detected faults that the tool considers as detected when calculating the test coverage, fault coverage, and ATPG effectiveness. Refer to "Testability Calculations" in the *Tessent Scan and ATPG User's Manual* for the equations that the tool uses in these calculations.

When you run the tool, the default credit value for possible-detected faults is 50 percent.

Arguments

- ***percentage***

  A required integer from 0 to 100 that specifies the percentage of possible-detected faults that you want the tool to consider as detected when calculating the test coverage, fault coverage, and ATPG effectiveness. The default value is 50.

Examples

The following example sets the credit for possible-detected faults as 25 percent for determining the fault coverage, test coverage, and ATPG effectiveness:

```
set_system_mode analysis
set_possible_credit 25
create_patterns
report_statistics
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_power_control

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Enables low-power ATPG and sets up a shift or capture threshold.

Usage

set_power_control [off | on | capture [off | on] | shift [off | on]
[-control_per_edt_block {off | on}]
[-rejection_threshold_percentage { *percentage* | off }
[-st_peak | -st_average | -wsa_peak | -wsa_average]]
[-switching_threshold_percentage *percentage*]]
[-all | -global | -partitions *object_spec* … | -instances *object_spec* … |
-modules *object_spec* … | -clock_domains *object_spec* …]

Description

The set_power_control command enables low-power ATPG and sets up a switching or capture threshold.

For more information, see the Low-Power ATPG section in the *Tessent Scan and ATPG User's Manual*.

Note

If you use the set_power_control command, the ATPG test pattern count may increase to satisfy the power constraints. The tool run time may also increase because of the extra effort required to compute the switching activity for every test pattern.

Hybrid TK/LBIST Flow Specifics

Specifies the switching threshold in the patterns for fault simulation.

You can only use the low-power shift usage of this command for logicBIST. Otherwise, the tool reports an error.

Note

You cannot use the command's existing -rejection_threshold_percentage switch with the low-power logic BIST controller.

Arguments

- off | on

  An optional switch that disables or enables both low-power capture and low-power shift.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- capture [off | on]

  An optional switch and literal pair that enables low-power capture. The tool bases low-power capture threshold calculations on state element activity, which is scan and non-scan cells. The options for low-power capture:

  off — Disables low-power capture.

  on — Enables low-power capture. This is the default.

- shift [off | on]

  An optional switch and literal pair that enables low-power shift. The tool bases low-power shift threshold calculations on scan cell activity. The options for low-power shift:

  off — Disables low-power shift.

  If you set the set_edt_power_controller command to "enabled", the default for the "set_power_control shift" command is "on" for TestKompress only.

  If you have run the "set_edt_power_controller shift enabled" command, the "set_power_control shift off" command directs ATPG to load the low-power shift bits with 1, which enables all internal chain inputs. However, it does not disable low-power TestKompress hardware. To disable low-power TestKompress hardware and bypass the low power registers, you must force "edt_low_power_shift_en off" in the test procedure file and run the "set_edt_power_controller shift disabled" command.

  For more information on using low-power ATPG with Tessent TestKompress, see "Low-Power Test" in the *Tessent TestKompress User's Manual*.

  on — Enables low-power shift. This is the default.

  If you did not enable the EDT low-power controller during EDT IP creation, but specified low-power shift with the "set_power_control shift on" command before ATPG, the tool reports an error message:

```
//  Error: Shift power control is requested, but power control
//    hardware is not present or is disabled.
//    Please check the "set_edt_power_controller shift" command.
```

- -control_per_edt_block { off | on }

  An optional switch and literal pair that enables the shift power calculation for each EDT block. The tool rejects test patterns with peak load and response shift transitions exceeding the thresholds you specify with the other arguments of this command.

  Note

  The tool excludes EDT blocks that only consist of uncompressed chains from this calculation and rejection.

  The options for controlling power by EDT block:

  off — Does not perform the shift power calculation independently for each EDT block. The tool applies the power thresholds globally to the whole design according to the other settings. This means that an individual block may have switching activity that

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

exceeds the threshold for that block while the whole design remains below the threshold so the tool does not reject the patterns. This is the default.

on — Performs the shift power calculation and rejection for each EDT block. If the design contains multiple EDT blocks and you specify shift power control, the tool applies the threshold to each EDT block independently and also globally across the whole design. This means that the tool may discard patterns because of some blocks with higher switching activity if they reach the threshold even though the threshold is not exceeded across the whole design.

- -rejection_threshold_percentage {*percentage* | off}

  An optional switch and integer or literal pair that specifies a state element switching limit per test pattern. The *percentage* integer can be 1 to 100. If you specify a threshold, the tool rejects test patterns exceeding that threshold, which may impact test coverage. The default is off.

  Tip

  Use the -switching_threshold_percentage switch when using this switch. The rejection threshold percentage should not be less than the switching threshold percentage. We recommend that the rejection threshold percentage is at least five percent higher than the switching threshold percentage.

  Note

  You cannot use this switch with the low-power logic BIST controller.

- -st_peak | -st_average | -wsa_peak | -wsa_average

  An optional literal that filters the test pattern rejection threshold based on a specified metric. When a specified metric exceeds the rejection threshold, the pattern is discarded. These options can only be used with the low-power capture feature. Literals include:

  -st_peak — Peak of state transitions. This is the default.

  -st_average — Average of state transitions.

  -wsa_peak — Peak of weighted switching activity.

  -wsa_average — Average of weighted switching activity.

  Weighted Switching Activity is defined in the *Tessent Scan and ATPG User's Manual*. Calculations of weighted switching activity (WSA) include all combinational gates.

  Note

  For backward compatibility, these switches are recognized without the leading hyphen (-); for example, "st_peak" instead of "-st_peak".

- -switching_threshold_percentage *percentage*

  An optional switch and integer pair that specifies the target switching threshold for low-power ATPG. The <*percentage*> integer can be 1 to 100 (except for the Hybrid TK/LBIST flow when it can be 1 to 50)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The tool attempts to create test patterns that are below the switching threshold you specified. It is possible that the tool may be unable to satisfy your request and the observed switching activity may be higher in some cases. To specify a hard limit on switching activity, use the -rejection_threshold_percentage option.

By default, for Tessent TestKompress applications, the switching threshold during shift for ATPG is set to match the maximum switching threshold defined for the power controller hardware if that threshold is equal to or greater than 15 percent. If the maximum hardware switching is less than 15 percent, the software default is 15 percent. Specify the maximum hardware switching using the following command:

```
set_edt_power_controller \
     Shift -MIN_Switching_threshold_percentage <percentage_integer>
```

The hardware switching setting determines a bias range for the software switching percentage you specify with the -switching_threshold_percentage argument. We recommend that you specify both switching thresholds within the same bias range. Otherwise, the tool reports a warning message, such as the following:

```
//  Warning: Specified software switching threshold [5] is not
//    consistent with the switching threshold used to generate the
//    shift power control hardware [20] in block "blk1".
//    The software and hardware thresholds should be in the same
//    bias range (except for the full control case).
//    The following are the valid bias ranges:
//      [0, 11], [12, 24], [25, 50].
```

For more information, see "Low-Power Test" in the *Tessent TestKompress User's Manual*.

In the Hybrid TK/LBIST flow, the tool runs fault simulation with the appropriate control register values to implement the specified switching threshold. The tool seeds the registers with the same values used during pattern simulation when saving the tester format patterns in Tessent Shell. It is not possible to save hardware default-mode tester patterns when the switching threshold specified during fault simulation is different from the switching threshold specified during IP creation. The tool reports an error message if the low-power hardware cannot support this switching threshold.

For more information, see the *Hybrid TK/LBIST Flow User's Manual*.

- -all

  An optional switch that turns off power control for the entire design—every instance, module, clock domain, and partition. You must specify this option with the "off", "capture off", or "shift off" argument. Specifying this argument with an argument that turns power control on results in the following error:

  ```
  Error: The switch -all cannot be specified with setting power
  control on.
  ```

- -global

  An optional switch that specifies power control for the entire design only. This is the default value.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -partitions *object_spec*

  An optional switch and repeatable string pair that specifies capture power control settings for every partition specified by the *object_spec*. You define partitions with the add_design_partition command. The -partitions argument applies the power control settings described by the other arguments to every instance in the specified partition.

  Note

  You can only specify this argument with the "capture" argument. Specifying this argument with "on", "off", or "shift" results in the following error:

  ```
  Error: The switch -partitions cannot be specified without using
  set_power_control capture.
  ```

- -instances *object_spec*

  An optional switch and repeatable string pair that specifies capture power control settings for every instance specified by the *object_spec*. An *object_spec* consists of one or more pathnames. The -instances switch applies the power control settings described by the other arguments to every specified instance.

  Note

  You can only specify this argument with the "capture" argument. Specifying this argument with "on", "off", or "shift" results in the following error:

  ```
  Error: The switch -instances cannot be specified without using
  set_power_control capture.
  ```

- -modules *object_spec*

  An optional switch and repeatable string pair that specifies capture power control settings for every instance of the specified module or modules. An *object_spec* consists of one or more module names. The -modules switch applies the power control settings described by the other arguments to every instance derived from the specified modules.

  Note

  You can only specify this argument with the "capture" argument. Specifying this argument with "on", "off", or "shift" results in the following error:

  ```
  Error: The switch -modules cannot be specified without using
  set_power_control capture.
  ```

- -clock_domains *object_spec*

  An optional switch and repeatable string pair that specifies capture power control settings for every gate controlled by the specified clock. An *object_spec* consists of one or more clock pin pathnames. The -clock_domains switch applies the power control settings described by the other arguments to every instance derived from the specified clock domains.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

You can only specify this argument with the “capture” argument. Specifying this argument with “on”, “off”, or “shift” results in the following error:

```
Error: The switch -clock_domains cannot be specified without using
set_power_control capture.
```

Examples

Example 1

The following example sets low-power capture with a 30 percent threshold and low-power shift with a 20 percent switching threshold and generates test patterns:

```
set_power_control capture on -switching_threshold_percentage 30 \
        -rejection_threshold_percentage 35
set_power_control shift on -switching_threshold_percentage 20 \
        -rejection_threshold_percentage 25
create_patterns
```

Example 2

The following example creates compression logic with a power controller configured to accommodate a minimum of 20 percent switching threshold and enabled for test pattern generation.

```
set_edt_power_controller shift -min_switching_threshold_percentage 20
…
write_edt_files low_power_enabled_edt.v
```

Given the compression logic created with the preceding command, the following commands are used during test pattern generation to enable the controller with a 22 percent switching threshold for shift and generate test patterns.

```
set_power_control shift on -switching_threshold_percentage 22
set_system_mode analysis
create_patterns
```

Example 3 (Hybrid TK/LBIST Flow Only)

This example configures the low-power logicBIST logic to target a switching threshold of 10 percent. You can see the actual values loaded in the low-power control registers to achieve this switching threshold using the report_edt_configurations command. Run the report_power_metrics command for a report of power metrics.

```
set_power_control shift on -switching 10

set_system_mode analysis

report_edt_configuration
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//
//  IP version:                     5
//  Shift cycles:                   128,  120 (internal scan length) + 8
//                                  (additional cycles)
//  External scan channels:         4
//  Internal scan chains:           480
//  Masking bits:                   53
//  Longest chain range:            2 - 120
//  Decompressor size:              32
//  Compactor type:                 Xpress
//  Scan cells:                     57600
//  Compression per pattern:        112.50x (ATPG bypass = 4 x 14400,
//                                  EDT = 4 x 128)
//  Clocking:                       edge-sensitive
//  Compactor pipelining:           Off
//  Chain mask:                     On (load 0, unload 0)
//  LBIST:                          On
//    MISR compactor pipelines:     0
//    PRPG seed:                    00000001
//    Low power shift controller:   Enabled and active
//      Min switching threshold:    15%
//      Hold value:                 1010
//      Toggle value:               1011
//      Switching control:          0011
//
```

report_power_metrics -shift

```
//  Note: Pattern classification is automatically disabled.
Power Metrics                 Min.   Average   Max.
--------------------------  ------  -------  ------
Load Shift Transitions       4.45%   14.26%  23.51%
Response Shift Transitions  42.48%   49.16%  50.04%
```

Example 4

The following example specifies different capture power control thresholds for an instance named core_ins_1, a module named core_mod_1, and a clock domain named fclk_1:

```
set_power_control capture on -switching_threshold_percentage 15
      -rejection_threshold_percentage 20
      -instances core_ins_1
set_power_control capture on -switching_threshold_percentage 10
      -rejection_threshold_percentage 15
      -modules core_mod_1
set_power_control capture on -switching_threshold_percentage 12
      -rejection_threshold_percentage 17
      -clock_domains fclk_1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 5

The following example specifies the capture power control thresholds for a collection of instances and a collection of modules:

```
set_power_control capture on -switching_threshold_percentage 15
        -rejection_threshold_percentage 20
        -instances [ get_instance -of_module core_mod_1 ]
                [ get_instance -of_module core_mod_2 ]
set_power_control capture on -switching_threshold_percentage 15
        -rejection_threshold_percentage 20
        -modules [ get_module -of_instance core_inst_1 ]
                [ get_module -of_instance core_inst_2 ]
```

Example 6

The following example turns off capture power control for an instance named core_ins_1, a module named core_mod_1, and a clock domain named fclk_1:

```
set_power_control capture off -instances core_ins_1
set_power_control capture off -modules core_mod_1
set_power_control capture off -clock_domains fclk_1
```

Example 7

The following example sets global thresholds across the design for power control and then overrides them with lower thresholds for the power-critical instance core_ins_1:

```
set_power_control capture on -switching_threshold_percentage 30 \
        -rejection_threshold_percentage 35 -global
set_power_control capture on -switching_threshold_percentage 15 \
        -rejection_threshold_percentage 20 -instances core_ins_1
```

Example 8

The following example turns off capture power control for the entire design, including every partition, module, instance, and clock domain:

```
set_power_control capture off -all
```

Example 9

The following example turns off capture power control for the entire design at the top level. It does not affect capture power control for partitions, modules, instances, and clock domains:

```
set_power_control capture off -global
```

Example 10

The following example rejects patterns with an average WSA percentage greater than 35% in the partition block1:

```
set_power_control capture on -switching_threshold_percentage 30 \
        -rejection_threshold_percentage 35 -wsa_average -partitions block1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

report_edt_configurations

report_power_metrics

set_edt_power_controller

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_power_metrics

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Calculates shift and capture power during test pattern generation and simulation.

Usage

set_power_metrics [-SHIft {OFf | ON}] [-CAPture {OFf | ON}]
[-COMPosite_shift_power {OFf | ON}] [-X_CAPture_transition_credit *float*]
[-X_SHIft_transition_credit *float*]

Description

Shift power is the test power consumed during the loading of test stimuli and the unloading of responses. Capture power is the test power consumed by switching activity in the circuit during capture cycles.

Note

When you use set_power_metrics, there is an impact to the tool run time during create_patterns and simulate patterns, because of the extra effort required to compute the switching activity for every test pattern.

Arguments

- -SHIft OFf | ON

  An optional switch and literal pair that enables the calculation of the power consumed for switching activity in the circuit during shift.

- -CAPture OFf | ON

  An optional switch and literal pair that enables calculation of the power consumed for switching activity in the circuit during capture cycles.

- -COMPosite_shift_power OFf | ON

  An optional switch and literal pair that combines the load and unload switching activity in shift power calculations. By default, the load and unload switching activity is computed separately.

- -X_CAPture_transition_credit *float*

  An optional switch and floating number pair that specifies the credit given for x-to-x, x-to-known, and known-to-x capture transitions when calculating switching activity in capture cycles (the default values are 0.1, 0.05, 0.05, respectively). The specified float value must be between 0 and 1, and it applies to all three x values.

- -X_SHIft_transition_credit *float*

  An optional switch and floating number pair that specifies the credit given for x-to-x, x-to-known, and known-to-x shift transitions when calculating switching activity during shift.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The specified value applies to all x values. The float number should be between 0 and 1. The default is 0.5.

Examples

Example 1

The following example calculates shift power and capture power during test pattern generation, fault simulation, and good simulation and displays the capture power for capture cycles and the shift power for shift transitions.

set_power_metrics -shift on -capture on
create_patterns

```
            Statistics report
-----------------------------------------
                          #faults   #faults
fault class               (coll.)   (total)
-----------------------   -------   -------
FU (full)                  70256     88262
-----------------------   -------   -------
UC (uncontrolled)             92       118
UO (unobserved)              482       626
DS (det_simulation)        35006     44416
DI (det_implication)       17321     22589
PU (posdet_untestable)        46        62
PT (posdet_testable)          11        23
UU (unused)                  134       134
TI (tied)                     94        98
RE (redundant)               240       265
AU (atpg_untestable)       16830     19931
-----------------------   -------   -------
test_coverage              75.02%    76.39%
fault_coverage             74.52%    75.96%
atpg_effectiveness         99.17%    99.14%
-----------------------------------------
#test_patterns                          494
    #clock_sequential_patterns          494
#simulated_patterns                     512
CPU_time (secs)                        32.4
-----------------------------------------
Power Metrics                Min.   Average   Max.
--------------------------  ------  -------  ------
WSA                          9.03%   18.74%  36.82%
State Element Transitions    8.99%   25.44%  47.33%
--------------------------  ------  -------  ------
Load Shift Transitions      39.06%   44.62%  51.66%
Response Shift Transitions   6.45%   39.38%  50.54%
-----------------------------------------
```

For Power Metrics:

- The WSA (Weighted Switching Activity) lists the minimum, average, and maximum percentage of the test power consumed by the switching activity of the combinational

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

logic and state elements during capture cycles. Weighted Switching Activity is defined in the *Tessent Scan and ATPG User's Manual*.

- The State Element Transitions row lists the minimum, average, and maximum percentage of test power consumed in terms of state element transitions during capture cycles.
- The Load Shift Transitions row lists the minimum, average, and maximum percentage of test power consumed in terms of scan cell transitions during the loading of test stimuli.
- The Response Shift Transitions row lists the minimum, average, and maximum percentage of test power consumed in terms of the scan cell transitions during the unloading of responses.

Example 2

The following example reports the power metrics for external test patterns, *mypat.ascii*, with the composite shift power metric on and then off:

read_patterns mypat.ascii
set_power_metrics -composite on
report_power_metrics -external

```
Power Metrics                 Min.   Average   Max.
--------------------------  ------  -------  ------
WSA                         10.62%   12.54%  13.65%
State Element Transitions   13.59%   18.71%  20.94%
--------------------------  ------  -------  ------
Peak Cycle
--------------------------
WSA                         10.62%   12.54%  13.65%
State Element Transitions   13.59%   18.71%  20.94%
--------------------------  ------  -------  ------
Load Shift Transitions      15.38%   25.00%  30.77%
Response Shift Transitions  17.95%   21.15%  25.64%
Composite Shift Transitions 13.24%   20.08%  26.69%
```

set_power_metrics -composite off
report_power_metrics -external

```
Power Metrics                 Min.   Average   Max.
--------------------------  ------  -------  ------
WSA                         10.62%   12.54%  13.65%
State Element Transitions   13.59%   18.71%  20.94%
--------------------------  ------  -------  ------
Peak Cycle
--------------------------
WSA                         10.62%   12.54%  13.65%
State Element Transitions   13.59%   18.71%  20.94%
--------------------------  ------  -------  ------
Load Shift Transitions      15.38%   25.00%  30.77%
Response Shift Transitions  17.95%   21.15%  25.64%
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

The following example sets up power metric calculations to use composite shift, generates test patterns, and reports shift power for all the patterns:

set_power_metrics -composite_shift on
create_patterns
report_power_metrics -shift -patterns all

```
//
//            Pattern             Shift
//   Pat. loaded Pat. unloadedAverage
//   ----------- -------------   -------
//   0            -              10.60%
//   1            0               8.60%
//   2            1              14.20%
//   3            2               8.20%
//   pad          3               9.56%
```

Example 4

The following example reports the power metrics with composite shift for both scan tests and chain tests:

set_power_metrics -composite_shift on
report_power_metrics -shift -all_test -patterns all

```
// Chain test :
//
//            Pattern             Shift
//   Pat. loaded Pat. unloaded   Average
//   ----------- -------------   -------
//   0           -               32.89%
//   1           0               48.60%
//   2           1               54.20%
//   pad         2               49.56%
//
// Scan test :
//
//            Pattern             Shift
//   Pat. loaded Pat. unloaded   Average
//   ----------- -------------   -------
//   0            -              10.60%
//   1            0               8.60%
//   2            1              14.20%
//   3            2               8.20%
//   pad          3               9.56%
```

Related Topics

report_power_metrics

set_power_control

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_procedure_cycle_checking

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup

Enables test procedure cycle timing checking to be done immediately following scan chain tracing during design rules checking.

Usage

set_procedure_cycle_checking **ON** | **OFf**

Description

The set_procedure_cycle_checking command helps detect timing problems in test procedures earlier on in the ATPG process. By default, the test procedure cycle timing checking is performed after scan chain tracing. If an error condition is detected, the tool remains in the Setup mode. You can then modify the test procedures and reissue the set_system_mode command.

Arguments

- **ON**

  A literal that specifies for the tool to set_procedure_cycle_checking ON. This is the default behavior upon invocation of the tool.

- **OFf**

  A literal that specifies for the tool to set_procedure_cycle_checking OFF. In order to turn procedure cycle_checking off, you must be in Setup mode.

Examples

```
set_system_mode setup
set_procedure_cycle_checking OFf
```

Related Topics

set_system_mode

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_procedure_retargeting_options

Context: dft, patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: setup

Controls the behavior of the automatic merging and mapping (retargeting) of core-level load_unload and shift procedures to top-level test procedures and mapping of core-level test_setup and test_end IJTAG procedures.

Usage

set_procedure_retargeting_options [-scan {ON | OFf}]
[-ijtag {ON | OFf}]
[-dft_signals_icall {ON | OFf}]
[-timeplate *timeplate_name*]
[-pulse_during_shift *clock_port_list*]
[-update_procedures_for_instruments {on | off}]

Description

By default, the tool automatically retargets core-level test_setup and test_end IJTAG procedures, and adds the needed iCalls to the test_setup or test_end procedure. If you disable the -ijtag behavior by setting this switch to off, you must manually add all core level iCalls.

In patterns -scan context, by default, the tool automatically retargets the core-level load_unload and shift test procedures to the chip-level and merges the retargeted core-level procedures with top-level procedures if top-level procedures exist.

In patterns -scan_retargeting context, by default, the tool automatically retargets the core-level load_unload and shift test procedures to the chip-level if chip-level load_unload and shift test procedures do not exist.

If you disable the -scan behavior and the chip-level load_unload and shift test procedures are missing from the top-level test procedure, the tool generates an error.

Arguments

- -scan {ON | OFf}

  An optional switch and literal pair that specifies to the tool whether to enable test procedure retargeting. By default upon invocation, test procedure retargeting is enabled. To disable test procedure retargeting, you must be in Setup mode. For more information on test procedure retargeting, see "Core Mapping for ATPG Process Overview" in the *Tessent Scan and ATPG User's Manual*.

- -ijtag {ON | OFf}

  An optional switch and literal pair that specifies whether automatic mapping of core-level IJTAG test_setup and test_end procedures to the next higher level of hierarchy is enabled.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You should use the Off switch to disable automatic IJTAG mapping if you are manually generating the entire top-level test_setup and test_end procedures.

Note

The tool still makes the existing iCalls for dft_signals and iReset even when you use the "set_procedure_retargeting_options -ijtag off" command. These are appended to the testproc you defined with the set_procfile_name command.

- -dft_signals_icall {ON | OFf}

  An optional switch and literal pair the specifies whether to automatically add the DFT signals iCall to the test_setup procedure. This is the default.

  You may manually add an iCall to the test_setup to configure a DFT signal; however, the tool's automation would still generate an iCall in test_setup for the same DFT signal. This is because the tool is unaware that the auto-generated and user-added iCalls are performing the same task. If you have ensured that all the iCalls needed for DFT signals have been added manually, you can use this switch to disable automatically adding the DFT signals iCall to the test_setup.

  When set to OFf, the tool does not add the DFT signals iCall to the test_setup procedure.

- -timeplate *timeplate_name*

  An optional switch and string pair that specifies the timeplate to be used in the generated load_unload and shift procedures. The tool determines the timeplate to use as follows:

  a. Uses the timeplate specified by the -timeplate switch if one is specified.

  b. If one is not specified:

     - In patterns -scan context, if the top-level load_unload and shift procedures are present, the tool uses the timeplate from the top-level procedure.

     - In patterns -scan_retargeting context or if top-level procedures are not provided, the tool uses the default test_setup timeplate.

  c. If neither a tool-generated or user-defined test_setup procedure exists, the tool uses the default timeplate. For more information on the tool-generated test_setup procedure, see the description in "Test Procedure Retargeting for Scan Pattern Retargeting" in the *Tessent Scan and ATPG User's Manual*.

  d. If the default timeplate is not specified, the tools uses the first timeplate read in.

- -pulse_during_shift *clock_port_list*

  An optional switch and string pair that specifies to pulse the clocks specified by the clock_port_list argument during the shift procedure. The clock_port_list argument can be a Tcl list of clocks or gate_pins that are primary inputs. To reset this specification, re-invoke this option and provide an empty list.

  This option is typically used to specify clocks that drive top-level pipeline stages outside the core. The tool can map the load_unload and shift procedures from the core(s) to the top.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

However, if top-level pipeline stages are clocked by separate clocks, the tool does not know to pulse them, and, as a result, is unable to trace the scan channels from the cores to the top (R2 DRC violations). This option instructs the tool to additionally pulse the specified clock(s) in the created top-level shift procedure.

- -update_procedures_for_instruments {on | off}

  An optional switch and literal pair that specifies whether to enable automated test procedure creation or updates required by instruments. Instruments can come from one of the following:

  - DFT IP core instances, such as EDT, OCC, and NcpIndexDecoder instances, added using the add_core_instances command.
  - Inferred by the tool during system mode transition.
  - Inferred by commands such as import_scan_mode.

  By default, the test procedure updates for instruments are enabled. To disable test procedure updates, you must be in Setup mode.

  Turn this switch off to ensure that the tool does not internally modify your custom test procedures. When SSN is present and active, this switch has no impact because the tool always creates the shift and load_unload procedures.

Examples

Example 1

The following example specifies a timeplate to use in the load_unload and shift procedures.

```
set_procedure_retargeting_options -timeplate gen_tp1
```

Example 2

The following example specifies to pulse two clocks.

```
set_procedure_retargeting_options -pulse_during_shift {top_clock pipe_clock}
```

Related Topics

set_procfile_name

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_procfile_name

Context: all contexts

Mode: setup

Specifies a new procedure file for the tool to process at a later time.

Usage

set_procfile_name ***procedure_filename*** [-override_scangroup_proc]

Description

The set_procfile_name command reads the specified procedure file and performs basic syntax checking. The tool reports errors it finds but does not process and store the file at the time you issue the command.

Note

The procfile may contain Tcl variable that are substituted when the file is read. The referenced Tcl variable must exist in the Tcl global namespace scope. If you are setting the variables within a proc, make sure to use :: as a prefix to the variable name such that it is set in the global namespace. For example, use "set ::my_period 4" such that $period exits when processing the proc files. For more info about Tcl variable substitution inside the procfile, refer to "Additional Support for Test Procedure Files" in the *Tessent Shell User's Manual*.

You can specify only one procedure file with this command. If you issue the command again, the new procedure file overrides the one previously set.

If procedure files with scan groups exist, the tool processes the scan group procedure files first, followed by the procedure file specified in this command. You can override the scan group procedure files and only process ***procedure_filename*** by specifying the -override_scangroup_proc option.

For scan pattern retargeting, specifies the top-level test procedure file to be used for saving top-level patterns. In this case, the following procedures are expected to be defined in this file: test_setup, load_unload, shift, and external_capture.

For core mapping for ATPG and pattern retargeting, you must provide the chip-level test procedure file; the test procedure file must minimally include the timeplate definition. You do not need to provide chip-level load_unload and shift test procedures; if they do not exist, the tool automatically generates them based on the information in the core-level test procedure files. This step merges the core test procedures if there is more than one core, and maps them to the top-level test procedures. You may also need to specify the test_setup procedure for the design, if you need to set up the chip. All other procedures are automatically mapped by the tool if they are not specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For techniques to handle slow scan enable transitions, see "Delaying Clock Pulses in Shift and Capture to Handle Slow Scan Enable Transitions" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- ***procedure_filename***

  A required string that specifies the pathname to the procedure file.

  You can use the tab completion method to list possible files for ***procedure_filename*** that exist in the current or specified directory. For example, specifying "set_procfile_name " and then pressing the Tab key lists the files in the current directory. Specifying "set_procfile_name ../data/test_pr" and pressing the Tab key lists the files with names starting with test_pr in the ../data directory.

- -override_scangroup_proc

  An optional argument that causes the tool to disregard scan group procedure files and only process ***procedure_filename***.

  Note

  This switch will be ignored in dft -scan and dft -test_points contexts.

Examples

Example 1

The following example shows how the set_procfile_name command is used in an IJTAG retargeting flow.

```
set_context patterns -ijtag -no_rtl
…
set_procfile_name data/setup.proc # initial syntax checking done
set_system_mode analysis # procedure file opened and parsed here
```

Example 2

The following example shows how to override the scan group procedure files.

```
set_procfile_name data/setup.proc -override_scangroup_proc
```

Example 3

The "Retargeting Example" in the*Tessent Scan and ATPG User's Manual* demonstrates the use of this command in a design context.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_quick_synthesis_options

Context: unspecified, dft, patterns

Mode: setup

Sets options for quick synthesis.

Usage

set_quick_synthesis_options [-Parallel_synthesis {maxcpu | *integer*}]
[-Skip_synthesis_of_two_dimensional_arrays_larger_than {unlimited | *integer*}]
[-blackbox_design_ware_modules {on | off}]
[-complete_synthesis {on | off}]
[-trace_allowed_no_source {on | off}]
[-trace_allowed_no_destination {on | off}]
[-backward_tracing_of_control_signal_limit {unlimited | *integer*}]
[-verbose {on | off}]

Description

The tool performs quick synthesis, if needed, during check_design_rules, extract_icl, or when switching to the analysis system mode. Quick synthesis creates new instance, port, pin, and net objects to implement RTL constructs so that they are usable by analysis. Objects created during quick synthesis have their created_by_synthesis attribute set to true.

In the case of RTL that is not synthesizable, the tool issues an error message. However, unlike many other error messages, the actual command that initiates quick synthesis does not fail automatically because the tool cannot predict that the related synthesized module is required for later DRC checks. Due to the nature of hierarchical pre-tracing, the tracing stops at those modules, and it is therefore possible that the tool is unable to synthesize those modules.

Note

You can report modules or instances that are synthesized using the is_synthesized instance or module attributes, respectively. For example:

```
get_module -filter is_synthesized
```

Arguments

- -Parallel_synthesis {maxcpu | *integer*}

  An optional switch and literal or integer pair that sets the number of parallel processes for quick synthesis. When specified with maxcpu, the number of parallel jobs is the number of available CPUs. When specified with an *integer*, the number of parallel jobs is the specified number. You should use a number less than or equal to the number of available CPUs; otherwise, the tool reports a warning. The default is four CPUs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The synthesis engine tries to partition the whole design in multiple groups, which can be synthesized in parallel. After design partition, the engine synthesizes either sequentially or in parallel (faster). If the number of CPUs is less than four, the tool uses only the available CPUs. If you specify *integer* as 1, synthesis is sequential.

The parallel processing is run on different cores of the same CPU and not different CPUs. You can use the sysconf(_SC_NPROCESSORS_ONLN) system command to return the number of cores in a CPU.

- -Skip_synthesis_of_two_dimensional_arrays_larger_than {unlimited | *integer*}

  An optional switch and literal or integer pair that specifies the bit-count threshold for the compilation of memories in a module. If this total bit-count is more than the specified threshold, the module is not compiled and is treated as a black box. The default setting is 4096 bits.

  The purpose of this feature is to avoid synthesizing memory models for which you have loaded the behavioral model in Tessent Shell. If you have large multi-dimensional arrays in your RTL and you want to synthesize them, specify the add_black_boxes -module *module* command on all behavioral memory models and enlarge this threshold to match your requirement. Explicitly black-boxing the memory modules excludes them from synthesis. You can also set the exclude_from_synthesis attribute on the memory modules if you don't want to black box them but only prevent them from being synthesized. During quick synthesis, modules which were automatically black boxed because they contained a multi-dimensional array exceeding the threshold value are listed in a warning similar to the following:

```
// Warning: The following modules were not synthesized because they
// include multi-dimensional arrays whose bit count exceeds the
// following limit: 4096. You can increase this limit with command:
// 'set_quick_synthesis_option
//  -skip_synthesis_of_two_dimensional_arrays_larger_than <int>'
// but you should first check that no memory module is listed below.
// 'SYNC_1RW_32x256_RC_BISR'
// To avoid synthesizing behavioral Verilog views of memories, you
// need to provide:
// - their tessent library description, or;
// - their verilog model along with their tcd_memory_lib
// description;
// If you run logictest DRCs, and do not have a tessent library
// description for you memory, you must provide a Verilog model
// which hides the memory core for all memory instances that have
// embedded test logic. Finally, you can add an explicit black box
// declaration for all memory instances if nothing in
// their models needs to be visible for the logictest DRC.
```

- -blackbox_design_ware_modules on | off

  An optional switch and literal pair that controls whether the tool automatically blackboxes DesignWare modules that contain no definitions. The default is off.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -complete_synthesis on | off

  An optional switch and literal pair that lead to the quick synthesis of all RTL modules in the current design during system mode transition to analysis. The recommendation for the switch and the default is off, because this option can have a significant impact on the performance.

- -trace_allowed_no_source on | off

  An optional switch and literal pair that controls whether the tool automatically performs hierarchical pre-tracing on design pins of ICL instrument input ports, that have an ICL connection_rule_option attribute set to 'allowed_no_source'. It is used only during ICL extraction pre-tracing. The default is off, because this option can have a significant impact on the performance.

- -trace_allowed_no_destination on | off

  An optional switch and literal pair that controls whether the tool automatically performs hierarchical pre-tracing on design pins of ICL instrument output ports, that have an ICL connection_rule_option attribute set to 'allowed_no_destination'. It is used only during ICL extraction pre-tracing. The default is off, because this option can have a significant impact on the performance.

- -backward_tracing_of_control_signal_limit {unlimited | *integer*}

  An optional switch and literal or integer pair that specifies the maximum number of elements the tool traces back during hierarchical pre-tracing for one input cone of a signal setting. The default setting is 100000 traced elements.

  Use this switch to avoid very long runtime due to signal setting backward tracing. You may need to enlarge this limit if you get a DRC error message due to blocked or non-transparent paths. If the tool reaches the limit for a specific signal setting, it reports a warning message as follows:

```
// Warning: The pre-tracing of 'corea/inst1/inst2/inst3/en' used to determine if
there is RTL logic in its fanout cone
//               has reached 100000 nodes during the backward tracing from the
encountered logic cell.
//              For performance reasons, only the first 100000 nodes will be checked
for the presence of RTL logic in the fanin path.
// Note: The previous warning(s) can lead to DRC violations because some relevant RTL
logic may not have been synthesized.
//           If you see DRC violations caused by unsynthesized RTL logic, you can
increase the limit with the following command:
//          'set_quick_synthesis_options -backward_tracing_of_control_signal_limit
integer'
//          or instead request for complete synthesis with the following command:
//          'set_quick_synthesis_options -complete_synthesis on'.
```

- -verbose on | off

  An optional switch that turns on verbose reporting to stdout or the tool's log file. The default is off. Verbose reporting includes the following:

  - activities of hierarchical pre-tracing with quick synthesis support

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- all synthesized modules
- the start point for every synthesized module
- time measurements

Examples

Example 1

The following example sets the number of parallel quick synthesis processes to 3:

```
set_quick_synthesis_options -parallel_synthesis 3
```

Example 2

The following example sets a new limit to avoid synthesis for modules with large multi-dimensional arrays to 128:

```
set_quick_synthesis_options -skip_synthesis_of_two_dimensional_arrays_larger_than  128
```

Example 3

The following example enables synthesis for modules with multi-dimensional arrays of any size by setting the threshold to unlimited:

```
set_quick_synthesis_options \
   -skip_synthesis_of_two_dimensional_arrays_larger_than  unlimited
```

Related Topics

Instance

Port

Net

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_ram_initialization

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup

Specifies whether to initialize RAM and ROM gates that do not have initialization files.

Usage

set_ram_initialization **Uninitialized** | **Random**

Description

The set_ram_initialization command enables the tool to internally generate random values and place them into all uninitialized RAM and ROM gates. This command is useful when simulating random patterns.

Arguments

- **Uninitialized**

  A literal that specifies for the tool to use unknown (X) values to set the memory elements of all RAM and ROM gates which do not have an initialization file. This is the default behavior upon invocation of Tessent FastScan and Tessent TestKompress.

- **Random**

  A literal that specifies for the tool to use random values to set the memory elements of all RAM and ROM gates which do not have an initialization file.

Examples

The following example places random values into all uninitialized RAM and ROM gates:

```
set_ram_initialization random
set_system_mode analysis
create_patterns
```

Related Topics

read_modelfile

write_modelfile

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_random_clocks

Context: dft -edt, patterns -scan

Mode: analysis

Specifies whether Tessent FastScan uses combinational or clock_sequential patterns for random pattern simulation.

Usage

set_random_clocks {***pin_name*** | ***pin_pathname***}…

Description

The set_random_clocks command specifies for Tessent FastScan to either use combinational random patterns or use the pins that you specify for clock_sequential random patterns during simulation.

Combinational random pattern simulation is the default upon invocation. You specify clock_sequential by entering the set_random_clocks command with an ordered set of clock/read/write lines which Tessent FastScan exercises in the same order during the clock_sequential random pattern simulation.

You can reset Tessent FastScan to its invocation default of combinational random pattern simulation by entering the set_random_clocks command without any arguments. Also, when you re-enter the Setup system mode, Tessent FastScan deletes the random clock list and is once again ready for combinational random pattern simulation.

Arguments

- ***pin_name*** | ***pin_pathname***

  A required repeatable string that specifies the names of defined clock, read, and write lines. You must list these pins in the order in which you want Tessent FastScan to exercise them during the clock_sequential random pattern simulation. The default is none, which specifies for Tessent FastScan to perform combinational random pattern simulation.

  Tessent FastScan displays an error if any specified pin is not a defined clock, read, or write line. The tool also displays an error if the number of pins that you list is equal to or exceeds the selected sequential depth.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example runs random pattern simulation with clock_sequential patterns:

```
set_pattern_type -sequential 10
add_scan_groups g1 seqproc.g1
add_scan_chains c1 g1 si so
add_clocks 0 sk1 sk2
set_system_mode analysis
set_random_clocks sk1
add_faults -all
simulate_patterns -source random
```

Related Topics

set_capture_clock

simulate_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_random_patterns

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies the number of random patterns that the tool simulates.

Usage

set_random_patterns ***integer***

Description

The set_random_patterns command specifies how many random patterns you want the tool to simulate.

Arguments

- ***integer***

  A required integer, greater than or equal to 0, that specifies the number of random patterns you want the tool to simulate. The invocation default is 1024.

Examples

The following example sets the number of random patterns to analyze its controllability effects:

```
set_system_mode analysis
set_random_patterns 612
```

Related Topics

set_capture_clock

set_observation_point

simulate_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_read_verilog_options

Context: unspecified, all contexts

Mode: all modes

Sets certain options globally for subsequent read_verilog commands.

Usage

set_read_verilog_options [-allow_enum_relaxation {on | off}]
[-vcs_compatibility {on | off}]
[-unbalanced_celldefine_severity {warning | error}]
[-strict_default_nettype_checks {on | off}]
[-max_for_loop_size *integer*]

Description

The set_read_verilog_options command specifies certain options globally for use in subsequent read_verilog commands.

Arguments

- -allow_enum_relaxation on | off

  An optional switch and literal pair that, when set to “on”, instructs the tool to bypass specified LRM Verilog rules. The default is off.

  The IEEE standard requires that when a driver is assigned to a variable of an enum type, the driver must itself be declared to be a variable of the same type or to be a value that lies within the enumeration set defined for that type unless an explicit cast is used or unless the enum variable is a member of a union. See section 4.10.3 in IEEE 1800-2005.

  If you want to relax this rule, then set this option to “on”.

  Note

  Once you set the -allow_enum_relaxation switch to “on” and load the design, you cannot set this argument to “off” in the same Tessent Shell session. If you attempt to do so, then the tool issues the following error:

  ```
  //  Error: The '-allow_enum_relaxation' option cannot be switched
  //  from 'on' to 'off' without first deleting the design.
  ```

- -vcs_compatibility on | off

  An optional switch and literal pair that controls whether VCS compatibility mode is on or off by default. The default is off.

- -unbalanced_celldefine_severity warning | error

  An optional switch and literal pair that controls whether the tool issues a warning or an error for unterminated “celldefines” or missing “endcelldefine” in a netlist or library file when reading the Verilog. The default is warning.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -strict_default_nettype_checks on | off

  An optional switch and literal pair that enables the tool to issue an error message when a port has no net-type and "`default_nettype is none". For normal nets (non-ports), the tool always issues an error message under that circumstance.

- -max_for_loop_size *integer*

  An optional switch-value pair that specifies the maximum number of loop iterations inside for, while, and forever loop variants during Verilog design elaboration. The loops are mainly inside a function or task definition. The default *integer* number of for-loop iterations is 5000, and there is a maximum threshold of 100000.

  Note

  If the maximum number of for loop iterations is exceeded, the elaboration fails, and you must change the *integer* value.

Related Topics

get_read_verilog_option

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_register_value

Context: all contexts

Mode: all modes

Enables you to specify the register value variable outside of Setup mode.

Usage

set_register_value ***value_name value_string***
[-RAdix {BInary | HEx | OCtal | Decimal}]

Description

When using this command, you must have previously specified the add_register_value command with the -width *integer* argument pair as well as any other required arguments.

See "Serial Register Load and Unload for LogicBIST and ATPG" in the *Tessent Shell User's Manual* for complete information.

Arguments

- ***value_name***

  A required string that specifies register value variable. The variable holds a value to load into the register. The register value variable is used in the procedure file to load the value into a register through the data input pin.

- ***value_string***

  A required state string whose default radix is binary. You can specify Z or X values in addition to numerical values appropriate for the radix.

- -Radix {Binary | Decimal | Octal | Hexadecimal}

  An optional switch and literal that specifies a different radix.

Examples

The following example deletes all register value variables, specifies a register value variable with a decimal radix and register width of 37, and then sets the value of the register.

delete_register_value -all

add_register_value setup_TDI1 0 -radix decimal -width 37

set_register_value setup_TDI1 0000000000000000000000000000000000111

report_register_value

```
//  Register value name               value string              width
//  -------------------  ---------------------------------------  -----
//       setup_TDI1      0000000000000000000000000000000000111   37
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

add_register_value

delete_register_value

report_register_value

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_relevant_coverage

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Determines which sub-class(es) of untestable faults are included in or excluded from fault coverage reports.

Usage

set_relevant_coverage [{-Include | -Exclude} {*fault_class*[.*fault_subclass*] …}]

Description

The set_relevant_coverage command specifies which sub-class(es) of untestable faults are included in or excluded from the relevant coverage reported by the tool.

Run this command without any arguments to report the current settings.

Arguments

- -Include

  An optional switch that includes the specified fault sub-class(es) in the relevant coverage reported by the tool.

- -Exclude

  An optional switch that excludes the specified fault sub-class(es) from the relevant coverage reported by the tool.

- *fault_class*[.*fault_subclass*]

  An optional, repeatable string that specifies the fault class or fault class/sub-class to include or exclude. Table 8-19 lists the fault classes that you can specify with this command and whether they are included in the reported relevant coverage by default. Table 8-20 lists the available fault sub-classes. You can specify a fault class or sub-class either by name or code. For AU faults, you can define a customized sub-class using a unique name and instruct the tool to include (default) or exclude this user-defined sub-class from the relevant fault coverage. For information about using a user-defined sub-calls, refer to "Example 2" on page 2827.

Table 8-19. Fault Classes for Relevant Coverage

Name	Code	Included in Relevant Coverage by Default?
ATPG_UNTESTABLE	AU	yes
REDUNDANT	RE	no
TIED	TI	no
BLOCKED	BL	no
UNUSED	UU	no

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-20. Fault Sub-Classes for Relevant Coverage

Fault Class	Fault Sub-Class	Code	Included in Relevant Coverage by Default?
AU	BLACK_BOXES	BB	yes
	CELL_CONSTRAINTS	CC	yes
	EDT_BLOCKS	EDT	no
	FALSE_PATHS	FP	no
	HOLD_PI	HPI	yes
	HYBRID_LBIST	LBIST	no
	IJTAG	IJTAG	no
	LOW_PIN_COUNT_TEST	LPCT	no
	MASK_PO	MPO	yes
	MULTICYCLE_PATHS	MCP	yes
	ON_CHIP_CLOCK_CONTROL	OCC	no
	PIN_CONSTRAINTS	PC	yes
	SEQUENTIAL_DEPTH	SEQ	yes
	SSN (for SSH instruments only)	SSN	no
	TIED_CELLS	TC	yes
	UNDRIVEN	UDN	yes
	USER-DEFINED	none	yes
	WIRE	WIRE	yes
UU	CONNECTED	CON	no
	FLOATING	FL	no

Examples

Example 1

The following example removes the PIN_CONSTRAINTS and CELL_CONSTRAINTS sub-classes from relevant coverage reported by the tool:

```
set_relevant_coverage -exclude AU.PC AU.CC
set_relevant_coverage
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
 Relevant Coverage Setting Report
-----------------------------------
 Fault Sub-class  Relevant Coverage
----------------  -----------------
AU.UDN                          yes
AU.BB                           yes
AU.WIRE                         yes
AU.EDT                           no
AU.PC                            no
AU.TC                           yes
AU.CC                            no
AU.FP                            no
AU.MCP                          yes
AU.HPI                          yes
AU.MPO                          yes
AU.SEQ                          yes
AU.LBIST                         no
AU.OCC                           no
AU.IJTAG                         no
AU.LPCT                          no
UU.FL                            no
UU.CON                           no
TI                               no
BL                               no
RE                               no
-----------------------------------
```

Example 2

The following example shows how to use a user-defined fault sub-class to exclude specific pins from reporting. This example is based on the following fault list contained in *flist.au.usr1*:

```
//    type    code    pin_pathname
//    ----   ------   ------------------------------------
      0     AU.USR1      /si7
      0     EQ.USR1      /sf_1_7/SI
      1     AU.USR1      /si7
      1     EQ.USR1      /sf_1_7/SI
      0     AU.USR1      /si8
      0     EQ.USR1      /sf_1_8/SI
      1     AU.USR1      /si8
      1     EQ.USR1      /sf_1_8/SI
```

This fault list creates a user-defined sub-class for AU faults on two pins for both SA0 and SA1.

```
read_faults flist.au.usr1 -retain
set_relevant_coverage -exclude AU.USR1
report_statistics
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
                              Statistics Report
                               Stuck-at Faults
-------------------------------------------------------------------------
Fault Classes                                      #faults     #faults
                                                    (total)  (total relevant)
----------------------------------------- ------------- ---------------
  FU (full)                                           1397        1389
  ----------------------------------------- ------------- ---------------
  UC (uncontrolled)                                 27 ( 1.93%)   same ( 1.94%)
  DS (det_simulation)                              728 (52.11%)   same (52.41%)
  DI (det_implication)                             315 (22.55%)   same (22.68%)
  UU (unused)                                      234 (16.75%)   same (16.85%)
  AU (atpg_untestable)                              93 ( 6.66%)     85 ( 6.12%)
-------------------------------------------------------------------------
Fault Sub-classes
-------------------------------------------------------
  AU (atpg_untestable)
    BB (black_boxes)                                12 ( 0.86%)   same ( 0.86%)
    PC* (pin_constraints)                           20 ( 1.43%)   same ( 1.44%)
    TC* (tied_cells)                                53 ( 3.79%)   same ( 3.82%)
    USR1                                             8 ( 0.57%)     deleted
*Use "report_statistics -detailed_analysis" for details.
-------------------------------------------------------------------------
Coverage
  -------------------------------------------------------
  test_coverage                                     89.68%            90.30%
  fault_coverage                                    74.66%            75.09%
  atpg_effectiveness                                98.07%            98.06%
```

Related Topics

report_statistics

set_fault_subclass_analysis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_run_synthesis_options

Context: unspecified, all contexts

Mode: all modes

Specifies the default values for options that can be set for the run_synthesis command.

Usage

set_run_synthesis_options [-default_synthesis_tool {dc_shell | genus}]
[-synthesis_output_directory *directory*]
[***synthesis_tool*** [-compilation_options *option_list*]
[-pre_compilation_commands *command_list*]
[-post_compilation_commands *command_list*] [-startup_file *filename*]
[-command_name *command_name_path*]]

Description

The options that are defined with set_run_synthesis_options are overridden if the are explicitly defined with the run_synthesis command.

Arguments

- -default_synthesis_tool dc_shell | genus

  An optional switch and value pair that specifies which synthesis tool the run_synthesis command uses. The default value is “dc_shell”. Future releases may support other synthesis tools.

- -synthesis_output_directory *directory*

  An optional switch and value pair that specifies the synthesis output directory. The default directory is *synthesis_outdir* and is located in the Tessent Shell invocation directory.

- ***synthesis_tool***

  A required value that specifies the synthesis tool when setting one or more tool-specific options. Possible values for ***synthesis_tool*** are:

  dc_shell

  genus

  The tool-specific options that require this switch are:

  - -compilation_options
  - -pre_compilation_commands
  - -post_compilation_commands
  - -startup_file
  - -command_name

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -compilation_options *option_list*

  An optional switch and option list that specifies additional options to be added to the synthesis compile command used in the *<synthesis_tool>.synthesis_tcl*. An example would be the -boundary_optimization option for dc_shell.

- -pre_compilation_commands *command_list*

  An optional switch and command list that adds the provided commands to the *<synthesis_tool>.synthesis_tcl* so that they are executed before the compilation command is run.

- -post_compilation_commands *command_list*

  An optional switch and command list that adds the provided commands to the *<synthesis_tool>.synthesis_tcl* so that they are executed after the compilation command is run.

- -startup_file *filename*

  An optional switch and file path that enables you to explicitly specify the startup file that is used by the synthesis tool. The file specified is copied into the actual synthesis directory.

  Note

  If a startup file is not specified and a *.synopsys_dc.setup* file exists in the Tessent run directory, it is automatically copied into the synthesis run directory. If relative paths are utilized within this setup file, it should be noted the synthesis run directory is two levels below where Tessent Shell is launched and these paths should account for this execution point.

- -command_name *command_name_path*

  Defines the command name to invoke the synthesis tool. The default for dc_shell is 'dc_shell' and the command is assumed to be available in the search path of the operating system.

Examples

The following examples of set_run_synthesis_options setting some of the possible options.

Example 1

This example adds the optimize_registers command before compile in the *dc_shell.synthesis_tcl* used to synthesize the test logic.

```
SETUP> set_run_synthesis_options dc_shell -pre_compilation_commands \
          optimize_registers
```

Example 2

This example adds the -map_effort high option to the compile command in the *dc_shell.synthesis_tcl* used to synthesize the test logic.

```
SETUP> set_run_synthesis_options dc_shell -compilation_options \
          {-map_effort high}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

check_synthesis

get_run_synthesis_options

report_run_synthesis_options

run_synthesis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_scan_chain_options

Context: pattern -scan

Mode: setup

Specifies options for managing scan chains.

Usage

set_scan_chain_options {{**-chain_name** ***chain_name...*** |
-INStance ***instance_name*** **-Block_chain_index_list** ***block_chain_index_list*** ...}
[-decompressor_to_scan_in_inversion {on | off}]
[-scan_out_to_compactor_inversion {on | off}]} |
-simple_chain_names {**on** | **off**}

Description

The set_scan_chain_options command sets scan chain options that deal specifically with the hybrid TK/LBIST flow. It also sets scan chain options that are independent of this flow.

Hybrid TK/LBIST Options

When working with the hybrid TK/LBIST flow, use the set_scan_chain_options command with the -chain_name or -instance switch. These switches are available in setup mode when LBIST is on. In LogicBIST mode, no hardware rediscovery is performed. As a result, the tool can be unaware of inversions that were inserted by a third-party tool between the scan chains and the PRPG/phase shifter on the input side and the compactor and the scan chains on the output side. Before LogicBIST fault simulation, you can specify the set_scan_chain_options command to force the inversions and proceed in such situations.

Using these switches with this command has no effect on ATPG pattern generation.

EDT finder can automatically rediscover the inversions between the EDT logic and scan chains, which means you can use it to export the inversions and reuse them for LogicBIST fault simulation. Use either of the following commands in analysis mode with EDT enabled (after passing K-rules) to find the inversions:

- report_scan_chains -inversions — This command reports the relevant inversions in the decompressor_inversion and compactor_inversion columns.
- write_core_description — The TCD includes information about the inversions in the decompressor_to_scan_in_inversion and scan_out_to_compactor_inversion properties within ScanChain wrappers. You can read back the TCD before performing fault simulation as described in "Fault Simulation When There Are Inversions" in the *Hybrid TK/LBIST Flow User's Manual*.

The test patterns in -mode_internal continue to be written out with respect to scan chain input and scan chain output. If an inversion is specified after the SCO, the unload values do not reflect the inversion. However, MISR values take that inversion into account.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

General Options

For general scan chain options, use the -simple_chain_names switch. This switch enables you to simplify the scan chain names for ATPG. Refer to the description of this switch for more details.

Arguments

- **-chain_name *chain_name...***

  Required switch and repeatable string that specifies the name of a scan chain in the current design.

- **-INStance *instance_name* -Block_chain_index_list *block_chain_index_list***...

  Required set of switches and values that specifies a core instance and the list of chain indices starting with "1" on the EDT controller. When this is used, the masking applied to the specified chain is OX. Use this option in conjunction with the -USed_chains switch.

- -decompressor_to_scan_in_inversion {on | off}

  Optional switch and literal pair that specifies whether the tool identifies inversions between the EDT decompressor and the SCI pin. The default is off.

- -scan_out_to_compactor_inversion {on | off}

  Optional switch and literal pair that specifies whether the tool identifies inversions between the SCO pin and the EDT compactor. The default is off.

- **-simple_chain_names** {**on** | **off**}

  Required switch and literal pair that specifies whether to use simplified names for scan chains in ATPG. By default ("off"), the tool uses descriptive names for scan chains. If you specify this option as "on," the tool instead uses <*chain_id*> (a single number) as chain names. When you use an ATPG core hierarchically, the lower-level name is formed from the <*core_instance_id*> and the <*chain_id*> separated by two underscores; for example, 1__1, 1__2, … 2__1, 2__2, ….

Examples

The following example makes Tessent Shell aware that there are inversions associated with both the SCI and SCO on chain3.

```
set_scan_chain_options -chain_name chain3 -decompressor_to_scan_in_inversion
on -scan_out_to_compactor_inversion on
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_scan_enable

Context: dft -scan, dft -test_points

Mode: setup, analysis (dft -test_points context)

Assigns scan_enable signals to specific scan chains.

Usage

Context: dft -scan

set_scan_enable [*scan_enable_pin_pathname*] [-isolate]
[*primary_input*] [-active {high | low}]
[-single_global_scan_enable {on | off}]
[-cluster_name *cluster_name*] [-power_domain_name *power_domain_name*]
[-clock *clock_pin*]

Context: dft -test_points

set_scan_enable [*scan_enable_pin_pathname*] [-isolate]
[*primary_input*] [-active {high | low}]
[-cluster_name *cluster_name*] [-power_domain_name *power_domain_name*]
[-clock *clock_pin*]

Description

The set_scan_enable command can be issued in a sequence to either refine the scan enable signals assignments or overwrite the previous assignments (see the example section for this command). In general, the following rules are applied to determine how signal assignments are affected by subsequent commands:

- The most recent command that assigns a scan_enable signal takes precedence.
- If the most recent command operates on a disjoint set of scan chains, then the previous scan enable signal assignments remain intact.
- If the most recent command operates on previously specified scan chains, then previous scan_enable signal assignments are overwritten.

When using separate scan enable signals for input, output, and core cells, for every set of constraints (clock domain, power domain, and/or cluster) defined with the set_scan_enable command, all three scan enable signals must be defined, otherwise system mode transition fails. For example, when the input scan enable for clock domain clk1 is defined, then the core and output scan enables for that clock domain must be defined.

Note

It is recommended that you do not use separate scan enable signals for input and output wrapper cells, either by setting the -single_global_scan_enable to off explicitly or by defining the input and output scan enables manually, as it is not fully compatible with hierarchical scan insertion.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- *scan_enable_pin_pathname*

  An optional string that specifies a pin pathname for the scan_enable signal driver. The specified pin can be either a top-level scan enable port or an internal instance pin (connection node). If an internal instance pin is specified, it must trace back to a primary input via a simple path (only inverters or buffers) or the primary_input argument.

  If no scan_enable signal driver is specified, the default scan enable name is used (scan_en).

- -isolate

  An optional string that specifies to isolate new fanouts of the specified scan enable signal. Each new fanout connection is gated by an AND or NOR gate and controlled by the global test enable signal. This switch is applicable to a scan enable signal driven by a top-level port or a top-level internal instance pin only.

- *primary_input*

  An optional string that specifies a top-level scan_enable port. This argument supplies a primary input/top-level port for an internal instance pin specified by the scan_enable_pin_pathname argument. The specified top-level port is used when generating the ATPG dofile and test procedure files. If the specified top-level port does not exist, it is created. The specified top-level port must be a primary input port.

- -active {high| low}

  An optional switch and literal pair that specifies whether the scan_enable signal is active low or high. The default setting is high.

- -single_global_scan_enable {on | off}

  An optional switch and literal pair that specifies to use a single scan enable signal for input wrapper chains, output wrapper chains and core chains. This is the default. The input wrapper chains scan enable signal is generated by ORing the specified or the default scan enable signal with INTEST mode test enable signal. The output wrapper chains scan enable signal is generated by ORing the specified or default scan enable signal with EXTEST mode test enable signal. If this switch is disabled, the tool uses three separate scan enable signals: scan_en_in for input wrapper chains, scan_en_out for output wrapper chains, and scan_en for core chains. Other switches are not accepted when the -single_global_scan_enable switch is used.

  See the Note above in the Description section recommending not to use separate scan enable signals for input and output wrapper cells.

- -cluster_name *cluster_name*

  An optional switch and string pair that specifies to assign a specified scan enable signal to scan chains in the specified cluster.

- -power_domain_name *power_domain_name*

  An optional switch and string pair that specifies to assign a specified scan enable signal to scan chains in the specified power domain.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -clock *clock_pin*

  An optional switch and string pair that associates the specified scan enable signal with the specified clock (clock domain). Clock_pin can be either an existing top level port (primary input pin) or an existing internal pin pathname.

Examples

Example 1

The following example defines two scan clusters: rx_clust and tx_clust, by assigning the cluster_name attribute to the selected instances. Each cluster has its dedicated scan enable defined with the set_scan_enable command. The first set_scan_enable command makes all scan chains controllable via the sen scan enable signal. The second set_scan_enable command refines the first command and makes scan chains of rx_clust controllable via the sen_rx scan enable signal. The third set_scan_enable command further refines the first command and makes scan chains of tx_clust controllable via the sen_tx scan enable signal.

```
set_scan_enable sen
set_scan_enable sen_rx -cluster_name rx_clust
set_scan_enable sen_tx -cluster_name tx_clust

set_system_mode analysis
set_attribute_value [get_scan_elements -below_instances i_rx_phy] \
      -name cluster_name -value rx_clust
set_attribute_value [get_scan_elements -below_instances i_tx_phy] \
      -name cluster_name -value tx_clust
add_scan_mode unwrapped
analyze_scan_chains
report_scan enable
```

The last command displays the following result:

```
//    command: report_scan_enable
Scan mode 'unwrapped' scan enables:
---------------------------------------------------------------------------
Primary Input  Internal Connection Node  Scan Chain  Scan Partition
---------------------------------------------------------------------------
/sen_rx        --                        chain0      rx_clust
---------------------------------------------------------------------------
/sen_tx        --                        chain1      tx_clust
---------------------------------------------------------------------------
/sen           --                       chain2      default_scan_partition
---------------------------------------------------------------------------
```

The second set_scan_enable command operates on a set of scan chains that is entirely within the set specified in the first command; therefore, the scan chains that are in both sets gets the most recent assignment. The third set_scan_enable command operates on a set of scan chains that is disjoint from the set in the second command but is entirely contained within the first set; therefore, the scan chains that are in both the first and the third sets get the most recent assignment.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example defines two scan clusters: rx_clust and tx_clust. The first set_scan_enable command makes all scan chains controllable via the sen scan enable signal. The second set_scan_enable command specifies an internal pin, rx_sen_reg/Q, associated with the primary input, sen_rx, as a driver of the scan enable signal controlling all scan chains of rx_clust. The third set_scan_enable command specifies an internal pin, tx_sen_reg/Q, associated with the primary input, sen_tx, as driver of the scan enable signal controlling all scan chains of tx_clust.

```
add_clock 0 clk2
set_scan_enable sen
set_scan_enable rx_sen_reg/Q sen_rx -cluster_name rx_clust
set_scan_enable tx_sen_reg/Q sen_tx -cluster_name tx_clust
set_scan_enable sen_clk2 -clock clk2
```

The last set_scan_enable command specifies that all scan chains in the clk2 clock domain are controllable by the sen_clk2 scan enable signal. The second and the third set_scan_enable commands overwrite the assignments of the first set_scan_enable command affecting chains that are in rx_clust and tx_clust. Since the second and the third set_scan_enable commands operate on disjoint sets, they do not affect their assignments.

The fourth command overwrites some of the assignments of the second and the third commands for scan chains that are in rx_clust and tx_clust and also in the clkInput clock domain. If you want to restrict the clock domain's assignments to a specific cluster, the -cluster_name and -clock options should be issued in the same set_scan_enable call.

Related Topics

add_input_constraints

report_scan_enable

set_scan_signals

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_scan_insertion_options

Context: dft -scan

Mode: setup, analysis

Specifies the high-level options for the scan insertion process.

Usage

```
set_scan_insertion_options
    [ -attribute_set_for_chain_assembly { attribute_list } ]
    [ -single_attribute_set_chains { on | off } ]
    [ -single_class_chains { on | off } ]
    [ -single_clock_domain_chains { on | off } ]
    [ -single_clock_edge_chains { on | off } ]
    [ -single_cluster_chains { on | off } ]
    [ -single_power_domain_chains { on | off } ]
    [ -single_wrapper_type_chains { on | off } ]
    [ -port_index_start_value int ]
    [ -port_scalar_index_modifier int ]
    [ -chain_length { unlimited | int } ]
    [ -chain_count { unlimited | int} ]
    [ -si_port_format format_str ]
    [ -so_port_format format_str ]
    [ -si_timing{ leading_edge | trailing_edge | any_edge } ]
    [ -so_timing { leading_edge | trailing_edge | any_edge } ]
    [ -lockup_cell_type { flop | latch } ]
    [ -si_lockup_cell_type { flop | latch } ]
    [ -so_lockup_cell_type { flop | latch } ]
    [ -unused_chain_connection_handling { auto | insert_pipelining_flop |
     tie_off_chain_output | dont_touch } ]
    [ -si_pipelining { all_chains | off | wrapper_chains_only } ]
    [ -si_pipelining_type { non_scan_flop | holding_scan_flop } ]
    [ -enable_scan_cell_mapping { on | off } ]
    [ -se_pipeline_count { 1 | 0 } ]
    [ -port_bus_range { ascending | descending } ]
    [ -fully_encoded_scan_mode_logic { on | off } ]
```

Description

Use this command to specify the high-level options for the scan insertion process.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Many of the options of this command are also available for the add_scan_mode and create_scan_chain_family commands. If you change any of these common options using the set_scan_insertion_options command, the tool issues a warning that the changes only apply when creating a new scan mode or chain family, and have no effect on existing scan modes or chain families.

Arguments

- -attribute_set_for_chain_assembly { *attribute_list* }

  An optional switch used to specify one or more string attributes to control scan chain assembly. When you use this switch and specify several string attributes, the tool creates clusters for every value combination of the specified attributes. For example, assume that you have defined the following quadrant and testpoint_type name attributes and that all scan elements are attributed accordingly:

```
>register_attribute \
-name quadrant \
-value_type string \
-object_types scan_element \
-enum { NE NW SE SW }

>register_attribute \
-name testpoint_type \
-value_type string \
-object_types scan_element \
-enum { control observation }
```

  Note that scan elements that are not test points would not be assigned a “testpoint_type” attribute value, whereas all scan elements would be assigned a “quadrant” attribute value.This implies that each scan element would have one of the following attribute value combinations:

Table 8-21. Attribute Value Combinations

“quadrant” attribute value	“testpoint_type” attribute value
NE	control
NE	observation
NE	--
NW	control
NW	observation
NW	--
SE	control
SE	observation

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-21. Attribute Value Combinations (cont.)

"quadrant" attribute value	"testpoint_type" attribute value
SE	--
SW	control
SW	observation
SW	--

Accordingly, you can set the default behavior for modes and scan chain families to create separate scan chains for each of the above 12 attribute value combinations as follows:

```
>set_scan insertion_options\
-attribute_set_for_chain_assembly { quadrant testpoint_type }
```

- -single_attribute_set_chains { on | off }

  An optional switch and literal pair that restricts a scan chain to a single set of attribute values. The default is on.

- -single_class_chains { on | off }

  An optional switch and literal pair that controls whether to mix wrapper and non-wrapper cells in a single chain. The default is off, so wrapper and internal cells can be distributed onto the same chains.

- -single_clock_domain_chains { on | off }

  An optional switch and literal pair that restricts a scan chain to a single clock domain. The default is off.

- -single_clock_edge_chains { on | off }

  An optional switch and literal pair that restricts a scan chain with flops having a single edge. The default is off.

- -single_cluster_chains { on | off }

  An optional switch and literal pair that restricts a scan chain to a single cluster using the built-in attribute "cluster_name". The default is on. For usage examples, see Scan Mode and Scan Insertion for Unwrapped Core sections in the *Tessent Scan and ATPG User's Manual*. For attribute details, see the table "Built in Attributes on Scan Element (SE) Object Type" in the Scan Element Object Type section of the *Tessent Shell Reference Manual*.

- -single_power_domain_chains { on | off }

  An optional switch and literal pair that restricts a scan chain to a single power domain using the built-in attribute "power_domain_island". The default is on. For attribute details, see the table "Built in Attributes on Scan Element (SE) Object Type" in the Scan Element Object Type section of the *Tessent Shell Reference Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -single_wrapper_type_chains { on | off }

  An optional switch and literal pair that specifies that "-single_class_chains ON" controls whether to mix input wrapper and output wrapper cells in the same chain. The default is off.

- -port_index_start_value *int*

  The starting index is 0 by default, but can be overridden by this option.

- -port_scalar_index_modifier *int*

  An optional switch that specifies the incremental value to add to the index portion (%d) of scalar scan port names. The default is 1

- -chain_length { unlimited | *int* }]

  An optional integer that specifies the target length of scan chains for a particular mode. By default, the no limit (unlimited) is imposed on the chain length.

- -chain_count { unlimited | *int* }

  An optional integer that specifies the number of scan chains that should be constructed for the specified mode. By default, the tool does not impose a limit (unlimited) on the chain count; otherwise, it sets the chain count to the number of SI or SO connections specified by the -si_connections or -so_connections previously set with the add_scan_mode command.

- -si_port_format format_str | -so_port_format *format_str*

  The tool sets the default si_port_format or so_port_format to "ts_%s_si/so[%d]" when there is more than one scan mode; otherwise, it defaults to "ts_si/so[%d]". It replaces the mode placeholder "%s" by the chain mode name, whereas it substitutes the index placeholder "%d" for the chain index. You may use a precision number when specifying the index placeholder; for instance, the format "lsi_si%2d" would create port names such as "lsi_si00" and "lsi_si01". The tool automatically inserts leading zero's when needed, making "%2d" implicitly equivalent to "%02d".

- -si_timing { leading_edge | trailing_edge | any_edge }

  An optional string and literal pair that provides the ability to control the capture edge of the first scan cell in a chain. The tool inserts a lockup (re-timing) cell between the scan chain input pin and the first scan cell in the chain to control the capture edge of the first cell as the leading or trailing edge. It does not insert a lockup cell if you use “any_edge” as the argument. The default edge type is “leading_edge”.

- -so_timing { leading_edge | trailing_edge | any_edge }

  An optional string and literal pair that provides the ability to control the change edge of the last scan cell in a chain for this scan mode. The tool inserts a lockup (re-timing) cell between the scan chain output pin and the last scan cell in chain to control the change edge of the last cell as leading or trailing edge. It does not insert a lockup cell if you use “any_edge” as the argument. The default edge type is “trailing_edge”.

- -lockup_cell_type { flop | latch }

  An optional switch and string that controls the type of lockup (re-timing) cell to use between timing-constrained scan elements of new scan chains. The default cell type is latch.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -si_lockup_cell_type { flop | latch }

  An optional switch and string that controls the type of lockup (re-timing) cell to use between the scan chain input pin and the first scan cell. The default cell type is latch.

- -so_lockup_cell_type { flop | latch }

  An optional switch and string that controls the type of lockup (re-timing) cell to use between the scan chain output pin and the last scan cell. The default cell type is latch.

  Note

  By default, the tool adds a pipeline flop in front of the wrapper chains to facilitate launching a transition from the first scan cells in the wrapper chains.

- -unused_chain_connection_handling {auto | insert_pipelining_flop | tie_off_chain_output | dont_touch}

  An optional switch and literal pair that selects how to handle unused scan in and scan out connections. Options are the following:

  auto — selects “insert_pipelining_flop” for EDT controllers with bypass chains, “tie_off_chain_output” for all other EDT controllers, and “dont_touch” for other connections. This option is the default.

  insert_pipelining_flop — inserts scan chains composed of a single pipelining flop.

  tie_off_chain_output — ties off the so_connection or the EDT compactor input when you specify an EDT instance.

  dont_touch — do not touch the unused scan chain connections.

- -si_pipelining { all_chains | off | wrapper_chains_only }

  An optional switch and literal that controls whether the tool inserts a pipelining flop at the beginning of each chain if the first scan element of the chain does not have the attribute “no_si_pipelining” set and is not an OCC segment. When set to all_chains, the tool adds a pipelining flop to all scan chains. When set to wrapper_chains_only, the tool adds a pipelining flop only to chains of internal or external mode types and composed of wrapper scan elements. The default is wrapper_chains_only when you specify “-se_pipeline_count 0” (the default value for this switch), but the default changes to all_chains when you specify “-se_pipeline_count 1”.

- -si_pipelining_type { non_scan_flop | holding_scan_flop }

  An optional switch and string that controls whether to implement pipelining using a simple non-scan data flop or a holding flop constructed using a scan flop with its output connected to its data input. The default is holding_scan_flop.

- -enable_scan_cell_mapping { on | off }

  An optional switch and literal pair that controls the mapping from non-scan to scan cells. The default is on.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -se_pipeline_count { 1 | 0 }

  An optional switch and literal pair used to implement launch-off-shift (LOS) at-speed testing. It specifies the number of pipelining stages on the scan_enable signal path(s) for each used clock domain. The default value is “0”.

  Unlike other options for this command, you cannot override it with the add_scan_mode and create_scan_chain_family commands.

- -port_bus_range { ascending | descending }

  An optional switch and literal pair that determines whether created bus ports are created with ascending range (for example, [0:31]) or descending range (for example,[31:0]). The default is descending.

- -fully_encoded_scan_mode_logic { on | off }

  An optional switch and literal pair that controls how the tool encodes the multi-mode scan paths. When set to “on”, the scan insertion tool does not use 2x1 multiplexers to encode the multi-mode scan paths. Multi-mode encoding is done exclusively with AND-OR logic, forcing zero-shifting when none of the modes is active. The default value is “off”.

Examples

Example 1

The following example demonstrates using the set_scan_insertion_options command to disable restricting a scan chain to a single power domain using the attribute “power_domain_name” and set the type of lockup (re-timing) cell between the scan chain output pin and the last scan cell to a flop:

```
set_scan_insertion_options -single_power_domain_chains off -lockup_cell_type flop
```

Example 2

If you run the command without any arguments, it reports all available options:

```
set_scan_insertion_options
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Option                          | Type |                Value
//  --------------------------------+------+--------------------
//  single_clock_domain_chains      | BOOL |                  off
//  single_clock_edge_chains        | BOOL |                  off
//  single_cluster_chains           | BOOL |                   on
//  single_power_domain_chains      | BOOL |                  off
//  single_wrapper_type_chains      | BOOL |                  off
//  single_class_chains             | BOOL |                  off
//  tie_off_set_reset               | BOOL |                   on
//  fully_encoded_scan_mode_logic   | BOOL |                  off
//  enable_scan_cell_mapping        | BOOL |                   on
//  port_bus_range                  | ENUM |           descending
//  si_pipelining_type              | ENUM |   holding_scan_flop
//  si_pipelining                   | ENUM | wrapper_chains_only
//  unused_edt_pin_handling         | ENUM |                 auto
//  so_lockup_cell_type             | ENUM |                latch
//  si_lockup_cell_type             | ENUM |                latch
//  lockup_cell_type                | ENUM |                latch
//  so_timing                       | ENUM |       trailing_edge
//  si_timing                       | ENUM |        leading_edge
//  chain_count                     | INT  |           unlimited
//  chain_length                    | INT  |           unlimited
//  port_scalar_index_modifier      | INT  |                   1
//  port_index_start_value          | INT  |                   0
//  so_port_format                  | STR  |        ts_%s_so[%d]
//  si_port_format                  | STR  |        ts_%s_si[%d]
```

Related Topics

add_scan_mode

create_scan_chain_family

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_scan_signals

Context: dft -scan, dft -test_points

Mode: setup

Sets the input control signals' drivers used by the insert_test_logic command.

Usage

set_scan_signals [-TEn *pathname*] [-int_test_enable *pathname*]
[-ext_test_enable *pathname*] [-se_pipeline_enable *pathname*]
[-TClk *pathname*] [-SET *pathname*] [-RESet *pathname*]
[-REAd *pathname*] [-Write *pathname*]
[-Muxed | -Disabled] [-Active {High | Low}]

Description

The set_scan_signals command specifies the drivers of input control signals for the insert_test_logic command to use. Additional control pins are specified with set_scan_enable, set_bidi_gating, and set_tristate_gating commands.

The tool uses the write control and read control pins when inserting test logic for RAMs. DRC verifies whether the RAMs can be held off when the write control pin is off. No test logic is inserted for RAMs that can be held off.

If the new scan design is intended for Tessent FastScan/Tessent TestKompress, and the write clock input of the RAM is not controllable, it is multiplexed out by a new write clock. The selector to the mux is the test enable pin.

You can use instance pin pathnames to define the test enable and clock pin names. The instance pathnames must trace back through a simple path (inverters or buffers only) to a primary input/output of the design, or you must manually develop the test procedure file and the dofile before running ATPG.

Arguments

- -TEn *pathname*

  An optional switch and string pair that specifies a name for the test enable pin. By default, the test enable pin is named test_en.

  Use the -Active switch to specify the active state of the Test enable signal.

- -int_test_enable *pathname*

  An optional switch and string pair that specifies a name for the test enable pin in the INTEST mode. By default, the test enable pin is named int_ltest_en.

- -ext_test_enable *pathname*

  An optional switch and string pair that specifies a name for the test enable pin in the EXTEST mode. By default, the test enable pin is named ext_ltest_en.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -se_pipeline_enable *pathname*

  An optional switch and string pair that specifies a name for the scan_enable pipelining enable pin. By default, the scan_enable pipelining enable pin is named se_pipeline_en. Use the -Active switch to specify the active state of the scan_enable pipelining enable signal.

- -TClk *pathname*

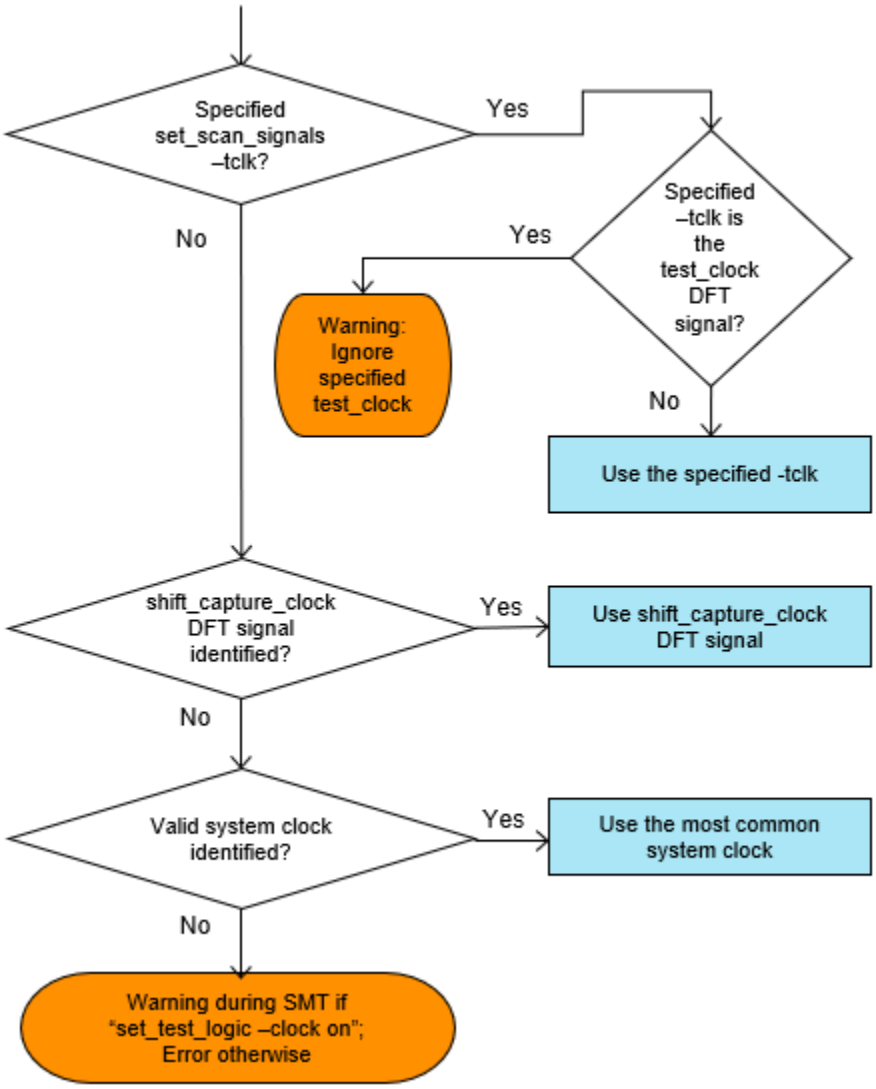
  An optional switch and string pair that specifies a pathname for the test clock pin for all scan types. If no test clock pin exists, one is created using the specified name. By default, the new test clock pin is named test_clk.

  If no test clock pin is specified, an existing clock pin from the design is used. Eligible clock pins must be defined with the add_clocks command and are capable of capturing data in a sequential cell without test logic in the clock path.

  The following figure shows a simplified decision tree that drives the selection process of a clock for scan insertion while using the dft -scan and dft -test_points contexts.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 8-12. Clock Selection Scheme



- -SET *pathname*

  An optional switch and string pair that specifies a pathname for the set pin used for the flip-flop or latch when inserting test logic. Use the -Active switch to specify the active state of the set signal. By default, the tool names the set pin "scan_set".

  Note

  You must specify the name of the currently defined test enable control signal with the -TEn switch if you want to use this switch with the -Disabled switch.

- -RESet *pathname*

  An optional switch and string pair that specifies a pathname for the reset pin used for the flip-flop or latch when inserting test logic. Use the -Active switch to specify the active state of the reset signal. By default, the tool names the reset pin "scan_reset".

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

You must specify the name of the currently defined test enable control signal with the -TEn switch if you want to use this switch with the -Disabled switch.

- -REAd *pathname*

  An optional switch and string pair that specifies a pathname for the read control pin used for RAM when inserting test logic. By default, the tool names the read control pin “read_clk”.

  Note

  You must specify the name of the currently defined scan enable control signal with the set_scan_enable command if you want to use this switch with the -Disabled switch.

- -Write *pathname*

  An optional switch and string pair that specifies a pathname for the write control pin used for RAM when inserting test logic. By default, the tool names the write control pin “write_clk”.

  Note

  You must specify the name of the currently defined scan enable control signal with the set_scan_enable command if you want to use this switch with the -Disabled switch.

- -Muxed | -Disabled

  An optional switch that determines how the specified set, reset, write control, or read control lines are gated. These options can only be used with -Set, -Reset, -Write, and -Read switches specified in the same command. Options include the following:

  -Muxed — Multiplexes all set and reset pins with the original signal. Only set and reset pins defined as clocks are affected. Default setting.

  -Disabled — Uses an AND gate with the test enable signal to disable the set and reset inputs of flip-flops and the SEN type scan enable signal to disable the write and read clocks.

- -Active High | Low

  An optional switch and literal pair that specifies the active state of the Test enable, Set or Reset signals. Within a single command, you can only use the -Active switch once.

  The -Active switch applies to any of the above signals that are specified in the command with the -Ten, -Set, and -Reset switches. By default, all these signals are active on high. The -Active switch is not applicable to any of the other signals specified in this command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example specifies an internal pin, modxyz/nand02/test_en_L, as the driver of the Test enable signal to be used during the scan insertion.

```
add_clocks 0 clock
set_scan_signals -ten modxyz/nand02/test_en_L -active low
set_system_mode analysis
insert_test_logic
```

Example 2

The following example shows how to set different controls with successive set_scan_signals commands.

```
set_scan_signals -ten TEST_MODE
set_scan_signals -reset RESET_L -active low
set_scan_signals -write ATPG_WRT_INH -read ATPG_READ_INH
set_scan_signals -set ATPG_SET -disabled
```

Related Topics

insert_test_logic

set_bidi_gating

set_scan_enable

set_tristate_gating

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_screen_display

Context: unspecified, all contexts

Mode: all modes

Specifies whether the tool writes the transcript to the session window.

Usage

set_screen_display **ON** | **OFf**

Description

If you create a logfile with the set_logfile_handling command, you may want to disable the tool from writing the same information to the session transcript window.

Arguments

- **ON**

  A literal that enables the tool to write the session information to the transcript window. This is the default behavior upon invocation of the tool.

- **OFf**

  A literal that disables the tool from writing any of the session information to the transcript window, including error messages.

Examples

The following example shows how to use the logfile functionality to capture the transcript in a file and then disables the tool from writing the transcript to the display.

```
set_logfile_handling /user/design/setup_file
set_screen_display off
```

Related Topics

set_logfile_handling

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_shadow_check

Context: dft -edt, dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis (dft -scan context only)

Specifies whether the tool identifies sequential elements as “shadow” elements during scan chain tracing.

Usage

set_shadow_check {**ON** | **OFf** | **REstricted**} [-ALLow_cross_clock_domains {ON | OFf}]

Description

You can use the set_shadow_check command to disable the checking and avoid corresponding error messages. This prevents identification of any non-scan sequential element as a shadow element.

In the dft -scan context, by default, the tool identifies shadow cells (set_shadow_check on). During scan insertion, the shadow cells are not targeted for non-scan to scan mapping, and do not directly become part of the scan chains. The shadow cell is always loaded with the same value that was loaded into the master cell during scan chain shifting. There is a trade-off between area overhead during scan insertion and maximum test coverage. The tool default is targeted at minimizing area overhead. In order to target the shadow flops for non-scan to scan mapping, and make them separate cells in the scan chains you must disable shadow checking prior to running the DRC rules (in setup mode).

Arguments

- **ON**

  A literal that enables shadow checking. This is the initial state upon invocation of the tool.

- **OFf**

  A literal that disables shadow checking.

- **REstricted**

  A literal that disables the tool’s identification of shadow cells that require independent shifts in certain pattern formats (for example, parallel Verilog format).

  Note

  An independent shift results from a one-shift “apply shift” statement in the load_unload procedure. For more information, refer to the description of the P19 rule.

- -ALLow_cross_clock_domains {ON | OFf}

  An optional switch and literal pair that enables consideration of cross-clock domain state elements as shadow cells. This option can increase test coverage, but you should turn on this

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

switch only when you are sure that the clocks are balanced and the shift values to the cross-domain cells are reliable.

Examples

The following example disables shadow checking:

```
set_shadow_check off
```

Related Topics

report_environment

set_drc_handling

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_shift_register_identification

Context: dft -scan, dft -test_points

Mode: setup

Enables or disables shift register identification. The default is ON.

Usage

set_shift_register_identification {**ON** | **OFF**}
[-ALLOW_COMBINATIONAL_logic_between_registers {ON | OFF}]

Description

For more information about automatic identification of shift registers, see "Automatic Recognition of Existing Shift Registers" in the *Tessent Scan and ATPG User's Manual.*

Arguments

- **ON**

  A required literal that enables shift registration identification. This is the default.

- **OFf**

  A required literal that disables shift registration identification and also resets all related configurations to their default values.

- ALLOW_COMBINATIONAL_logic_between_registers {ON | OFF}

  An optional switch and literal pair that specifies for the tool to consider or ignore the use of the DRC calculated state-stability values during shift register identification. When the switch is Off, the constraint values that sensitize a data path between sequential cells are ignored. As a result, multiple-input gates (AND, OR, MUX, etc.) between sequential cells cannot be traced during shift register identification. Single-input gates (BUF and INV), on the other hand, can always be traced. When the switch is On, the constraint values that sensitize a data path between sequential cells are considered during shift register identification tracing. Multiple-input gates do not block the tracing as long as a single path is sensitized through them.

  OFF

  A literal that specifies ignoring the constraint values. This is the default.

  ON

  A literal that specifies considering the constraint values.

Examples

The following example disables automatic shift register identification.

```
set_shift_register_identification off
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

report_scan_cells

report_shift_registers

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_silicon_insight_option

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Specifies optional settings for running Tessent SiliconInsight.

Usage

set_silicon_insight_option [-cdp_verification { on | off }]
[-ignore_uncontacted_pin_data { on | off }]
[-interactive_ijtag { on | off }]
[-interactive_ijtag_pattern_type { SVF | STIL}]
[-simdut_port *name*] [-simdut_timeout *integer*]
[-maximum_failing_cycles *max_failing_cycles*]
[-allow_compressed_patterns { on | off }]

Description

The set_silicon_insight_option command specifies optional global settings that determine the behavior of subsequent Tessent SiliconInsight sessions. These global settings remain in effect until you change them.

Some of the options apply to particular Tessent SiliconInsight modes of operation, such as SimDUT mode, and others are general.

Arguments

- -cdp_verification { on | off }

  An optional switch that specifies whether to automatically perform CDP verification. Set this option to “off” when you are not using a TSDB. For example, when you are running custom tests added using the add_cdp_test command.

  In addition, this switch ensures that new test patterns added as part of a test were generated by Tessent tools (and hence contain a signature that links it to the design it was generated for). You may want to turn verification off if you are adding test patterns that were not generated by Tessent tools.

- -ignore_uncontacted_pin_data { on | off }

  An optional switch that specifies whether to ignore uncontacted pin data on the selected_setup adaptor. You may want to set this option to “on” when, for example, you are running a custom test that uses more pins than are declared contacted in the pinmap and you know that these extra pins can safely be ignored.

- -interactive_ijtag { on | off }

  An optional switch that specifies whether to turn on interactive IJTAG so that you can debug a device at the PDL level. See section “PDL-Level Debugging for Tessent IJTAG-Inserted Devices” in the *Tessent SiliconInsight User’s Manual for Tessent Shell* for details.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -interactive_ijtag_pattern_type { SVF | STIL}

  An optional switch and string pair that specifies whether to generate SVF or STIL patterns for interactive IJTAG analysis and debugging. The default is SVF.

- -simdut_port *name*

  An optional switch and string pair that specifies the name of the SimDUT port.

- -simdut_timeout *integer*

  An optional switch and integer pair that controls the length of time before SimDUT mode times out. You can set this option to ensure that the tool does not time out before elaboration completes. Specify the time in seconds. The default is 600.

- -maximum_failing_cycles *max_failing_cycles*

  An optional switch and integer pair that enables you to control the maximum number to capture failing cycles in pattern execution. By default, the tool collects and processes all failures. In cases of potential catastrophic defects, pattern execution can take a long time and result in unusable data. This option enables you to collect and process subsets of the failures. The default is 2000000000.

- -allow_compressed_patterns { on | off }

  An optional switch that specifies whether to enable Tessent SiliconInsight to use compressed STIL or SVF patterns. The default is off. When you enable compressed patterns, the tool generates and uses compressed patterns automatically when they are available.

Examples

The following example turns on interactive IJTAG debugging.

```
set_silicon_insight_option -interactive_ijtag on
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_simulation_library_sources

Context: unspecified, all contexts

Modes: all modes

Defines the directories and files in which to search for the simulation Verilog library cells that are to be used by the run_testbench_simulations command.

Usage

set_simulation_library_sources {[-v *file_path* ...] [-y *dir_path* ... -extensions *ext* ...]}
[-sv_extensions *ext* ...]
[-logical_library_map_list *name path2dir* ...]}
[-include_directory_list *dir_path* ...]
[-f *file_path* ...]
| [-clear]

Description

Defines the directories and files in which to search for the simulation Verilog library cells that are to be used by the run_testbench_simulations command. The -logical_library_map_list is used to reference already compiled libraries.

When you use the -logical_library_map_list option, you must make sure to reference pre-compiled libraries that are consistent with the simulator and simulation version you are using with the run_testbench_simulations command. For example, a pre-compiled library compiled with the Questa 7.2 vlog command cannot be simulated with a different simulator nor with a newer version of vsim.

All switches for this command are interrelated. Each switch is set to their new value each time this command is issued. For example, specifying the command without the -v switch does not preserve the previously set -v values but instead clears them.

Arguments

- -v *file_path* ...

  An optional repeatable switch and value pair that specifies one or more files in which to search for Verilog simulation modules. The relative order of all -v and -y switches are preserved: the library modules are searched from the first one to the last. and the -v files are search from top to bottom.

- -y *dir_path* ...

  An optional repeatable switch and value pair that specifies one or more directories in which to search for files matching Verilog simulation module names. Files that have a leaf name that matches a module name and have one of the extensions specified with the -extension switch are automatically loaded if the module does not already exist in memory. You must specify the -extension option when you have one or more usages of the -y switch. The

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

relative order of all -v and -y switches is preserved, and the library modules are searched from the first one to the last.

Note

When you use the -y switch, all the library filenames should have the *cell_name.v* format. If you have a single file, the tool does not search in the file, so it does not read the file. If a single file has all the library cells, use the -v switch instead.

- -extensions *ext*

  An optional non-repeatable switch and value pair that specifies the list of extensions that are used to search the specified -y directories for files matching the module names. You must specify the -extension switch once and only once when you specify the -y switch one or more times.

- -sv_extensions *ext*

  An optional repeatable switch and value pair that specifies the list of SV extensions that are used to search the specified directories matching the module names.

- -logical_library_map_list *name path2dir*

  An optional non-repeatable switch and value pair that specifies a list of logical library name and directory name pairs. Each of those pairs associates a physical directory to a pre-compiled logical library name. You can map multiple logical libraries to their respective directories by specifying multiple pairs of elements. The pre-compiled libraries must be consistent with the simulator and simulator version that you are using in the run_testbench_simulations command. For example, a pre-compiled library compiled with the Questa 7.2 vlog command cannot be simulated with a different simulator nor with a newer version of vsim.

- -include_directory_list *dir_path*

  An optional switch and repeatable string that specifies one or more directory paths that contain `include files.

- -f *file_path* ...

  An optional switch and Tcl list that specifies the name or names of one or more ASCII files that contain pointers to design files. The referenced file is usually generated by another tool, so the -f option enables you to conveniently use the file rather than to reformat the contents of the file into a set_simulation_library_sources command line. The file can contain only the following elements, one per line:

  - *design_path* — A string that specifies the name of a Verilog file to load.
  - -v *design_path* — A string that specifies the name of a Verilog file to load.
  - -y *dir_path* — A string that specifies a path to a directory that contains Verilog files to load.
  - +libext+*file_extension* — A string that specifies which files in the “-y dir_path” to load. For example: “+libext+.v” loads all .v files.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- +define+*XXX[=YYY]* — A string that defines a macro called "XXX" with an optional value of "YYY". The defined macro is used when compiling any files listed in the -f file.
- +incdir+*dir_path1*+*dir_path2*+... — A string that defines one or more directory paths to directories that contain files that use the 'include directive.

If you use the -f switch with the set_simulation_library_sources command, the *file_path* can be relative; however, when specifying the -f switch with the run_testbench_simulations command, you must specify an absolute path for *file_path*. The *file_path* must be an absolute path to the file it references because the "compilation options" is used verbatim and not interpreted at all by the run_testbench_simulation command as follows:

```
run_testbench_simulations -compilation_options { -f file_path }
```

In all cases, the content of the file referenced by *file_path* must include only absolute paths to files/directories.

- -clear

  An optional switch that clears the previously specified values for all other switches in this command as if they had never been specified before. This switch is mutually exclusive with all other switches.

Examples

The following example defines the simulation library sources to be a directory called "./mems" which contains the memory simulation files with a .v extension and a directory named "./tsmc130_questa_10.2" in which the TMSC Verilog simulation cells were pre-compiled using vlog version 10.2. The assumption is that the run_testbench_simulations command uses Questa as its -simulator value and that the Unix search path environment is such that it picks the Questa commands associated with version 10.2.

```
> set_simulation_library_sources -y ./mems -extensions v \
    -logical_library_map_list tsmc130 ./tsmc130_questa_10.2
> run_testbench_simulation -simulator questa
```

Related Topics

report_simulation_library_sources

run_testbench_simulations

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_simulation_options

Context: all contexts

Mode: setup, analysis, insertion

Controls the behavior of the tool during pattern generation and simulation.

Usage

set_simulation_options [-SET_Reset_dominate_clock {OFf | ON}]
[-C6_mask_races {OFf | ON}]
[-C26_mask_races {OFf | ON | CRoss_domain_only}]
[-MULTIcycle_fault_simulation {ON | OFf}]
[-MUX_Select_x_sim_x {OFf | ON}]
[-FEEDback_buffer_x_init_value {ON | OFf}]
[-REPORT_X_capturing_cells {OFf | ON}]
[-REPORT_Race_masking {OFf | ON | VErbose }]
[-SIMULATE_LEARNED_SEQuential_tie_gates {OFf | ON}]
[-OBS_SCAN_PER_cycle_data_limit *integer*]
[-OBS_SCAN_LAST_shift_cycles *integer*]

Description

The set_simulation_options command specifies options to control the behavior of the tool during pattern generation and simulation.

Arguments

- -SET_Reset_dominate_clock {OFf | ON}

  An optional switch and literal pair that determines whether the assertion of the set or reset port dominates the clock value for both flip-flops and latches when multiple ports are on simultaneously. When the switch is on, and either the set or reset port is on (but not both), the evaluation value of the state element is dependent on the set or reset port. When the switch is off and multiple ports are on simultaneously, the evaluation value is dependent on all enabled ports. When enabled ports agree on the same value, the state element is set to the agreed value. Otherwise, the state element is set to X. Also, the state element is set to X when Set and Reset port are both on simultaneously despite this switch setting.

  OFf

  A literal that specifies disabling set/reset as the dominant port.

  ON

  A literal that specifies enabling set/reset as the dominant port. This is the default.

- -C6_mask_races {OFf | ON}

  An optional switch and literal pair that turns on pessimistic simulation on C6 violated state elements and masks the capture values only when the data port transitions at the same frame the clock port captures. This option enables you to change the handling of C6 violated cells when data and clock ports have a race transition. By default, create_patterns turns on

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

clock_off simulation and the tool assumes that the clock port's transition is earlier than the data port's transmission. If this assumption is not true, this option is recommended to mask the C6 violated cells to prevent mismatches due to races. This functionality is less pessimistic than the "add_cell_constraints -DRC C6" command, because the latter command masks C6 violated cells even when no races occur.

In cases where a C6 violation results from a race through combinational logic that reconverges, the tool automatically adds false paths when you enable this option. This masks out these races. When adding these paths, the tool issues the following message:

```
//  <N> false paths are added for C6 violation.
```

OFf

A literal that disables this functionality. This is the default.

ON

A literal that enables this functionality.

- -C26_mask_races {OFf | ON | CRoss_domain_only}

  An optional switch and literal pair that controls masking to prevent mismatches when a race condition occurs on sequential elements that are violating C26 DRC rules. The default value, "cross_domain_only", masks the capture value of state elements affected by C26 violations only for cross-domain violations. The "on" value masks the capture value of all state elements affected by C26 violations. When there are C26 violations on flops, the tool adds hold-time false paths on the data path of the cells with violations to mask the race conditions for values that enable masking. The "off" value means that state elements that violate C26 DRC rules still capture values according to regular state elements (for example, capturing clock_off values for a flop and clock_on values for a latch).

- -MULTIcycle_fault_simulation {ON | OFf}

  An optional switch and literal pair that determines whether the tool runs multicycle path ATPG and fault simulation. The default is ON. When this switch is ON, ATPG creates patterns to detect faults on the multicycle paths and specifies that the fault delay on the multicycle path is (n+1) cycles for fault simulation, where n is the cycle multiplier of the multicycle path. When a fault pin is included in multiple multicycle path cones, the fault delay is determined by the smallest cycle multiplier.

- -MUX_Select_x_sim_x {OFf | ON}

  An optional switch and literal pair that specifies to model _mux primitives in the design as either consensus or non-consensus. By default, the tool always models _mux as consensus (evaluates _mux with a known value as long as both data ports are the same known value).

  You can use the ON argument to model the _mux primitive as non-consensus (output X for every _mux instance whose select line is X). Note that changing _mux primitives to non-consensus can reduce the test coverage. Also, you can use this switch in any mode, but when you use it in setup mode, the tool evaluates _mux instances as non-consensus mux instances, which can result in additional DRC violations.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -FEEDback_buffer_x_init_value {ON | OFf}

  An optional switch and literal pair that enables the FB_BUF primitive to carry over its value from the previous cycle instead of being initialized to X. The default is ON, which means that the FB_BUF is still initialized with X value at the beginning of a simulation cycle. You can choose to turn off X initialization to get less pessimistic simulation results. Note that turning off this option increases the chance of pattern simulation mismatches if all FB_BUF primitives in the design cannot be safely simulated in this manner.

- -REPORT_X_capturing_cells {OFf | ON}

  An optional switch and literal pair that reports scan cells that capture X during simulation. When you set this option to ON, the tool displays this message when detecting a parallel pattern that captures X:

  ```
  // 10 scan cells capture X
  ```

  You use "report_patterns -unload_x" to find the patterns that capture X on scan cells. In addition, when you specify the -pattern_index switch with -unload_x, the report_patterns command reports the scan cells that capture X on the specified pattern.

- -REPORT_Race_masking {OFf | ON | VErbose}

  An optional switch and literal pair that enables the tool to report when races or glitches are detected that cause sequential element capture values to be masked. These are reasons for races or glitches that can cause the sequential elements to be masked:

  - C6 races — The sequential element is masked due to C6 port races, where the C6 data port value is changing in the same frame as the clock port capture frame.
  - Glitches on set/reset port in the scan cell loading frame — The set or reset port has potential glitches and the sequential value can be different depending on the glitch value.
  - Set or reset port race with the clock port — The set or reset port is de-asserted in the same frame as the clock port capture frame and the capture value can be different if the de-assert happens before or after the clock port capture event.

  This message is reported during the good machine simulation when this switch is set to "on":

  ```
  //  Warning: Forcing gate '/cm0507/cm2304/' (7956) to X due to
  //  glitch at set or reset port for pattern 7.
  ```

  Note

  To prevent excessive messages, only the first ten masked cell are reported for by this tool for race and glitch masking cases (except C6 races). If there are more than ten masked cells, this message is reported:

  ```
  // Only the first 10 have been reported.
  // Use 'report_gates -glitch_masked_cells' to report all masked
  // cells.
  ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -SIMULATE_LEARNED_SEQuential_tie_gates {OFf | ON}

  An optional switch and literal pair used to change the ATPG simulator behavior for sequential gates learned as tied during DRC. When you specify off, the switch default, the tool maintains such sequential tie gates at the learned tie values during simulation.

  When you specify on, the tool does not use the learned tie values and evaluate all such sequential gates during simulation.

  One possible use for this switch would be to resolve Verilog simulation differences due to X handling on a control port, where the Verilog simulator pessimistically evaluates the gate as X, even though a constant value can be preserved in silicon.

- -OBS_SCAN_PER_cycle_data_limit *integer*

  Used only for observation scan. An optional switch and integer pair that changes the default maximum number of patterns for which the tool stores per-cycle data (that the tool can monitor for debugging purposes). The specified integer is the maximum number of patterns that the tool can write to the PatDB. For more information, see "Simulation Options for Observation Scan" in the *Hybrid TK/LBIST Flow User's Manual*.

- -OBS_SCAN_LAST_shift_cycles *integer*

  Used only for observation scan. An optional switch and integer pair that enables you to extend the number of shift cycles simulated in the parallel testbench. (Normally, a parallel testbench contains only one shift cycle and one capture cycle.) For *integer*, you can specify up to 10% of the total shift length. For more information, see "Simulation Options for Observation Scan" in the *Hybrid TK/LBIST Flow User's Manual*.

Examples

Example 1

The following example demonstrates disabling set/reset as the dominant synchronous port:

set_simulation_options -set_reset_dominate_clock off

Example 2

The following example illustrates how to change the tool to model all _mux instances as non-consensus for both DRC checking and ATPG:

SETUP> set_simulation_options -mux_select_x_sim_x on

SETUP> set_system_mode analysis

```
//  Flattening process completed, cell instances=28234, gates=45400,
//  PIs=110, POs=103, CPU time=93.00 sec.
//  ---------------------------------------------------------------------
//  Begin circuit learning analyses.
//  ---------------------------------------------------------------------
//  Equivalent gates=54   classes=0   CPU time=9.00 sec.
...
```

ATPG> create_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

The following example illustrates masking as a result of a C6 race condition:

SETUP> set_simulation_options -report_race_masking on -report_x_capturing_cells on

SETUP> set_simulation_options -c6_mask_races on

SETUP> set_system_mode analysis

```
//  Flattening process completed, cell instances=29, gates=67, PIs=9,
//  POs=4, CPU time=0.02 sec.
//  ---------------------------------------------------------------------
//  Begin circuit learning analyses.
//  ---------------------------------------------------------------------
...
//  All scan clocks successfully passed off-state check.
//  Warning: There were 2 clock rule C6 fails (clock may capture data
//   affected by itself).
...
```

ATPG> create_patterns

```
…
//  Warning: Forcing gate '/ff3/' (61) to X due to glitch on data port with
//   C6 violation for pattern 2.
//  Warning: Forcing gate '/ff1/' (63) to X due to glitch on data port with
//   C6 violation for pattern 2.
//  Warning: Forcing gate '/ff5/' (59) to X due to glitch on data port with
//   C6 violation for pattern 2.
//  5 scan cells capture X in 5 out of 6 patterns
…
```

Related Topics

add_cell_constraints

get_simulation_option

report_patterns

set_clock_off_simulation

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_skewed_load

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the tool includes a skewed load in the patterns.

Usage

set_skewed_load **OFf** | **ON**

Description

The set_skewed_load command either enables patterns to include a skewed load or restricts patterns from including a skewed load. You can use this command in any system mode, but if you use it outside the Setup mode, you must provide a skew_load procedure in the test procedure file. If the skew_load procedure does not exist, the tool issues an error.

You use the skew_load procedure in LSSD designs to place different data in the master and slave of a scan cell by applying an additional clock pulse to the master shift clock.

Arguments

- **OFf**

  A literal that specifies to not include a skewed load in the patterns. This is the default behavior upon invocation of the tool.

- **ON**

  A literal that specifies to include a skewed load in the patterns.

Examples

The following example specifies for patterns to include the skewed load:

```
set_skewed_load on
set_system_mode analysis
create_patterns
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_split_capture_cycle

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Controls whether the tool updates simulation data between clock edges.

Usage

set_split_capture_cycle **ON** | **OFf**

Description

Tip

If you use ATPG Expert (via the create_patterns command), you do not need to issue this command. ATPG Expert automatically enables this command when it is needed. You may get the best results by letting ATPG Expert decide when to use this command.

The set_split_capture_cycle command enables or disables the simulation of level-sensitive and leading edge state elements updating as a result of applied clocks. When set to on, the tool updates simulation data between clock edges. This “split capture cycling” of data enables the tool to determine correct capture values for trailing edge and level-sensitive state elements, even in the presence of C3 and C4 violations. When set to “off”, simulation data is updated once each clock cycle.

Note that the set_split_capture_cycle command affects ATPG (subsequent create_patterns commands) but not logic simulation (subsequent simulate_patterns commands).

Arguments

- **ON**

  A literal that specifies to update ATPG data between clock edges. This is referred to as split-capture cycling.

- **OFf**

  A literal that specifies to not update ATPG data between clock edges. This is the default behavior upon invocation of the tool.

Examples

The following example enables split capture cycling:

```
set_split_capture_cycle on
```

Related Topics

set_clock_off_simulation

set_pattern_type

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_ssn_datapath_options

Context: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: setup, analysis

Use this command to specify options controlling SSN datapaths.

Usage

set_ssn_datapath_options [-datapath *datapath_id*]
[-alternate_input_id *alternate_input_id*] [-alternate_output_id *alternate_output_id*]
[[-input_bus_width {auto | *int*}] | [-output_bus_width {auto | *int*}]]

Description

This command enables you to specify options controlling SSN datapaths. You can specify the maximum bus width for each SSN datapath, and you can create input/output groupings for pattern generation.

An error occurs if the value of the SSN bus width is not divisible by the frequency_ratio of one of the SSN BusFrequencyMultiplier or BusFrequencyDivider nodes.

Arguments

- -datapath *datapath_id*

  An optional switch and string pair that identifies which SSN datapath to target. If unspecified, all SSN datapaths are targeted.

- -alternate_input_id *alternate_input_id*

  An optional switch and string pair that specifies an alternate input ID to identify the SSN bus input ports intended for use in the datapath <*datapath_id*>, instead of the default bus input ports. You must specify this argument with the -datapath switch. There must be ICL ports that have the tessent_ssn_datapath_input_ids attribute with <*alternate_input_id*> as one of the list values. The tool uses these ports for the specified datapath. The original datapath inputs are no longer considered part of a named SSN datapath.

  To undo this specification, use this command again with this switch set to an empty string or the original datapath name.

- -alternate_output_id *alternate_output_id*

  An optional switch and string pair that specifies an alternate output ID to identify the SSN bus output ports intended for use in the datapath <*datapath_id*>, instead of the default bus output ports. You must specify this argument with the -datapath switch. There must be ICL ports that have the tessent_ssn_datapath_output_ids attribute with <*alternate_input_id*> as one of the list values. The tool uses these ports for the specified datapath. The original datapath outputs are no longer considered part of a named SSN datapath.

  To undo this specification, use this command again with this switch set to an empty string or the original datapath name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -input_bus_width auto | *int*

  An optional switch to specify the width of the SSN datapath. This switch is mutually exclusive with the -output_bus_width switch. You can use this switch in cases where there is no output stream observation, such as with on-chip compare for a pass/fail test. Choose from the following:

  auto — The tool calculates the SSN datapath width. This is the default.

  int — An integer value; must be equal to or greater than 1 up to a maximum of the maximum input bus width.

- -output_bus_width auto | *int*

  An optional switch to specify the width of the SSN datapath. This switch is mutually exclusive with the -input_bus_width switch. Choose from the following:

  auto — The tool calculates the SSN datapath width. This is the default.

  int — An integer value; must be equal to or greater than 0.

  For a value of 0, the datapath is intended to be an input-only datapath. In this case, the tool checks that all nodes on the datapath have on-chip compare enabled, and it does not write scan chain outputs into the generated patterns.

Examples

Example 1

The following example sets the width of all SSN datapaths to 48:

set_ssn_datapath_options -output_bus_width 48

Example 2

The following example sets the SSN datapath width to 32 for ScanHost ID 1:

set_ssn_datapath_options -datapath 1 -output_bus_width 32

Example 3

The following example manually sets the input bus width of datapath 1 to 1, with an alternate datapath input ID to alt_dp_in1:

set_ssn_datapath_options -datapath 1 -input_bus_width 1 -alternate_input_id alt_dp_in1

To clear this bus width and alternate input ID setting, use the following command. This command generates a warning in the tool.

set_ssn_datapath_options -datapath 1 -alternate_input_id {}

```
//  Warning: Previous -input_bus_width specifications of the datapath '1'
//  are now deleted due to the use of -alternate_input_id.
```

Related Topics

get_ssn_datapath_option

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_ssn_options

Context: dft, patterns

Mode: setup, analysis

Enables or disables SSN on the current physical block or design.

Usage

set_ssn_options [{auto | off}]
[-streaming_interface {bus | ijtag}]
[-stream_optimization {test_time | data_volume}]
[-packet_constraints {off | no_rotation}]
[-throttling {auto | off}] [-ssh_mapping_annotations {on | off}]

Description

Enables or disables SSN for the current physical block or design.

Arguments

- auto | off

  An optional literal that enables or disables SSN. Choose from the following:

  auto — Enables SSN if the tool finds SSN in the design. This is the default.

  off — Disables SSN (enables SSN bypass).

  This argument is available only in setup mode.

- -streaming_interface {bus | ijtag}

  An optional switch that selects which ScanHost interface to use during streaming for either ATPG or pattern retargeting. The same patterns delivered through the parallel bus can also be rewritten to be applied through the ScanHost IJTAG interface. See the section "Streaming-Through-IJTAG Scan Data" in the *Tessent Shell User's Manual* for details on why you may want to enable Streaming-Through-IJTAG.

  bus — Selects the parallel bus interface as the access method to the ScanHost. The tool reports an error when you select "bus" and the ScanHost only supports Streaming-Through-IJTAG hardware. This is the default.

  ijtag — Selects the IJTAG interface as the streaming interface to the ScanHost. This selection only works if all active ScanHosts support IJTAG streaming.

  Note

  When a chip-level IEEE 1149.1 JTAG TAP Controller exists in your design, this is normally also the IJTAG interface.

  This argument is available only in setup mode.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -stream_optimization {test_time | data_volume}

  An optional switch and literal pair that specifies whether SSN should optimize for test time or for test data volume.

  test_time — By default, SSN optimizes for reduced test application time by finding the packet size that provides the best test time.

  data_volume — Causes the tool to find the smallest possible packet. This provides the smallest data volume but can increase test time.

  Note

  For sign-off simulation (*not* for production patterns), it may be useful to pad the packet to enable operating the speed at maximum frequency. Set the SIM_SSN_MAXIMUM_BUS_SPEED parameter to 1 to enable this packet padding. For more information about sign-off simulation, refer to the section "Signoff Patterns With SSN" in the *Tessent Shell User's Manual*.

  This argument is available only in setup mode.

- -packet_constraints {off | no_rotation}

  An optional switch and literal pair that controls rotation of packet position within the bus by padding packets to be a multiple of the bus width. Packets can continue to span multiple bus words or cycles. By default, rotation of packet position within the bus is enabled. Disabling rotation enables you to more precisely identify failing cores.

  Note

  Disabling rotation by padding packets generally increases the test data volume and test application time.

- -throttling {auto | off}

  An optional switch and literal pair that controls data throttling when creating packets. By default, SSN may throttle data to a core that has a smaller pattern count—that is, it sends less data per packet to the throttled core. This frees up bandwidth for cores with larger pattern counts, enabling all cores to finish at relatively similar times. However, throttling means that a given bit on the bus does not always correspond to the same core-level channel.

  Disabling this throttling behavior along with rotation (by also using the "-packet_constraints no_rotation" option) can provide greater granularity in identifying failing core channels instead of identifying only the failing SSH when you disable packet rotations without also disabling throttling.

  To include annotations in the STIL or WGL file so you can map to the failing channel instead of just the failing SSH, you must disable throttling and enable mapping annotations

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

directly with the -ssh_mapping_annotations argument. The following is an example annotation from the STIL/WGL files where throttling and packet rotation are both disabled:

```
Ann {* TESSENT_PRAGMA ssn_mapping -datapath_id 1 -cycle_repetition 2
-cycle_mod 0 -design_port "ssn_bus_data_out[3]" -ssh_icl_instance
xtea_3.xtea_rtl_tessent_ssn_scan_host_1_inst -core_instance
"xtea_3" -ssh_chain_group 3 -ssh_chain_group_chain_index 2 *}
```

These notations are identical to the ones generated with only "-packet_constraints no_rotation", except that two additional values at the end indicate the chain_group on the SSH and the index of the chain within that chain_group—that is, a pointer to the exact from_scan_out port of the SSH (driven by the EDT channel or uncompressed scan chain) that maps to the datapath output in those cycles.

- -ssh_mapping_annotations {on | off}

  An optional switch and literal pair that controls inclusion of annotations in STIL and WGL files that specify the mapping from the top-level ports to the SSH. By default, the pattern files contain these mapping notations.

  These annotations are based on shift cycles, excluding test_setup, ssn_setup, load_unload, and capture. To indicate where the SSN shift cycles begin, the files include the following annotation:

  ```
  Ann {* TESSENT_PRAGMA ssn_mapping -cycle_repetition_beginning *}
  ```

  For internal capture, this annotation occurs before the first SSN shift cycle. For external capture, this annotation occurs before each SSN shift sequence.

  When you write the pattern to a single file, the annotation is *not* at the top of the pattern file. Instead, it occurs later in the file, after the test_setup and ssn_setup procedures. To locate the annotations you need for mapping, search for the "cycle_repetition_beginning" annotation string.

  When you write the scan payload separately from the test_setup/ssn_setup procedures, the annotation appears at the top of the payload file and matches the cycle count for the entire pattern file.

  For more information about how you can write the SSN patterns in single or multiple files, refer to the documentation for the write_patterns command, and in particular the -test_setup, -ssn_setup, and -all_setup_only switches, along with the examples for writing SSN patterns.

  For a given cycle, you must have a mapping per top-level port per value of *cycle* % *cycle_repetition*, where *cycle_repetition* is the number of bus cycles per packet. For example, if the packet spans two tester cycles, you need a mapping from the top-level datapath outputs to the respective SSH for even cycles and another mapping for odd cycles. In general, you need as many mapping sets for each datapath output port as the length of the packet.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following is an example annotation where throttling is enabled:

```
Ann {* TESSENT_PRAGMA ssn_mapping -datapath_id 1 -cycle_repetition 2
-cycle_mod 0 -design_port "ssn_bus_data_out[3]" -ssh_icl_instance
xtea_3.xtea_rtl_tessent_ssn_scan_host_1_inst -core_instance
"xtea_3" *}
```

This example annotation contains the following values:

- **cycle_repetition** — The number of tester cycles each packet spans. The mapping repeats every *cycle_repetition* tester cycles.
- **cycle_mod** — The value of *cycle* modulo (%) *cycle_repetition*. You need a mapping set per each value of *cycle_mod*, where *cycle_mod* is in the range of 0 to *cycle_repetition* - 1.
- **design_port** — The top-level datapath output port name to which this mapping statement applies.
- **ssh_icl_instance** — The ICL instance name of the SSH whose value is observed on this design port when *cycle* % *cycle_repetition* is equal to *cycle_mod*.
- **core_instance** — The design instance name of the core being retargeted.

For more information about these annotations, refer to "Yield Statistics on ATE With SSN" in the *Tessent Shell User's Manual*.

Examples

This example demonstrates using the get_ssn_option and set_ssn_options commands by attempting to use set_ssn_options to turn off SSN in analysis mode and then using it correctly in setup mode.

ANALYSIS> get_ssn_option -is_active

```
1
```

ANALYSIS> set_ssn_options off

```
//  Error: SSN mode can only be changed in SETUP mode.
```

ANALYSIS> set_system_mode setup

SETUP> set_ssn_options off

SETUP> get_ssn_option

```
off
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_stability_check

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the tool checks the effect of applying the main shift procedure on non-scan cells.

Usage

For dft -scan or when EDT is Off

set_stability_check {**ON** | **OFf** | **All_shift**}
[-SIM_Static_atpg_constraints {OFf | ON}]
[-MAX_shift_cycles {*max_number* | DEfault | LOngest_chain_length}]
[-SIM_Capture_procedures
{ON | OFf | ON_Exclude_empty_capture_cycle_for_chain_test}]

When EDT is On

set_stability_check {**ON** | **OFf**}
[-SIM_Static_atpg_constraints {OFf | ON}]
[-MAX_shift_cycles {*max_number* | DEfault | LOngest_chain_length}]
[-SIM_Capture_procedures
{ON | OFf | ON_Exclude_empty_capture_cycle_for_chain_test}]

Description

To perform scan chain tracing, design rules checking (DRC) sometimes requires values to be present from non-scan state elements. For example, a control register in a test controller, such as a JTAG TAP, needs to be checked to see that the TAP register holds state during the shift procedure. By default, this checking is performed. This command controls the level of checking.

For detailed information about the use and reporting of state stability, refer to the "State Stability Issues" appendix in the *Tessent Scan and ATPG User's Manual*.

Arguments

- **ON**

  A literal that enables the tool to perform a fast check of the effect of applying the main shift procedure on non-scan cells. This is the default behavior upon invocation of the tool.

- **OFf**

  A literal that disables the tool from performing any checks on the effect of applying the main shift procedure on non-scan cells.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **All_shift**

  A literal that enables the tool to perform the most detailed level of checking. The main shift procedure is simulated for as many applications as the procedures call for. When you specify this option, it can significantly increase your run time.

  Note

  This option is primarily for logic BIST applications where the contents of shift registers need to be known at the end of scan loading, at the beginning of the capture cycle.

- -SIM_Static_atpg_constraints OFf | ON

  An optional switch and literal pair that enables or disables the simulation of static ATPG constraints during DRC's state stability analysis. Static ATPG constraints are those you specify using the -Static switch with the add_atpg_constraints command.

  When you enable simulation of static ATPG constraints, the tool assumes that the constraints are satisfied and simulates their values during DRC's state stability analysis of nonscan memory elements. This simulation is useful in some clock gating situations.

  The invocation default is for DRC not to perform this simulation. The argument choices are as follows:

  OFf — A literal that disables the simulation of static ATPG constraints during DRC. This is the invocation default.

  ON — A literal that enables the simulation of static ATPG constraints during DRC.

- -MAX_shift_cycles *max_number* | DEfault | LOngest_chain_length

  An optional switch and integer or literal pair that limits the number of shift cycles. You can only use this switch when stability check is set to ON (set_stability_check ON).

  - The *max_number* is an integer greater than or equal to 1 (one) that limits the maximum number of shift cycles the state stability analysis can simulate. When setting *max_number*, you should follow the guidance of the D12 DRC warning message for the number of shift cycles; the tool automatically considers existing independent shift cycles when calculating this recommended number.
  - Use the DEfault option with this switch to revert to the default behavior of state stability analysis without imposing a specific shift cycle limit using the following syntax:

    ```
    set_stability_check ON -MAX_shift_cycles default
    ```

  - The LOngest_chain_length option instructs the tool to use the full-chain length as the iteration limit instead of the default. Using this option can significantly increase runtime.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -SIM_Capture_procedures {ON | OFf | ON_Exclude_empty_capture_cycle_for_chain_test} **(setup mode only)**

  An optional switch and literal pair that enables or disables a more accurate stability analysis for the NCP (named capture procedure) flow. The two "ON" options are useful when all capture sequences are using predefined NCPs. These options are needed only if the events applied in the NCP matter to state stability and the tool's ability to pass DRC. When performing state stability in preparation for DRC, the tool usually simulates the clocks as *off_value*-X-*off_value*. However, if the exact capture sequence to be applied during capture needs to be simulated accurately so the state of a clock or shift controller is in the right state, then you can use this option for that purpose.

  By default, the more accurate stability analysis for NCPs is turned OFF. The argument choices are as follows:

  ON — A literal that enables a more accurate stability analysis for the NCP flow. During state stability analysis for capture windows, the tool simulates every NCP, plus a virtual NCP that includes an empty cycle that does nothing (in order to mimic a chain test).

  For each gate, the tool takes the common simulation values at the end of all NCPs (and includes the empty cycle NCP) as the end value of capture window. That is, the gate's final stability value at the end of capture is 1 or 0 if and only if its simulation value at the end of all NCPs is 1 or 0; otherwise, the final stability value is X. This is still pessimistic in the sense that only the common values of all NCPs are preserved, but it is more accurate because the simulation of the capture is exactly as specified by those NCPs.

  For the case of an NCP with multiple loads, the tool splits the NCP during simulation, so there is an NCP for each individual load.

  OFf — A literal that disables the more accurate stability analysis for the NCP flow. This is the default.

  ON_Exclude_empty_capture_cycle_for_chain_test — A literal that enables the more accurate stability analysis, but specifies that the tool not include the virtual NCP with an empty cycle in the stability analysis (as with the ON option). Also, the tool does not save chain test patterns if the stable values at the beginning and end of the capture window for all NCPs are not the same.

  Note that once the tool has performed DRC stability analysis based on the NCPs, you must only run subsequent ATPG operations using those NCPs.

  For more information about the effect of using this switch on gate reporting, refer to "Example 12 — Basic with Enhanced Stability Check for NCPs" in the *Tessent Scan and ATPG User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example enables fast stability checking of the effect of applying the main shift procedure on non-scan cells:

set_stability_check on

Example 2

The following example enables fast stability checking as well as the simulation of static ATPG constraints during DRC. Notice the use of the report_environment command to confirm the resultant changes in tool settings:

report_environment

```
...
split capture cycle =     OFF
stability check =         OFF
...
```

set_stability_check on -sim_static_atpg_constraints on
report_environment

```
...
split capture cycle =     ON
stability check =         ON
...
```

Related Topics

set_gate_report

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_static_dft_signal_values

Context: dft patterns

Mode: setup

Prerequisites: ICL is elaborated.

Sets the static DFT signal that exists in the design such that the signal can be used as part of a test_setup.

Usage

set_static_dft_signal_values
[-all | -current_design | -icl_instances *icl_instances* | -instances *instances*]
dft_signal_value_pairs

Description

All static DFT signals implemented with IJTAG can be set using this command. See the add_dft_signals command for information on adding the DFT signals. The DFT signals are grouped by each module in which you added the DFT signals and ran the process_dft_specification command. These modules have the tessent_design_level attribute equal to “chip”, “physical_block”, and “sub_block” in their extracted ICL module view.

When setting a static DFT signal within an instance, the other DFT signals without an explicit setting are assigned an implicit value from the following using the heuristic described below. An implicit value results in an iWrite command in the iCall when the implicit value is different than the reset value. Explicit value always results in an iWrite even if the value matches the reset value of the signal. Use the report_static_dft_signal_settings command to see the effect of the set command. A - in the value column means the value is kept to its reset value:

- When setting a DFT signal of usage “scan_mode”:
  - All other DFT signals within the same instance with usage scan_mode are set to 0 unless the signal had a previous explicit setting.
  - All DFT signals within the same instance with usage “logic_test_control” are set to their “value_in_pre_scan_drc” values unless the signal had a previous explicit setting. If the “value_in_pre_scan_drc” value is an X, it is converted to a 0.
  - If the scan_mode_type is “internal”, the following DFT signals within the same instance are set to 1 unless the signals had a previous explicit setting:
    - int_ltest_en
    - output_pad_disable
  - If the scan_mode_type is “external”, the following DFT signals within the same instance are set to 1 unless the signals had a previous explicit setting:
    - ext_ltest_en

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

        - bscan_clamp_enable
    - All DFT signals within the same instance with usage “global_dft_control” are set to their “default_value_in_all_test” values unless the signal had a previous explicit setting.
    - When using scan_mode signals (edt_mode, multi_mode, and so on), only one mode is active at a time.
- When setting a DFT signal of usage “logic_test_control”:
    - All DFT signals within the same instance with usage “logic_test_control” are set to their “value_in_pre_scan_drc” values unless the signal had a previous explicit setting. If the “value_in_pre_scan_drc” value is an X, it is converted to a 0.
    - All DFT signals within the same instance with usage “global_dft_control” are set to their “default_value_in_all_test” values unless the signal had a previous explicit setting.
- When setting a DFT signal of usage “global_dft_control”:
    - All DFT signals within the same instance with usage “global_dft_control” are set to their “default_value_in_all_test” values unless the signal had a previous explicit setting.

To remove a previous explicit setting, use the reset_static_dft_signal_values command.

Arguments

- -all

  An optional switch that applies the setting to the specified DFT signals found within all ICL instances having the tessent_design_level attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition. The DFT signals found within the top module are also set.

  The -all, -current_design, -icl_instances, and -instances switches are mutually exclusive. When unspecified, the -current_design switch is the default.

- -current_design

  An optional switch that applies the setting to the specified DFT signals found within top module. The top-level module must have the tessent_design_level attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition.

  The -all, -current_design, -icl_instances, and -instances switches are mutually exclusive. When unspecified, the -current_design switch is the default.

- -icl_instances *icl_instances*

  An optional switch that applies the setting to the specified DFT signals found within the specified ICL instances. The specified ICL instances must have the tessent_design_level attribute set to “chip”, “physical_block”, and “sub_block” within their ICL module definition.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The -all, -current_design, -icl_instances, and -instances switches are mutually exclusive. When unspecified, the -current_design switch is the default.

- -instances *instances*

  An optional switch that applies the setting to the specified DFT signals found within the specified design instances. The specified design instances must match the value of the "hierarchical_design_instance" attribute found on an ICL instance with the tessent_design_level attribute set to "chip", "physical_block", and "sub_block" within their ICL module definition.

  The -all, -current_design, -icl_instances, and -instances switches are mutually exclusive. When unspecified, the -current_design switch is the default.

- *dft_signal_value_pairs*

  A repeatable set of string pairs used to specify DFT signal name and value pairs. The string list must include an even amount of elements where the odd elements correspond to DFT signal names and the even values correspond to 0 or 1. The string can also be a single well-formatted Tcl list containing an even number of element where the odd elements correspond to DFT signal names and the even values corresponds to 0 or 1.

Examples

The following example shows the use of the set_static_dft_signal_values followed by report_static_dft_signal_settings and the get_static_dft_signal_icall inside an open_pattern_set.

Note how the DFT signals start with no explicit nor implicit setting. You can use the reset_static_dft_signal_values to return to this state after you have specified the settings. In the example, the "int_edt" DFT signal is set to 1. Consistent with the algorithm described above, the other DFT signals of usage "scan_mode" are left to stay in their reset state. The DFT signals of usage "logic_test_control" are set to their "value_in_pre_scan_drc" values. Because "int_edt" is a scan_mode of type "internal", the int_ltest_en is also asserted high. The DFT signals of usage "global_dft_control" are automatically set to the "default_value_in_all_test" values. Remember that implicit values matching the reset values are shown as -. No iWrite commands are issued for them as their reset states are already the needed values.

To reuse this scan mode in an unwrapped manner, you explicitly set int_ltest_en to 0 to prevent the signal from being implicitly set. Similarly, you can explicitly set DFT signals of usage "global_dft_control" to a value that is different than their "default_value_in_all_test" values.

As described in the get_static_dft_signal_icall command description section, the get_static_dft_signal_icall command is automatically invoked to configure the test_setup when in a context that supports set_test_setup_icall.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
> set_context patterns -ijtag
> read_icl ../sub_block_l3.icl
> set_current_design sub_block_l3
>
> report_static_dft_signal_settings

// ICL Module      : sub_block_l3

// ---------------  -------------------  ---------  ----------
// DFT Signal Name  Usage                Set value  Set source
// ---------------  -------------------  ---------  ----------
// all_test         global_dft_control       -      -
// ltest_en         logic_test_control       -      -
// int_ltest_en     logic_test_control       -      -
// ext_ltest_en     logic_test_control       -      -
// int_edt          scan_mode(internal)      -      -
// ext_multi        scan_mode(external)      -      -
//
>   set_static_dft_signal_values int_edt 1
>   report_static_dft_signal_settings

// ICL Module      : sub_block_l3

// ---------------  -------------------  ---------  ----------
// DFT Signal Name  Usage                Set value  Set source
// ---------------  -------------------  ---------  ----------
// all_test         global_dft_control       1      Inferred
// ltest_en         logic_test_control       1      Inferred
// int_ltest_en     logic_test_control       1      Inferred
// ext_ltest_en     logic_test_control       0      Inferred
// int_edt          scan_mode(internal)      1      Explicit
// ext_multi        scan_mode(external)      0      Inferred
//
> set_system_mode analysis
> open_pattern_set P1
> puts [get_static_dft_signal_icall]
dft_signal_iproc
  sub_block_l3_rtl2_tessent_tdr_sri_ctrl_inst.all_test 1
  sub_block_l3_rtl2_tessent_tdr_sri_ctrl_inst.int_edt 1
  sub_block_l3_rtl2_tessent_tdr_sri_ctrl_inst.ext_muti 0
  sub_block_l3_rtl2_tessent_tdr_sri_ctrl_inst.int_ltest_en 1
  sub_block_l3_rtl2_tessent_tdr_sri_ctrl_inst.ext_ltest_en 0
  sub_block_l3_rtl2_tessent_tdr_sri_ctrl_inst.ltest_en 1
>   iCall {*}[get_static_dft_signal_icall]
> close_pattern_set
> write_patterns p1.v -verbose
```

Related Topics

add_dft_signals

get_static_dft_signal_icall

report_static_dft_signal_settings

reset_static_dft_signal_values

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_static_learning

Context: unspecified, all contexts

Mode: setup

Specifies whether the tool performs the learning analysis to make the ATPG process more efficient.

Usage

set_static_learning [ON | OFf] [-Limit *integer*] [-Verbose {ON | OFf}]

Description

The set_static_learning command controls whether the tool performs the learning analysis immediately after design flattening. The tools use the learned behavior for intelligent decision making in later processes, such as ATPG and DRC.

When static learning analysis is on, the additional learned behavior focuses primarily on bus gates. Static learning analysis also enables the test pattern generation process to immediately recognize conflicts and restricted decisions on ATPG constraints that result from the gate assignments. By enabling static learning, you prevent the tools from unnecessarily remaking many decisions, thereby improving the ATPG performance.

For more information about the learning analysis, refer to "Learning Analysis" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- ON | OFf

  An optional literal that enables or disables additional static learning analysis. The default enables the tool to perform additional learning analysis, but you may want to disable to save time if you are not running ATPG.

- -Limit *integer*

  An optional switch and integer pair that specifies a single gate simulation activity threshold. When the tool reaches that threshold, it discontinues learning on gates in that design region. You would use the -Limit switch for performance reasons. The default value for the *integer* option is 1000.

  Caution

  While changing the learning limit increases performance for some designs, it significantly decreases performance for a vast majority of designs. It is very unusual to need to change the learning limit and is therefore not recommended.

- -Verbose {ON | OFf}

  An optional switch and literal pair that enables additional detailed reporting of analysis results, including the numbers of equivalent gates, classes, implications, and tied gates.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example first enables access to the learned information and then enables the tool to perform additional learning analysis before design flattening:

```
set_learn_report on
set_static_learning on -limit 500
set_system_mode analysis
```

Related Topics

set_learn_report

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_system_mode

Context: all contexts

Mode: all modes

Specifies the operational state you want the tool to enter.

Usage

set_system_mode {**setup** | **insertion** | **analysis**} [-force]

Description

The default system mode after tool invocation is setup. The context the tool is in determines which system mode you can specify with the set_system_mode command. See the *Tessent Shell User's Manual* for more information about contexts and system modes.

If you attempt to change back to setup mode and have generated patterns, the tool prompts you to save the patterns with an error message similar to the following:

```
// Error: Current test pattern set has not been saved. Save the
// patterns or use the -force switch.
```

Arguments

- **setup**

  A required literal that specifies to enter Tessent Shell setup mode. This is the default.

- **insertion**

  A required literal that specifies to enter Tessent Shell insertion mode.

- **analysis**

  A required literal that specifies to enter Tessent Shell analysis mode.

- -Force

  An optional switch that suppresses the errors that are normally generated when exiting analysis or insertion system modes with unsaved data. Instead of issuing an error, the tool silently deletes unsaved data.

  You can also use the -force switch in the following situations:

  - When you are in the "patterns -ijtag" context and try to move out of analysis system mode without saving your pattern sets, you cannot change the system mode until you either save the pattern sets or use the -force switch to discard all unsaved pattern sets.
  - When you are in insertion system mode and in the rtl context, if you try to change to setup system mode without first saving edited or created modules, you cannot change the system mode until you either save the modules or use the -force switch to discard the unsaved modules.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example changes the system mode so you can perform an ATPG run:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 indata2 outdata4
set_system_mode analysis
create_patterns
```

Example 2

The following example changes the system mode so that you can perform scan stitching:

```
add_tied_signals 1 vcc
add_tied_signals 0 vss
add_clocks 0 clock
set_system_mode analysis
report_scan_elements
analyze_scan_chains
insert_test_logic
```

Example 3

The following example changes the system mode so that you can perform an insertion run:

```
set_context dft -no_rtl
read_verilog mydesign.v
read_cell_library tessent_cell.tcellib
set_current_design
set_system_mode insertion
create_instance u1/myInst -of_module myModule
```

Related Topics

get_system_mode

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_tcl_shell_options

Context: all contexts

Mode: setup

Specifies options for the Tcl shell.

Usage

set_tcl_shell_options [-abort_dofile_on_error {off | on | exit}]
[-legacy_dofile_comments {off | on}]
[-legacy_commands_with_spaces {off | on}]
[-minimum_command_typing {off | on}]
[-change_no_result_warnings_to_errors {off | on}]
[-regex_style {extended | perl}]

Description

You can use this command to control features that provide backward compatibility to old dofiles.

Arguments

- -abort_dofile_on_error {off | on | exit}

  An optional argument that controls whether the tool aborts or continues dofile execution if it detects an error condition.

  Caution

  Use caution when specifying “-abort_dofile_on_error off” because it traps all potential error conditions. Preferably, you should trap error conditions of specific commands by using the Tcl built-in catch command or the Tessent catch_output command.

- -legacy_dofile_comments {off | on}

  An optional argument that controls the treatment of "//" as a comment prefix. If treating "//" as a comment prefix causes issues use "set_tcl_shell_options -legacy_dofile_comments off" to disable it. For new dofiles, you should always use the "#" as a comment prefix.

- -legacy_commands_with_spaces {off | on}

  An optional argument that controls whether the tool accepts legacy, space-separated commands or not.

- -minimum_command_typing {off | on}

  An optional argument that controls whether the tool accepts abbreviated commands or not. Using abbreviated commands in your dofile carries the risk that future tool versions do not accept the abbreviations because of ambiguities with new commands.

  Specify off for this option to ensure that you always type the full name of a command. The scripts can also run faster when commands are typed in full.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -change_no_result_warnings_to_errors {off | on}

  An optional argument that controls whether the tool flags a no result returned from an introspection command as an error or a warning.

  The default setting is on, which means that the tool flags as an error a no result returned from an introspection command, unless you specify the -silent switch.

  Setting this switch to off instructs the tool to flag with a warning a returned no result list or collection from an introspection command. However, this option introduces a risk if your scripts cannot deal with empty collections correctly. As this may result in incorrect/unintended behavior, it is not recommended.

  It is recommended that you ensure your scripts can properly handle an empty collection when it is possible that a get_* introspection command finds nothing. For example, if you use the following, then it must be guaranteed that "get_instances XXX" command finds an instance:

  ```
  get_modules -of_instance [get_instances XXX]
  ```

  If it is possible for the get_instances command to return an empty collection, use the -silent switch and determine the size of the returned collection before passing it to other procedures or Tessent Shell commands:

  ```
  set inst [get_instances XXX -silent]
  if {[sizeof_collection $inst] > 0} {
     my_procedure $inst
     set mods [get_modules -of_instances $inst]
     ...
  }
  ```

  The additional check eliminates any risk of unexpected behavior if the procedure or the command processing the collection cannot correctly handle an empty collection.

  Do not use the -silent switch if you expect a command to always return a non-empty collection, and your script does not check the size of the returned collection as shown above.

  Using the -silent switch in such cases can cause a valid error message to go unnoticed and result in undesired behavior.

- -regex_style {extended | perl}

  An optional switch and literal pair to specify which type of regular expression Tessent Shell uses.

  extended — Tessent Shell uses POSIX extended regular expression matching.

  perl — Tessent Shell uses Perl-style regular expression matching.

Examples

The following example uses Perl-style regular expression syntax to look for all instances that do not start with the prefixes inst_top_0, inst_top_1, inst_top_5, and inst_top_6.

```
set_tcl_shell_options -regex_style perl
get_instances "(?!inst_top_(0|1|5|6)).*" -regexp
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
inst_top_2 inst_top_3 inst_top_4
```

Related Topics

get_tcl_shell_option

report_tcl_shell_options

catch_output

dofile

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_test_end_icall

Context: dft, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: setup

Adds an iCall to the test_end procedure.

Usage

set_test_end_icall {[icl_instance_path.]**iproc_name** [arguments …]} [[-merge] -append] [-non_retargetable] [-timeplate *icall_timeplate_name*]

Description

The set_test_end_icall command adds an iCall to the test_end procedure. All iCalls added by means of this command are executed before the iCall/iMerge commands that have been manually added to the test_end procedure by creating or editing the procedure file.

The command accepts a Tcl list consisting of the three components that normally follow an iCall: the name of the iProc; preceded by the optional ICL instance path and separated by a period; and one or more optional arguments to the iProc.

If you specify the -append switch, the new iCall is appended to the iCalls in memory. Each iCall with its arguments is called in the order the iCall was appended.

If you issue the command without the -append switch, the tool overwrites the previous values. To clear the list, assign an empty string or empty list without the -append switch.

The -merge switch provides support for iMerge in the test procedure file. It can only be used if the -append switch is also specified.

Arguments

- {[icl_instance_path.]**iproc_name** [arguments …]}

  The name of the iProc, preceded by the optional ICL instance path and separated by a dot, and one or more optional arguments to the iProc.

- -append

  An optional switch that appends the specified list of iCalls to the accumulated list of iCall commands. Note that if you issue the command without the -append switch, the accumulated iCalls are cleared and only the current specified iCall command is stored.

- -merge

  An optional switch to merge iCall commands together. The -merge option can only be used with the -append switch to merge the specified iCall with the previous iCall. All the merged iCall commands infer the -error_on_conflict option of the iMerge command and cannot be unspecified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Refer to "iMerge Conflict Reporting" in the *Tessent IJTAG User's Manual* for more information on the -error_on_conflict option.

The merged iCall commands can be specified in the test_setup/test_end procedures using the Tessent Shell commands set_test_setup_icall/set_test_end_icall. Later, when you use the report_procedures command, the merged iCall commands of the test_setup/test_end procedures appear as follows:

```
iMerge
   iCall iProcName iProcArgs
   ....
iMerge -end;
```

The -error_on_conflict option is always enabled, so it is not necessary to specify it explicitly.

- -non_retargetable

  An optional switch that specifies that iCalls that are not retargeted at the next level can be added. Use this switch to apply the iCall at the current (core) level and so it is not forwarded when the core is embedded in a design.

- -timeplate *icall_timeplate_name*

  An optional switch and string pair that specifies an iCall timeplate name other than the default.

Examples

In the following example, the iCall commands specified using the -merge option in the test_end file merge iProc2, iProc3 and iProc4 into iProc1:

```
set_test_end_icall iProc1 -append
set_test_end_icall iProc2 -append
set_test_end_icall iProc3 -merge -append
set_test_end_icall iProc4 -merge -append
```

These commands are equivalent to the sequence of the iCall commands in the test procedure file:

```
Procedure test_end =
        iCall iProc1;
        iMerge =
            iCall iProc2;
            iCall iProc3;
            iCall iProc4;
        end;
end;
```

Related Topics

get_test_end_icall_list

get_test_setup_icall_list

iCall

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_test_setup_icall

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_test_logic

Context: dft -scan, dft -test_points

Mode: setup

Inserts test logic to control the set, reset, clock, enable, or write control signals to make them scannable when scan chains are inserted.

Usage

set_test_logic {**-Set** {**ON** | **OFf**} | **-REset** {**ON** | **OFf**} | **-Clock** {**ON** | **OFf**} | **-RAm** {**ON** | **OFf**} | **-C6** {**ON** | **OFf**}}…

Description

Before inserting scan chains, you must specify the cell models with the add_cell_models command.

By default, no test logic is inserted on control signals. You can use the report_environment command to display the current test logic settings.

If all enable signals of a bus are driven purely by combinational logic, no additional test logic is needed to prevent contention due to scan shifting.

Arguments

- **-Set ON | OFf**

  A switch and literal pair that enables the insertion of test logic to make set signals controllable. The default setting is off.

- **-REset ON | OFf**

  A switch and literal pair that enables the insertion of test logic to make reset signals controllable. The default setting is off.

- **-Clock ON | OFf**

  A switch and literal pair that enables the insertion of test logic to make clock signals controllable. The default setting is off.

- **-RAm ON | OFf**

  A switch and literal pair that enables the insertion of test logic to make read and write signals controllable. The default setting is off.

  This option does not affect the set and reset signals of the RAM. You should use the edge_trigger attribute in the library definition of the RAM to specify inserting test logic to control these signals. For more information, see the "Attributes of RAM/ROM Primitives" section of the *Tessent Cell Library Manual*.

- **-C6 {ON | OFf}**

  A switch and literal pair that, when enabled, instructs the tool to add hardware to fix identified C6 violations. The tool fixes a C6 violation by inserting a multiplexer into the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

data path that feeds the flip-flop generating the violation. The test mode input of the multiplexer is driven by a nearby scan flip-flop. When a top-level clock feeds the data input of many flip-flops (which results in multiple C6 violations), the multiplexer is inserted at the highest possible level of hierarchy. The driving scan cells are selected to be in the same clock domain as the flip-flops generating the C6 violations. This setting is off by default.

Note

This option cannot be used when the "set_drc_handling -Conservative" switch is enabled.

Examples

The following example checks the set and clock signals of uncontrollable memory elements and makes them controllable with the addition of test logic:

```
add_clocks 0 clk
set_test_logic -set on -clock on
set_system_mode analysis
add_cell_models and2 -type and
add_cell_models or2 -type or
add_cell_models mux21h -type mux s a b
add_cell_models nor2 -type nor
report_cell_models
insert_test_logic
```

Related Topics

add_cell_models

delete_cell_models

report_cell_models

set_latch_handling

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_test_point_analysis_options

Context: dft -test_points

Mode: setup, analysis

Sets the maximum number of test points, the breakdown in control and observe points, the target fault coverage and the number of pseudo random patterns to be applied and some other parameters that are taken into account during test point analysis.

Usage

For VersaPoint test points when test point type is set to lbist_test_coverage:

```
set_test_point_analysis_options
    [-total_number {integer | float%}]
    [-control_points_number {integer | float%} -observe_points_number  {integer | float%}]
    [-test_coverage_target percentage]
    [-pattern_count_target integer]
    [-max_control_points_per_path integer]
    [-exclude_cross_domain_paths {on | off}]
    [-observe_primary_outputs {on | off}]
    [-observe_blocked_faults {on | off}]
    [-exclude_clock_domains {off | object_spec}]
    [-allow_in_xbounding_regions {on | off}]
    [-assume_clock_controllability {ideal | default}]
    [-shared_control_points_per_flop integer]
    [-shared_observe_points_per_flop integer]
    [-minimum_shift_length integer]
    [-capture_per_cycle_observe_points {on | off}]
    [-cascaded_icg_synchronization {on | off}]
```

For VersaPoint test points when test point type is set to edt_pattern_count:

```
set_test_point_analysis_options
    [-total_number {integer | float%}]
    [-control_points_number {integer | float%} -observe_points_number {integer | float%}]
    [-max_control_points_per_path integer]
    [-exclude_cross_domain_paths {on | off}]
    [-exclude_clock_domains {off | object_spec}]
    [-observe_blocked_faults {on | off}]
    [-allow_in_xbounding_regions {on | off}]
    [-assume_clock_controllability {ideal | default}]
    [-shared_control_points_per_flop integer]
    [-shared_observe_points_per_flop integer]
```

For VersaPoint test points when test point type is set to {edt_pattern_count lbist_test_coverage}:

```
set_test_point_analysis_options
    [-total_number {integer | float%}] |
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

[-control_points_number {*integer* | *float%*} -observe_points_number {*integer* | *float%*}]
[-test_coverage_target *percentage*]
[-pattern_count_target *integer*]
[-max_control_points_per_path *integer*]
[-exclude_cross_domain_paths {on | off}]
[-observe_primary_outputs {on | off}]
[-observe_blocked_faults {on | off}]
[-exclude_clock_domains {off | *object_spec*}]
[-allow_in_xbounding_regions {on | off}]
[-assume_clock_controllability {ideal | default}]
[-shared_control_points_per_flop *integer*]
[-shared_observe_points_per_flop *integer*]
[-minimum_shift_length *integer*]

Description

Not all gates in the design are considered by the tool for inserting test points. Test points are not inserted on gates in the clock tree or in the scan path, gates in protected blocks, or gates where you have explicitly specified that no test points should be inserted. If there are many locations where test points cannot be inserted, the tool may not be able to find the best locations for test points. This may increase the number of test points that are needed to reach the fault coverage target. Therefore, if test points cannot be inserted at more than 20 percent of the gate-pins the tool generates a warning. The warning specifies the number of these locations and the main reasons why test points cannot be inserted at these locations.

By default, if you do not specify the maximum number of test points using the -total_number, -control_points_number or -observe_points_number switches, the tool assumes the total number of test points to be 1 percent of the identified memory elements in the design for EDT test points. When the test point type lbist_test_coverage is used, the tool assumes the number of test_points to be 10000000.

Refer to "Test Points for Improving the Test Coverage of Deterministic Patterns" in the *Tessent Scan and ATPG User's Manual* for generating test points that improve the testability of a subset of faults.

Arguments

- -total_number {*integer* | *float%*}

  An optional switch and value pair that specifies the maximum number of test points to be inserted in the design, with no maximum as default. The value can be an integer or a percentage of the total number of flops. This maximum only applies to the test points automatically generated by the tool using the analyze_test_points command.

  Note

  The percentage must be specified as a floating number followed by the percentage sign (%).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -control_points_number {*integer* | *float%*}

  An optional switch and value pair that specifies the maximum number of control points to be inserted in the design, with no maximum as default. The value can be an integer or a percentage of the total number of flops.

  This maximum only applies to the control points automatically generated by the tool using the analyze_test_points command. The number of control and observe test points generated by the tool when you use the -total_number option is closer to optimal than specifying the maximum number of each through the use of the -control_points_number and -observe_points_number options.

  Note

  The percentage must be specified as a floating number followed by the percentage sign (%).

- -observe_points_number {*integer* | *float%*}

  An optional switch and value pair that specifies the maximum number of observe points to be inserted in the design, with no maximum as default. The value can be an integer or a percentage of the total number of flops, specified as a floating number followed by the percentage sign (%).

  This maximum only applies to the observe points automatically generated by the tool using the analyze_test_points command. The number of control and observe test points generated by the tool when you use the -total_number option is closer to optimal than specifying the maximum number of each through the use of the -control_points_number and -observe_points_number options.

  Note

  The number of control points and the number of observe points must both be specified in the same manner, that is, as an integer or percentage.

- -test_coverage_target *percentage*

  An optional switch and integer that specifies the target fault coverage to be achieved by the number of random patterns. The tool inserts test points until either the maximum number of test points is reached or the desired fault coverage with the target number of random patterns is achieved. The default is 99 percent.

  When you perform test point analysis for a set of target faults (instead of all faults), the default value is 100 percent.

  Note

  This option is not relevant when test point type is set to edt_pattern_count.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -pattern_count_target *integer*

  An optional switch and integer that specifies the number of random patterns to be applied to the design. Fault coverage estimations are done using this parameter as reference. The default is 100,000.

  When you perform test point analysis for a set of target faults (instead of all faults), the default value is 1000.

  Note

  This option is not relevant when test point type is set to edt_pattern_count.

- -max_control_points_per_path *integer*

  An optional switch and integer that specifies the maximum number of control points to be inserted on a single path. The default maximum number of control points per path is five.

- -exclude_cross_domain_paths {on | off}

  An optional switch and literal that prevents the tool from inserting test points in inter-clock domains. The default is off.

- -observe_primary_outputs {on | off}

  An optional switch and literal pair that controls the insertion of observe points at primary outputs. The default is off—no observe points are inserted at primary outputs.

  Note

  This option is not relevant when test point type is set to edt_pattern_count.

  If you set this switch to on, an observe point is inserted at each primary output where at least three previously unobserved cells can be observed.

  If you have specified a target set of faults, the tool by default inserts an observe point at each primary output where at least ten user-specified faults are observed by the inserted observe point that would otherwise not be observable.

  When you are in analysis mode and have selected test points with the analyze_test_points command, if you try to change the setting of the -observe_primary_outputs switch from on to off, you get an error message telling you that you need to first delete all test points before you can change the -observe_primary_outputs switch. If you try to change the setting of the -observe_primary_outputs switch from off to on, however, and test points have already been selected, you get a warning that some of the test points may no longer be optimal after adding test points to primary outputs, and for best results you should delete all test points before proceeding with the analyze_test_points command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The number of observe points inserted on primary outputs is independent of the total number of test points or the number of observe points that you may specify with the -total_number or -observe_point options of the set_test_point_analysis_options command. The total number of test points that is reported by report_test_points includes test points inserted at the primary outputs because the -observe_primary_outputs option was used, as well as those identified using the standard algorithm. Therefore, the total number may exceed the user-specified threshold.

- -observe_blocked_faults {on | off}

  An optional switch and literal pair that specifies that the tool insert observe points at locations where it helps in detecting faults because of a blocked path. The default is off.

  If you change this switch from on to off after selecting test points, the tool issues an error message that you must first delete all test points before changing the switch.

- -exclude_clock_domains {off | *object_spec*}

  An optional switch and literal that prevents the tool from inserting test points at any gate within a specified clock domain. The default is off.

  The value of *object_spec* can be a Tcl list of one or more object names or a collection of one or more objects. Typically, these objects are primary input pins that have been specified as clocks by the add_clocks command. The default test_clock or the test clock specified with the set_scan_signals command is also an acceptable clock object.

  When you use the "set_test_point_analysis_options -exclude_clock_domains" command multiple times, the latest specification of the -exclude_clock_domains switch overrides all previous specifications. Using the "-exclude_clock_domains off" switch deletes all excluded clock domains.

  By using this switch, you can prevent test points from the following locations:

  - Any gate that receives data that is launched from a flip-flop that is clocked by an excluded clock.
  - Any gate whose output value is captured at the data port of a flip-flop that is clocked by an excluded clock.

  If you want to allow test points at some gates in the excluded clock domains, set the no_control_point and no_observe_point attributes to false or use the delete_notest_point command to overwrite the -exclude_clock_domain switch. The set_attribute_value and delete_notest_point commands have a higher priority than the -exclude_clock_domains switch. You can enter these specifications (delete_notest_points command, no_control_point and no_observe_point attribute values, -exclude_clock_domains switch) in any order in setup mode or analysis mode. The locations where no test points are allowed are consistent, irrespective of the order of the commands and the time (setup or analysis mode) when they are entered.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In both setup and analysis modes, the set_test_point_analysis_options command with no options reports the excluded clock domains along with the settings of the other switches. In analysis mode, the no_control_point and no_observe_point attributes of the gates in the excluded clock domains are set to true. Therefore, the tool can report or introspect the actual gates that are in these excluded clock domains in analysis mode. For example, in analysis mode, you can use the report_notest_points command to show where test points are not allowed, including the locations in the excluded clock domains.

In analysis mode, if you have selected test points with the analyze_test_points command and then want to change the excluded clock domains, the tool reports an error and advises you that all test points must be deleted before the excluded clock domains can be changed.

When you specify an excluded clock domain, the tool first checks whether the specified pin is indeed a clock. That is, it verifies that the excluded clock domain has previously been specified as a clock with the add_clocks command (it also accepts the default test_clk or the test clock specified with the set_scan_signals command). It reports an error if the pin is not a clock. In analysis mode (or during the transition to analysis mode), the tool also verifies whether the excluded clock is actually clocking any flip-flops. It reports a warning if an excluded clock pin connects to the set/reset inputs of one or more flops. The warning mentions the clock pins that connect to a set/reset pin, but does not list the flops with a set/reset pin that is connected to an excluded clock. It ignores the connections to set/reset inputs. It only uses flip-flops that are actually clocked by an excluded clock pin to identify gates in the excluded clock domains.

Note

When you exclude a top clock domain, the tool does not automatically exclude clock domains derived from it. This gives you the flexibility to exclude the top clock domain without excluding all derived clock domains. However, the tool reports a warning when it excludes a top clock domain but not one or more of the derived clock domains.

- -allow_in_xbounding_regions {on | off}

  An optional switch and literal pair that specifies whether or not to allow the insertion of test points in the X-bounding regions identified during X-bounding analysis.

  on — Set this switch to on to enable the tool to insert test points in the X-bounding regions. The default is on if you use the edt_pattern_count test point type.

  off — The default value is off when the test point type is set to only lbist_test_coverage, or if the test point type is set to both lbist_test_coverage and edt_pattern_count.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

During X-bounding, the tool adds muxes or other logic to block X-sources at specific nets within the design. This blocking logic is placed as close to the X-source as possible, but may not be directly at the source. X-bounding regions are defined as any nets or pins that are in the path of the X-source prior to reaching the X-bounding hardware. These nets and pins may attain an X state even after the X-bounding hardware has been added.

- -assume_clock_controllability {ideal | default}

  An optional switch and literal pair which defines the behavior of test point analysis when calculating the controllability of the clock to scan elements. By default, the tool automatically calculates the controllability of the clocks to the scan elements and uses that information during analysis when determining where to insert test points. The options are:

  ideal — The tool assumes perfect controllability for each scan element's clock. This would likely be the case when additional circuit edits are performed after scan insertion and before ATPG, forcing the clocks to be completely controllable.

  default — The analysis takes into account the calculated controllability value when computed using the COP testability measure. This is the default.

- -shared_control_points_per_flop *integer*

  An optional switch and integer that specifies the maximum number of control points that can be driven by a single flop. The default number is 1.

  For more information, refer to "Test Point Sharing" in the *Tessent Scan and ATPG User's Manual*.

  Note

  Test point sharing for edt_pattern_count test points is supported. Sharing test point flops reduces the overall cell area impact of adding them to the design. This may impact your results in the following ways:

  - Coverage — While outliers may occur, ATPG test coverage impact is expected to be negligible.
  - Patterns — While outliers may occur, ATPG pattern count reduction may be impacted by 3-5%.

- -shared_observe_points_per_flop *integer*

  An optional switch and integer that specifies maximum number of observe points that can be driven by a single flop. The default number is 1.

  For more information, see "Test Point Sharing" in the *Tessent Scan and ATPG User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -minimum_shift_length *integer*

  Required for observation scan (DFT specification flow only). An optional switch and integer that specifies the minimum observation scan shift length for fault simulation. For *integer*, specify a length roughly equal to the length of the scan chains in BIST mode.

- -capture_per_cycle_observe_points {on | off}

  An optional switch and literal that activates ("on") or deactivates ("off") observation scan mode. For accurate test coverage estimation, you must also set the anticipated scan chain length of the design with the "set_test_point_analysis_options -minimum_shift_length" command. By default, the observation scan mode is deactivated.

  Refer to the Hybrid TK/LBIST Flow User's Manual for more information.

- -cascaded_icg_synchronization {on | off}

  An optional switch and literal that specifies to insert test points for improving clock controllability of cascaded clock gaters. By default, the tool does not insert any test points specifically for improving clock controllability.

Examples

Example 1

The following example specifies that enough test points should be generated to achieve 99.5 percent fault coverage, but not more than a maximum number of 3000 test points, for a design that to be tested with a set of 131072 random patterns:

```
set_test_point_analysis_options  -test_coverage_target 99.5 -total 3000 \
-pattern_count_target 131072
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following is a snippet from the log file that shows the test coverage reporting before and after selecting test points:

```
//  command: analyze_test_points
//  Identifying locations of X-bounding muxes prior to test point analysis
//
//  Test Coverage Report before Test Point Analysis
//  ----------------------------------------------
//  Target number of random patterns           131072
//
//  Total Number of Faults                      15577
//    Testable Faults                           15063  (  96.70%)
//      Logic Bist Testable                     14970  (  96.10%)
//      Blocked by xbounding                        0  (   0.00%)
//      Uncontrollable/Unobservable                93  (   0.60%)
//
//  Estimated Maximum Test Coverage                       99.38%
//  Estimated Test Coverage (pre test points)             96.19%
//  Estimated Relevant Test Coverage (pre test points)    96.24%
//
//  Warning: Test points cannot be inserted at 4033 (33.6%) gate-pins.
// This may increase the number of test points that is needed to reach the
// fault coverage target.
// 3929 (32.7%) gate-pins are located on clock lines or on the scan path.
//
//  Incremental Relevant Test Coverage Report
//  -----------------------------------------
//  TPs   50 =   21 (CP) +   29 (OP), TC 99.41
//  TPs  100 =   42 (CP) +   58 (OP), TC 99.41
//  TPs  150 =   62 (CP) +   88 (OP), TC 99.42
(snipped)
//  TPs 1150 =  128 (CP) + 1022 (OP), TC 99.49
//  TPs 1200 =  135 (CP) + 1065 (OP), TC 99.49
//  TPs 1250 =  148 (CP) + 1102 (OP), TC 99.50
//
//  Test point analysis completed: target estimated test coverage has been
//  achieved.
//  Inserted Test Points                         1297
//    Control Points                              157
//    Observe Points                             1140
//
//
//  Test Coverage Report after Test Point Analysis
//  ----------------------------------------------
//  Target number of random patterns           131072
//
//  Total Number of Faults                      18042
//    Testable Faults                           17528  (  97.15%)
//      Logic Bist Testable                     17435  (  96.64%)
//      Blocked by xbounding                        0  (   0.00%)
//      Uncontrollable/Unobservable                93  (   0.52%)
//
//  Estimated Test Coverage (post test points)            99.03%
//  Estimated Relevant Test Coverage (post test points)   99.50%
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The "Total Number of Faults", the number of "Testable Faults" and the number of "Logic Bist Testable" faults are larger after test point insertion as these numbers include the (testable) faults at the new test point logic.

Example 2

The following example specifies that enough test points should be generated to achieve 99 percent fault coverage (default), but not more than a maximum number of 3000 control points, and a maximum of 2000 observe points:

set_system_mode analysis

set_test_point_analysis_options -control_points 3000 -observe_points 2000

analyze_test_points

report_test_points

insert_test_logic

Example 3

The following example specifies excluded clock domains and uses the set_test_point_analysis_options command to display the excluded clock domains. Then it adds a few more clock domains and displays the excluded clock domains again. The example uses a collection named $some_clocks which is comprised of two clock pins and a collection named $more_clocks which is comprised of four clock pins:

set_test_point_analysis_options -exclude_clock_domains $some_clocks
set_test_point_analysis_options

```
//  Maximum Number of Test Points     :         50
//  Maximum Number of Control Points  :         50
//  Maximum Number of Observe Points  :         50
//  Target Test Coverage              :         99.00
//  Target Pattern Count              :         1000
//  Maximum Control Points per Path   :         65535
//  Exclude Cross Domain Paths        :         off
//  Excluded_Clock_Domains            :         fast_clock1
//                                              fast_clock2
```

set_test_point_analysis_options -exclude_clock_domains $more_clocks
set_test_point_analysis_options

```
//  Maximum Number of Test Points     :         50
//  Maximum Number of Control Points  :         50
//  Maximum Number of Observe Points  :         50
//  Target Test Coverage              :         99.00
//  Target Pattern Count              :         1000
//  Maximum Control Points per Path   :         65535
//  Exclude Cross Domain Paths        :         off
//  Excluded_Clock_Domains            :         fast_clock1
//                                              fast_clock2
//                                              fast_clock3
//                                              fast_clock4
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

This example specifies that no control points can be inserted in clock domain fast_clock1. The report_notest_points command shows the locations in the excluded clock domain (gates U10 to U90). Next the no_control_point attributes for the pins on gate U10 are reset to false. After that the report_notest_points command shows all gates in the excluded clock domain. Note that gate U10 is no longer excluded:

set_test_point_analysis_options -exclude_clock_domains fast_clock1
set_system_mode analysis
report_notest_points

```
/U10/A
/U10/B
/U10/Z
/U11/A
/U11/B
/U11/Z
...
/U90/Z
```

set_attribute_value U10 -name no_control_point -value false

```
{U10}
```

report_notest_points

```
/U11/A
/U11/B
/U11/Z
...
/U90/Z
```

Example 5

In this example, you specify an excluded clock domain after test points have been inserted. An error is reported as this cannot be done until all test points have been deleted:

set_system_mode analysis
analyze_test_points

```
// ...
//  Test point analysis completed: target estimated test coverage has been
//  achieved.
//  Inserted Test Points                          2
//    Control Points                              2
//    Observe Points                              0
// ...
```

set_test_point_analysis_options -exclude_clock_domains fast_clock1

```
//  Error: Excluded clock domains cannot be specified when test points
//  have already been inserted.
//  Please delete all test points before proceeding with specifying (an)
//  excluded clock domain(s).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 6

The number of control points and observe points must both be specified as an integer or as an integer percentage. The following command results in an error.

set_test_point_analysis_options -control_points 30 -observe_points 5%

Example 7

In the following example the tool inserts observe points for blocked faults when you specify the "-observe_blocked_faults on" switch.

set_test_point_analysis_options -observe_block_faults on

analyze_test_points

```
//  Test point analysis completed: target estimated test coverage has been achieved.
//  Inserted Test Points                        134
//    Control Points                            43
//    Observe Points                            91
//      Observe Points for blocked faults       10
```

Example 8

The following example highlights the changes in the test point analysis report when clock controllability analysis is on.

set_test_point_analysis_option -cascaded_icg_synchronization on

analyze_test_points

```
//  Test Coverage Report before Test Point Analysis
//  ----------------------------------------------
//  Target number of random patterns            10000
//
//  Total Number of Faults                      2136342
//    Testable Faults                           2125871  (  99.51%)
//      Logic Bist Testable                     2109360  (  98.74%)
//      Target Fault Testable                   1912193  (  89.51%)
//      Blocked by xbounding                          0  (   0.00%)
//      Uncontrollable/Unobservable                16511  (   0.77%)
//
//  Estimated Maximum Test Coverage                       99.22%
//  Estimated Test Coverage (pre test points)             54.33%
//  Estimated Relevant Test Coverage (pre test points)    50.09%
//
//  Estimated Test Coverage by turning all clock gaters on   70.12%
.
.
//  Test point analysis completed: target estimated test coverage has been achieved.
//  Total number of  test points           1420
//    Control Points                       800
//    Observe Points                       500
//    CG Control Points (always-on)        120(3)
//    Maximum control point per path       5
//    CPU_time (secs)                      4313.2
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 9

This example shows how to report only the control points inserted for improving the clock gaters controllability.

```
report_test_points [get_test_points -control_point_target clock_path]
```

or

```
report_test_points [get_test_points -filter control_point_target == "data_path"]
```

Related Topics

analyze_test_points

report_test_points

set_test_point_insertion_options

set_test_point_types

write_scan_setup

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_test_point_insertion_options

Context: dft -test_points

Mode: setup, analysis

Prerequisites: None

Sets parameters related to test point insertion.

Usage

set_test_point_insertion_options
[-control_point_enable *pin_path_name*]
[-observe_point_enable *pin_path_name*]
[-test_point_clock *system_clocks* | *always_tclk* | *tclk_as_needed*]
[-prefer_scan_cells on | off]
[-capture_per_cycle_en *path_name*]

Description

The set_test_point_insertion_options command specifies parameters that impact the structure of the test logic that is generated when the insert_test_logic command is issued.

Arguments

- -control_point_enable *pin_path_name*

  An optional switch and string pair that specifies the pin and path name of the enable signal for control points. In its absence, the name lbist_en is used at the top level.

- -observe_point_enable *pin_path_name*

  An optional switch and string pair that specifies the pin and path name of the enable for observe points. In its absence, the name lbist_en is used at the top level.

- -test_point_clock *system_clocks* | *always_tclk* | *tclk_as_needed*

  An optional switch and literal pair that specifies what clock to use to drive scan cells for test points.

  The valid literals are as follows:

  system_clocks — A literal that specifies that all test points should be controlled by system clocks. Normally, the test point clock is selected based on analysis of the fan-in/fanout cone. Instead of using test_clock, when this analysis fails to identify a suitable clock, the tool selects the system clock that drives the most scannable cells within the design. This is the default.

  Note

  When selecting an existing system clock, the tool gives preference to clocks that do not drive the set or reset port of any scannable flops. The test clock must pulse during shift, so we do not want to choose a clock that is also connected to the set/reset port of scannable flops since this would cause S2 violations for those flops.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

always_tclk — A literal that specifies to use the user_defined test_clock signal for all sequential elements that are inserted during test point insertion. When this option is specified, the tool takes the global clock name from the -tclk option of the set_scan_signals command.

tclk_as_needed — A literal that specifies to use the user_defined test_clock when no other option is found in the fan-in or fanout cone, or for observe points, when both the fan-in and fanout cone connect to multiple clock domains and false or multicycle paths have been defined.

- -prefer_scan_cells on | off

  An optional switch and literal that controls the search order used to identify a scan cell that controls or observes a test point. The default is on. The tool first looks for a scan cell, and only uses a non-scan cell if a scan cell is not found. The scan is automatically identified by the tool and reported by "report_cell_models" or manually specified by the add_cell_models command.

  When -prefer_scan_cells is off, the tool first looks for a non_scan cell, and uses a scan cell only when the report_cell_models command reports that all available cell models are scan cells.

- -capture_per_cycle_en *path_name*

  (Observation scan only). An optional switch and string pair that enables you to specify the name of the observation scan enable pin rather than use the default name "capture_per_cycle_static_en".

  To insert observation scan test points, either the "capture_per_cycle_static_en" dft signal should be available or you should specify an enable signal for capture_per_cycle_en using the set_test_point_insertion_options command, so the tool assumes observation scan is on.

  If you specify "set_test_point_insertion_options -capture_per_cycle_en" after analyze_test_points, the tool issues the following error message:

```
// Error: The capture_per_cycle_en switch is being specified after
test point analysis. If you want to generate observation scan test
points, delete the existing test points and run test point analysis
again.
```

Examples

Example 1

The following example specifies the names of enables for control points and observe points, respectively. It also specifies that the global clock that was specified with the set_scan_signals command ('test_clk') is to be used to drive the scan cells for the test points:

set_scan_signals -tclk test_clk

...

set_test_point_insertion_options -control_point_enable control_enable

set_test_point_insertion_options -observe_point_enable observe_enable

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_test_point_insertion_options  -test_point_clock always_tclk
```

Example 2

The following example generates a maximum of 5000 test points. Then it specifies the names of enables for control points and observe points respectively, and it inserts the test points into the netlist:

```
set_system_mode analysis
set_test_point_analysis -total_number 5000
analyze_test_points
report_test_points
set_test_point_insertion_options -control_point_enable cp_enable \
        -observe_point_enable op_enable
insert_test_logic
```

Related Topics

add_cell_models

analyze_test_points

insert_test_logic

set_test_point_analysis_options

set_test_point_types

write_scan_setup

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_test_point_sharing_restrictions

Context: dft -test_points

Mode: all

Specify sharing restrictions during test point insertion.

Usage

set_test_point_sharing_restrictions [on | off]
[-instances *instance_spec*]
[-modules *module_spec*]
[-cluster *cluster_list*]

Description

The default implementation of test point sharing uses logical proximity to identify test points that a single flip-flop can control. In addition to logical proximity, the tool ensures that grouped test points are in the same clock and power domain.

You can extend this capability by using this command to specify a region where you want to avoid sharing test points with the rest of the design. You can identify the restricted regions by instances, modules, or cluster_name attributes.

Arguments

- on | off

  An option to toggle whether or not the tool restricts test point sharing within instances and modules.

  on — Directs the tool to restrict the sharing of the given instances or modules.

  off — Directs the tool to remove any restriction during test point sharing. The tool also clears any previous restrictions. This is the default.

- -instances *instance_spec*

  An optional switch and string pair that specifies that the tool restricts the instances for test point sharing to inside the *instance_spec* instances. The *instance_spec* string can be a list or a collection of one or more instances. If the instance is hierarchical, the tool also flags all instances beneath it.

- -modules *module_spec*

  An optional switch and string pair that specifies that the tool restricts the modules for test point sharing to inside the *module_spec* modules. The *module_spec* can be the name of a library model or a collection containing a library model object.

- -cluster *cluster_list*

  An optional switch and string pair that specifies that the tool only shares test points that have scan elements from the same cluster in either their input or output cone of logic. The

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

cluster_name attribute identifies the scan elements in each cluster. You can set the cluster_name attribute by running the set_attribute_value command in the dft -test_point context. For attribute details, see the "Built in Attributes on Scan Element (SE) Object Type" table in the Scan Element Object Type section of the *Tessent Shell Reference Manual*.

Note

The add_scan_mode command defaults to "on" for the -single_cluster_chains switch.

If you perform test point insertion and scan stitching in the same pass, by default the tool uses the existing cluster_name attribute during stitching to restrict the scan chain to a single cluster. You must reset the cluster_name attribute value if you do not intend to restrict the chains to clusters during scan stitching.

Examples

Example 1

The following example shows how to restrict control point sharing for specific instances. Here, control points inside instance A can only be shared with other control points in instance A. The same is true for the control points in instance B.

```
set_test_point_analysis_option -shared_control_points_per_flop 4
set_test_point_sharing_restrictions on –instance { A B }

analyze_test_points
```

Example 2

The following example shows how to set the cluster_name attribute and how to restrict sharing to clusters during test point analysis. In this example, the observe points located in the rx_clust cluster can only be shared together and cannot be shared with any other observe point with a different cluster name. The same is true for the observe points located in the tx_clust cluster.

```
set_context dft -test_points -scan -no_rtl
set_system_mode analysis
set_attribute_value [get_scan_elements -below_instances i_rx_phy] \
        -name cluster_name -value rx_clust
set_attribute_value [get_scan_elements -below_instances i_tx_phy] \
        -name cluster_name -value tx_clust
set_test_point_analysis_option -shared_observe_points_per_flop 4
set_test_point_sharing_restriction on –cluster { rx_clust tx_clust }
analyze_test_points
```

Related Topics

get_test_point_sharing_restrictions

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_test_point_types

Context: dft -test_points

Mode: setup

Specifies the type of test points to insert in the design.

Usage

set_test_point_types [lbist_test_coverage | edt_pattern_count |
{lbist_test_coverage edt_pattern_count}]

Description

Specifies the type of test points to insert in the design. If no argument is specified, the default is both lbist_test_coverage and edt_pattern_count test points.

Arguments

- lbist_test_coverage

  An optional switch that specifies that the test points inserted are for improving random pattern test coverage.

  When you specify this switch:

  - The tool performs X-bounding analysis by default. Any node with an Xsource is assumed to be bounded and therefore fully controllable.
  - All the faults behind X-bounding logic are ignored during analysis. Both incremental test coverage and test coverage estimates are written out before and after test point analysis.
  - The targeted pattern counts and test coverage thresholds are taken into account.
  - The default pattern count is 10000 and test coverage is 99.

  If you provide a fault list, test point analysis tries to improve ATPG test coverage for the fault list provided. In this case, X-bounding analysis is not done by default. The default pattern count is 1000 and the default coverage is 100.

- edt_pattern_count

  An optional switch that specifies that the test points inserted are for reducing deterministic pattern count.

  When you specify this switch:

  - The tool does not perform X-bounding analysis by default.
  - No incremental test coverage and test coverage estimates are written out.
  - If a fault list is provided in conjunction with this option, this is the fault list used to insert the specified number of test points for ATPG pattern count reduction, instead of the full fault list, which may result in non-ideal results.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- lbist_test_coverage edt_pattern_count

  If you specify both types of test points, the tool identifies test points to reduce both pattern count reduction and improve random pattern testability.

  - The tool performs X-bounding analysis by default.
  - Both incremental test coverage and test coverage estimates are written out before and after test point analysis.
  - The targeted pattern counts and test coverage thresholds are taken into account.
  - If a fault list is provided in conjunction with both these options specified together, this is the fault list used to insert the specified number of test points for ATPG pattern count reduction and LBIST coverage improvement, instead of the full fault list which may result in non-ideal results. Coverage improvement may not be as good as it could be when using just the lbist_test_coverage option alone.
  - The default pattern count is 10000 and test coverage is 99.

Examples

The following example sets the test points type for test coverage improvement, performs test point analysis and then reports the test points for your design:

```
set_context dft -test_points -no_rtl
set_test_point_types lbist_test_coverage
set_system_mode analysis
set_test_point_analysis_options -total_number 1000
analyze_test_points
report_test_points
```

Related Topics

add_control_points

add_observe_points

analyze_test_points

get_test_point_type

set_test_point_analysis_options

set_test_point_insertion_options

write_scan_setup

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_test_setup_icall

Context: dft, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: setup

Adds an iCall to the end of the test_setup procedure.

Usage

set_test_setup_icall {[*icl_instance_path*.]***iproc_name*** [*arguments* …]}
[[-merge] -append] {[-front] | [-end] [-non_retargetable]} [-timeplate *icall_timeplate_name*]

Description

The set_test_setup_icall command adds an iCall to the end of the test_setup procedure.

The command accepts a Tcl list consisting of the three components that normally follow an iCall: the name of the iProc, preceded by the optional ICL instance path and separated by a dot, and one or more optional arguments to the iProc.

If you specify the -append switch, the new iCall is appended to the iCalls in memory. Each iCall with its arguments is called in the order they were appended.

If you issue the command without the -append switch, it overwrites the previous values. To clear the list, assign an empty string or empty list without the -append switch.

The -merge switch provides support for iMerge in the test procedure file. It can only be used if the -append switch is also specified.

If the test_setup procedure contains iCalls, then Tessent Shell infers input constraints on the ports with ICL port function DataInPort. For more details, see the description of the add_input_constraints command.

To keep the iCalls of the set_test_setup_icall command in the correct order, the iCall can be added to the front, middle, or end list.

The iCalls from the front list are executed first. The iCalls from the middle list are executed second. The iCalls from the end list are executed last.

By default, the targeted list that the new iCall is added to is the middle list, but you can override this default with the following options:

- -front — This option adds the iCall to the front list.
- -end — This option adds the iCall to the end list.

The -append switch adds the iCall to the end of the targeted list. If you add an iCall without using the -append switch, then all the previous iCalls of the targeted list are cleared. Note: This only clears all the iCalls of the targeted list, not all the iCalls of all three lists. When you add an

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

iCall with the -merge switch, then this iCall is merged with the previous iCall on the same targeted list if a previous iCall exists.

Additionally, you can add report_test_setup_icall and report_test_end_icall commands to report iCalls configuration in test_setup and test_end procedures.

If you need to clear the test_setup_icall list, specify either quotes ("") or braces ({}) as an argument to the set_test_setup_icall command. Both arguments work the same and the list is cleared.

Arguments

- {[*icl_instance_path*.]***iproc_name*** [*arguments* …]}

  The name of the iProc, preceded by the optional ICL instance path and separated by a dot, and one or more optional arguments to the iProc.

- -append

  An optional switch that appends the specified list of iCalls to the accumulated list of iCall commands. Note that if you issue the command without the -append switch, the accumulated iCalls are cleared and only the current specified iCall command is stored.

- -merge

  An optional switch to merge iCall commands together. The -merge option can only be used with the -append switch to merge the specified iCall with the previous iCall. All the merged iCall commands infer the -error_on_conflict option of the iMerge command and cannot be unspecified.

  Refer to "iMerge Conflict Reporting" in the *Tessent IJTAG User's Manual* for more information on the -error_on_conflict option.

  The merged iCall commands can be specified in the test_setup/test_end procedures using the Tessent Shell commands set_test_setup_icall/set_test_end_icall. Later, when you use the Tessent Shell command report_procedures (available in analysis mode), the merged iCall commands of the test_setup/test_end procedures appear as follows:

  ```
  iMerge
     iCall iProcName iProcArgs
     ....
  iMerge -end;
  ```

  The -error_on_conflict option is always enabled, so it is not necessary to specify it explicitly.

- -front

  An optional switch that instructs the tool to add the specified iCall to the first list that causes it to run first.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -end

  An optional switch that instructs the tool to add the specified iCall to the first list which that causes it to run last.

- -non_retargetable

  An optional switch that enables adding iCalls that are not retargeted at the next level. Use this switch to apply the iCall at the current (core) level and not forward it when the core is embedded in a design. This is especially needed to handle BISR programming when memories are used during scan test.

- -timeplate *icall_timeplate_name*

  An optional switch and string pair that specifies an iCall timeplate name other than the default.

  There is a limitation when your test procedures have iCalls with specified timeplates that are other than the default. Specifically, when you write the procedure file using the write_procfile command, the tool writes the iCalls, but not the specified timeplates. When you read the procedure file back into the tool, the tool solves the iCalls with the default timeplate, not the specified timeplate.

  To work around this limitation, use the "write_procfile procfilename -expand_iprocs" command and switch. Doing this ensures the iCalls and specified timeplates are written out.

Examples

Example 1

The following example adds an iCall to the end of the test_setup procedure, then appends and merges a second iCall. The get_test_setup_icall_list command displays the added iCalls.

```
set_test_setup_icall "m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on \
                      edt_bypass off"

set_test_setup_icall "m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on \
                      edt_bypass off" -append -merge

get_test_setup_icall_list

{m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on edt_bypass off}
{m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on edt_bypass off}
```

Example 2

In the following example, the iCall commands specified using the -merge option in the test_setup file merge iProc2, iProc3 and iProc4 into iProc1:

```
set_test_setup_icall iProc1 -append
set_test_setup_icall iProc2 -append
set_test_setup_icall iProc3 -merge -append
set_test_setup_icall iProc4 -merge -append
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

These commands are equivalent to the sequence of the iCall commands in the test procedure file:

```
Procedure test_setup =
        iCall iProc1;
        iMerge =
            iCall iProc2;
            iCall iProc3;
            iCall iProc4;
        end;
end;
```

Example 3

The following command sequence constructs three lists of iCalls. The report_test_setup_icall command is then used to report all of the existing iCalls that were specified with set_test_setup_icall:

```
set_test_setup_icall userProc_1 -front
set_test_setup_icall userProc_2 -front -append
set_test_setup_icall userProc_3 -append
set_test_setup_icall userProc_4 -append -merge
set_test_setup_icall userProc_5 -append -end
set_test_setup_icall userProc_6 -append -end
set_test_setup_icall toolProc_1 -front -append
set_test_setup_icall toolProc_2 -append
set_test_setup_icall toolProc_3 -append -merge

//  first list of iCalls with -front:
//    userProc_1
//    userProc_2
//    toolProc_1

//  second list of iCalls without -front and -end
//    userProc_3
//    userProc_4 -merge
//    toolProc_2
//    toolProc_3 -merge

//  third list of iCalls with -end
//    userProc_5
//    userProc_6
report_test_setup_icall
```

```
iCall userProc_1
iCall userProc_2
iCall toolProc_1
iMerge -begin
    iCall userProc_3
    iCall userProc_4
iMerge -end
iMerge -begin
    iCall toolProc_2
    iCall toolProc_3
iMerge -end
iCall userProc_5
iCall userProc_6
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

get_test_end_icall_list

get_test_setup_icall_list

iCall

report_test_end_icall

report_test_setup_icall

set_test_end_icall

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_testbench_simulation_options

Context: unspecified, all contexts

Mode: all modes

Specifies the default values for a set of options that exist in the run_testbench_simulations command.

Usage

set_testbench_simulation_options [-default_simulator {questa | vcs | incisive}]
[-simulation_output_directory *directory*]
[-parallel_simulations {1 | 2...*MAX_INT* | maxcpu}]
[-keep_simulation_data {on | off | on_failure}]
[-store_simulation_waveforms {on | off}]
[-simulation_timeout {*time* | unlimited}]
[-compilation_options *option_list*]
[{questa | vcs | incisive} [-simulator_options *option_list*]
[-simulation_run_commands *simulator_commands*]
[-waveform_configuration_commands *simulator_commands*]]

Description

Specifies the default values for a set of options that exist in the run_testbench_simulations command.

This command also configures the default value of the -simulation_output_directory_list of the check_testbench_simulations command.

Arguments

- -default_simulator questa | vcs | incisive

  An optional switch and value pair that, when specified, sets the default value for the run_testbench_simulations option of the same name.

- -simulation_output_directory *simulation_output_directory*

  An optional switch and value pair that, when specified, sets the default value for the run_testbench_simulations option of the same name. It also sets the default value of the check_testbench_simulations -simulation_output_directory_list option.

- -parallel_simulations {1 | 2...*MAX_INT* | maxcpu}

  An optional switch and value pair that, when specified, sets the default value for the run_testbench_simulations option of the same name. *MAX_INT* is the maximum integer value of the host machine. The maxcpu literal specifies using all available CPUs on the host machine.

- -keep_simulation_data on | off | on_failure

  An optional switch and value pair that, when specified, sets the default value for the run_testbench_simulations option of the same name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -store_simulation_waveforms on | off

  An optional switch and value pair that, when specified, sets the default value for the run_testbench_simulations option of the same name.

- -simulation_timeout time | unlimited

  An optional switch and value pair that, when specified, sets the default value for the run_testbench_simulations option of the same name.

- -compilation_options *option_list*

  An optional switch and value pair that specifies a list of compilation options (for example, +initreg) that modify the compilation script during runtime. The *option_list* uses the following syntax:

  -compilation_options {[*global_options*] [verilog *verilog_options*}] [vhdl {*vhdl_options*}]}

  - The verilog options are passed only to the command that compiles Verilog source files.
  - The vhdl options are be passed only to the command that compiles the VHDL source files.
  - The *global_options* can also be expressed as "global {*global_options*}".

  The braces are mandatory in order to identify the options preceded by the language or "global".

  See Example 4.

- questa | vcs | incisive

  A simulator literal that enables you to specify default values for options that pertain to the specified simulator.

- -simulator_options {-timescale *value* ...}

  An optional simulator-specific switch and value pair that, when specified, sets the default value for the run_testbench_simulations option of the same name. This can only be specified with one simulator literal.

  The -timescale *value* option overrides the default timescale for a specific simulator for the following default values:

  - incisive — Default timescale is 1ns/1ps

- -simulation_run_commands *simulator_commands*

  An optional simulator-specific switch and value pair that, when specified, sets the default value for the run_testbench_simulations option of the same name. This can only be specified with one simulator literal.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -waveform_configuration_commands *simulator_command_list*

  An optional simulator-specific switch and value pair that, when specified, sets the default value for the run_testbench_simulations option of the same name. This can only be specified with one simulator literal.

Examples

Example 1

The following command sets the default value of the run_testbench_simulation -keep_simulation_data option to on. It demonstrates using a simulator-independent option.

```
set_testbench_simulation_options -keep_simulation_data on
```

Example 2

The following command sets the default value of the run_testbench_simulation -simulator_options passing the simulator-specific option of -novopt. This is a simulator-specific option so the simulator literal (questa) is specified. The options take effect when you use run_testbench_simulations -simulator questa, or when the default simulator is questa and you specify run_testbench_simulations without the "-simulator" argument.

```
set_testbench_simulation_options questa -simulator_options {-novopt}
```

Example 3

In the following example, the options take effect when you use run_testbench_simulations -simulator incisive, or when the default simulator is incisive and you specify run_testbench_simulations without the "-simulator" argument.

```
set_testbench_simulation_options incisive -simulator_options \
{+nctimescale+1ns/100ps +access+rwc+nowarn+CUVWSP +nowarn+CUVWSI
+nowarn+CUVWSP}
```

Example 4

The following example demonstrates using the -compilation_options switch. In this example, the run_testbench_simulations command uses the VCS simulator and the -compilation_options value is set by the set_testbench_simulation_options command.

```
set_testbench_simulation_options vcs \
    -compilation_options { global { -kdb } vhdl { -skip_translate_body }
     verilog { -timescale=1ns/1ps } } -default_simulator vcs

run_testbench_simulations
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_tester_options

Context: patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting, patterns -silicon_insight

Mode: setup, analysis

Enables you to define tester-specific options that control how the tool writes tester patterns when you use SSN.

Usage

set_tester_options [-timeplate_constraints {same_period | none}]
[-use_timing_variables {on | off}]
[-timing_variables_group_name *timing_variables_group_name*]
[-min_time_unit {auto | *time*}]

Description

Depending on the capabilities of your ATE, you may need to manage timeplate creation. Low-cost ATEs require significant time to change the tester cycle period.

Commonly-used testers, such as the Advantest 93K, efficiently support timeplate constraints with the same period but have a prohibitively high cost of switching to a timeplate with a different frequency.

This command enables you to create multiple timeplate constraints with the same period or with different periods, depending on the capabilities of your ATE.

You can use the get_tester_option command to check the value of one of the options set by the set_tester_options command, or the report_tester_options command to check on all of the values set by the set_tester_options command.

Note

These tester options apply only to designs with SSN.

Arguments

- -timeplate_constraints same_period | none

  An optional switch and value pair that specifies whether timeplate constraints are created with the same period or with different periods.

  same_period — In this mode, the tool can create multiple timeplates, but all must have the same period.

  This is the default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

When all timeplate constraints use the same period, each clock pulse in test_setup is globally stretched across multiple fast cycles. As such, the test time remains the same, but the cycle count increases.

This global cycle stretching applies to patterns you write out but does not apply to test_setup simulation within the tool. So, "set_gate_reporting drc test_setup" does not show the global cycle stretching.

none — In this mode, the tool can create multiple timeplates with different periods.

Note

The -timeplate_constraints switch is available only in setup mode. Using this switch in analysis mode causes the tool to report an error.

- -use_timing_variables on | off

  An optional switch and literal pair that specifies to use timing variables to model timeplates for all tool-generated timeplates in the SSN flow. Using timing variables can make data available for a shmoo plot. This behavior is on by default.

- -timing_variables_group_name *timing_variables_group_name*

  An optional switch and string pair that changes the name of the STIL spec variables category. Valid names must consist only of letters, digits, and underscores (_).

- -min_time_unit {auto | *time*}

  An optional switch and literal or value/unit pair that specifies the minimal time unit for the timeplate period when SSN is present. By default, the tool automatically selects the appropriate minimal time unit. If you require a specific minimal time (for example, if you use an emulator that has a minimum precision limit), specify a value/unit pair, such as "0.25ns". The write_patterns command applies the specified granularity value. In this case, the tool adjusts the original timeplate event times by delaying events within a given timeplate to meet the requested minimal time unit. This can result in an increase in the overall timeplate period. A <*time*> value of 0 does not require a unit and removes all 1 ps delays added to the timeplate.

  Caution

  A <*time*> value of 0 is likely to cause zero-delay simulation to fail, because the delays enable the simulator to correctly queue events.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_tied_signals

Context: all contexts

Mode: setup

Changes the default value for floating pins and floating nets which do not have assigned values.

Usage

set_tied_signals **X** | **1** | **0** | **Z**

Description

The set_tied_signals command specifies the default value that the tool ties to all floating nets and floating pins that you do not specify with the add_tied_signals command. Upon invocation of the tool, if you do not assign a specific value, the tool assumes the default value is unknown (X).

If the model is already flattened and then you use this command, you must delete and recreate the flat model.

Arguments

- **X**

  A literal that ties the floating nets or pins to unknown. This is the default upon invocation of the tool.

- **0**

  A literal that ties the floating nets or pins to logic 0 (low to ground).

- **1**

  A literal that ties the floating nets or pins to logic 1 (high to voltage source).

- **Z**

  A literal that ties the floating nets or pins to high-impedance.

Examples

The following example ties floating net vcc to logic 1, ties the remaining unspecified floating nets and pins to logic 0, then performs an ATPG run:

```
set_tied_signals 0
add_tied_signals 1 vcc
set_system_mode analysis
create_patterns
```

Related Topics

add_tied_signals

delete_tied_signals

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_tied_signals

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_timing_exceptions_handling

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Changes the default handling of timing exception paths (false paths and multicycle paths) read by the read_sdc command.

Usage

```
set_timing_exceptions_handling [ON | OFf]
    [-CLOCK_groups {ON | OFf}]
    [-ADD_Hold_time_when_no_settings_in_sdc {ON | OFf}]
    [-LIne_no {ON | OFf}] [-SHOW_Max_skew {ON | OFf}]
    [-SETUp {ON | OFf}]
    [-MAX_Skew {HALf_cycle | ONE_cycle | MULtiple_cycles}]
    [-APPLY_Clock_scaling {WIthin_each_source_clock | ACross_all_source_clocks | OFf}]
    [-DERIVE_Fault_detection_cycles_from_SDC_generated_clocks {ON | OFf}]
    [-DERIVE_Mcp_from_sdc_clocks {ON | OFf}]
    [-X_Statistics {ON | OFf}]
    [-SLOW_Cycle_mask {ON | OFf}]
    [-ALLOW_Invalid_pin_names {ON | OFf}]
    [-MERGE_compatible_paths {ON | OFf}]
    [-sdc_clock_check {ON | OFf}]
```

Description

Changes the default handling of timing exception paths (false paths and multicycle paths) read by the read_sdc command and how certain report commands display information about these paths.

When you run the set_timing_exceptions_handling command without options, the command reports the current settings.

Arguments

- ON | OFf

  A literal that enables or disables handling of timing exceptions on false paths and multicycle paths. This capability is enabled by default upon running the tool for any fault types. Turning off this switch is useful for debugging low coverage or simulation mismatches because you can quickly verify if the masking values or test coverage loss is due to timing exceptions. Turning off this switch can result in pattern mismatches, so you should not turn off this switch other than for debugging purposes.

- -CLOCK_groups {ON | OFf}

  A switch and literal pair that changes the clock groups handling in SDC. By default, the tool expands timing exceptions for the clocks in asynchronous clock groups as both setup and hold timing exceptions only when the clocks can be pulsed by ATPG concurrently. If you

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

are not generating any patterns that could pulse asynchronous clocks concurrently during ATPG for the ones defined in the SDC files, you can disable this path expansion by specifying OFf.

If you use "-clock_groups on" and there are already false paths derived from existing clock groups, the tool reports the following message:

```
// Warning: The clock groups have been expanded into false paths.
You must delete SDC to remove false paths derived from clock groups.
```

To avoid this issue, use the "set_timing_exceptions_handling -clock_groups off" command prior to running read_sdc.

- -ADD_Hold_time_when_no_settings_in_sdc {ON | OFf}

  A switch and literal pair that treats the timing exceptions as including both setup and hold-time violations when the SDC files specify neither -setup nor -hold. By default, the tool matches the SDC file used by your static timing analyzer, which is useful if your design must include hold timing exceptions during stuck-at ATPG. When you disable this switch, the tool assumes that all false paths without -setup or -hold options have only setup timing exceptions.

  Table 8-22 further explains how the tool treats timing exception paths based on the setting in the SDC file and the -ADD_Hold_time_when_no_settings_in_sdc switch.

Table 8-22. SDC File and -ADD_Hold_time_when_no_settings_in_sdc switch

SDC file switches	Default tool behavior	Tool behavior with switch OFF
none	setup & hold	setup *
-setup	setup	setup
-hold	hold	hold
-setup -hold	setup & hold	setup & hold
* True unless the timing exception path is added across clock domains, in which case the path is treated as both setup & hold. As an example, the following SDC commands are equivalent: set_false_path -from {get_clocks {CLK1}} -to {get_clocks {CLK2}} set_false_path -from {get_clocks {CLK1}} -to {get_clocks {CLK2}} -setup -hold		

- -LIne_no {ON | OFf}

  An optional switch and literal pair that turns on reporting of SDC filename and line numbers for timing violations when using the report_false_paths or the report_multicycle_paths command. By default, the tool does not report the SDC filename and line numbers.

- -SHOW_Max_skew {ON | OFf}

  An optional switch and literal pair that causes the report_false_paths command to always show the max skew setting for false paths that contain -hold. When this is disabled (the default setting), the max skew setting is only shown when the max skew is not half_cycle

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

and not the same as the global max_skew setting from the -max_skew argument to this command.

- -SETUp {ON | OFf}

  An optional switch and literal pair that disables setup time simulation without having to modify the SDC file. By default, setup time simulation is enabled.

  Note

  The tool automatically reclassifies AU faults when you change this option, so there is no need to issue a "reset_au_faults -reclassify" command.

- -MAX_Skew {HALf_cycle | ONE_cycle | MULtiple_cycles}

  An optional switch that specifies the maximum clock skew the tool can use when simulating the hold time exception. This option impacts the masking effect of most hold time exceptions defined using either the add_false_path command or the hold time exceptions from read_sdc. This option does not impact false paths added with the "add_false_paths -cross_clock_domains" command and the false paths derived from the asynchronous clock groups in SDC files.

  HALf_cycle

  Specifies half cycle as the maximum clock skew. This is the default value for this option, unless -cross_clock_domains is specified, in which case, the max skew is multiple cycles.

  ONE_cycle

  Specifies one cycle as the maximum clock skew.

  MULtiple_cycles

  Specifies multiple cycles as the maximum clock skew.

- -APPLY_Clock_scaling {WIthin_each_source_clock | ACross_all_source_clocks | OFf}

  An optional switch and literal pair that specifies the behavior of multicycle path (MCP) cycle scaling.

  For synchronous clock domains with different frequencies (either within each source clock or across all source clocks), the tool scales the clocks according to their frequency information provided in the SDC file. In cases when the driving clock of a multicycle path has a different frequency source clock, this switch defines how the tool should determine their relationship. For example, consider a multicycle path with 2 cycles defined with -end in the SDC file. If the frequency of the end-point driving clock is twice as slow as its source clock, the tool scales the clocking so that the path is treated as a multicycle path with 4 cycles.

  This switch has the following options:

  WIthin_each_source_clock — Applies the cycle scaling only for the generated clocks defined in SDC using the create_generated_clock command with the same source

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

clock. Treats different source clocks independently, and treats all generated clocks with the same source clock as synchronous clocks. This is the default.

ACross_all_source_clocks — Applies the cycle scaling across all source clocks. This option is applicable when treating all source clocks as synchronous.

OFf — Does not apply cycle scaling, so the tool uses the cycle count as defined in SDC set_multicycle_path commands.

- -DERIVE_Fault_detection_cycles_from_SDC_generated_clocks {ON | OFf}

  An optional switch and literal pair that ignores SDC-generated clocks from which to derive the fault detection cycles. By default, the tool permits ATPG and the fault simulator to consider the transition faults in slower clock domains to be detected with multiple activation cycles without the need to define multicycle paths. For example, if a transition fault is on a divide-by-2 clock domain, the fault can be activated for up to 2 cycles before capture. The switch default is ON, which instructs ATPG and the fault simulator to consider the transition faults activated up to *n* cycles for divide-by-*n* clock domains unless explicitly specified using multicycle paths. The tool takes into account the SDC command "create_generated_clock -divide_by", but only in the absence of the -multiply_by switch. The SDC command "create_generated_clock -multiply_by" is unsupported.

  Tip

  You can use the report_clocks command to list all of the SDC-defined clocks.

- -DERIVE_Mcp_from_sdc_clocks {ON | OFf}

  An optional switch and literal pair that derives multicycle paths from SDC generated clocks with divide_by ratios greater than one. If you do not specify this option and SDC files include clocks with a divide_by ratio greater than one, you may get simulation mismatches when the test clock frequency is higher than the specified SDC clock frequency; the tool reports a warning in this case. Specifying this option when it is not required may result in overmasking, thus reducing the test coverage.

- -X_Statistics {ON | OFf}

  An optional switch and literal pair that collects statistics on masking that result from timing exceptions when running the create_patterns or simulate_patterns commands. When you enable this option before pattern generation or simulation, the tool collects statistics about the number of Xs captured into each to-point of timing exception paths. This option is useful to identify the timing exception paths with the most masking effect that could impact the test coverage. Collection of masking statistics is disabled by default. For an example of using this switch, refer to "Example 7" on page 1837.

- -SLOW_Cycle_mask {ON | OFf }

  An optional switch and literal pair that treats the slow cycles defined in the named capture procedure as at-speed cycles and applies all timing exceptions during pattern simulation. By default, the slow cycles are not considered as at-speed cycles and setup timing exceptions are not considered.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -ALLOW_Invalid_pin_names {ON | OFF}

  An optional switch and literal pair that specifies to report invalid SDC commands. The default is on. Setting the switch off disables reporting.

  When reporting is on, the read_sdc and add_false_paths commands report warnings when the tool finds invalid design names such as missing pins. The tool prints a summary message for the number of timing exception paths with invalid pins after SDC analysis.

  A path with invalid pins is treated as a partial path. The tool ignores a whole path only when all pins are invalid and specified in the same -from, -through, or -to switch. A path such as this has no cone in the current flat model.

- -MERGE_compatible_paths {ON | OFF}

  An optional switch and literal pair that merges compatible false paths. To enhance performance in static false path analysis and dynamic false path simulation, by default, the tool merges simple, compatible false paths into a single path.

  The tool can only merge false paths if they are error free, they have exactly the same "setup time" and "hold time" properties, and they meet any of these criteria:

  - All the false paths that contain from-points only, and no through-points, to-points or any other conditions can be merged into a single false paths that combines all the from-points.
  - All the false paths that contain to-points, and no from-points, through-points, or any other conditions, can be merged into a single false path that combines all the to-points.
  - All the false paths that contain one-level through-points only, and no from-points, to-points, or any other conditions can be merged into a single false path that combines all the through-points.

  By default the tool merges compatible false paths and issues a summary message. For example:

```
// Note: N false paths have been merged into M paths to speed up
   timing exception analysis.
//       To disable merging for debug, you may invoke
   'set_timing_exceptions_handling -merge_compatible_paths
    off'.
```

  When reporting false paths, the tool reports the merged paths and includes their original paths, as annotated with the "|+" symbol. For example:

```
//  command: report_false_paths
//  False path -from clk4 -through /u7  -setup (id=0)
//  False path -from clk5 -through /u8  -setup (id=1)
//  False Path -from clk1 clk2 clk3 -setup -hold  (id=2) (merged)
//  |+ False Path -from clk1 -setup -hold
//  |+ False Path -from clk2 -setup -hold
//  |+ False Path -from clk3 -setup -hold
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You cannot add or delete merged paths directly. You can only add or delete the original paths and let the tool re-merge them. All the original false paths are preserved in the flat model. You can disable this feature by using the -off option for this switch.

Also, when you invoke switches such as “set_timing_exceptions_handling -x_statistics on”, the tool assumes you are debugging based upon original paths and disables the “merge_compatible_paths” feature, unless you specify it as “on” in the same command line.

- -sdc_clock_check {ON | OFf}

  An optional switch and literal pair that specifies whether timing exception masking should check if the launch and capture clocks of the timing exception path are pulsed when masking the path. By default, this check is enabled, so that the masking effect is more precise and avoids over-masking. In some cases, SDC constraints may cause ATPG to avoid masking paths that result in simulation mismatches. In these cases, disabling this check avoids these mismatches without having to change the SDC files.

  For example, an SDC file used in functional mode may constrain an SDC clock that is also used to define some false paths that ATPG must consider. If you are unable to obtain an SDC file revised for ATPG test mode, you can use this option to force the tool to consider the false paths on the constrained SDC clock.

  Note

  Disabling this check by specifying “-sdc_clock_check off” can result in over-masking, reducing test coverage.

Examples

Example 1

The following example disables expanding timing exception paths for the clocks in asynchronous clock groups as both setup and hold timing exception paths.

set_timing_exceptions_handling -clock_groups off

Example 2

The following example shows how to specify reporting of SDC filename and line numbers for timing violations:

set_timing_exceptions_handling -line_no on
read_sdc timing.sdc
report_false_paths

```
//  False Path -from_clock cmem_fclki cmem_fclkin -to_clock p_clk
    spm_pci_clk -setup  (filename =  \
     timing.sdc, line 406)
//  False Path -from_clock p_clk spm_pci_clk -to_clock cmem_fclki
    cmem_fclkin -setup  (filename =  \
     timing.sdc, line 408)
...
//  Total reported false paths = 177
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

This example shows how to specify reporting merged false paths:

set_timing_exceptions_handling -merge_compatible_paths on
report_false_paths -all

```
//  Note: 2 false paths have been merged into 1
    path to speed up timing exception analysis.
//        To disable merging for debug, you may invoke
     'set_timing_exceptions_handling -merge_compatible_paths off'.

//  False Path -from_clock U1/CP U2/CP -setup -hold
//  False Path -from_clock U3/CP -to U8/CP -hold
//  False Path -from nclk6 -setup -hold
//  False Path -through G7/B G3/A nclk7 G3/A -setup -hold (merged)
//   |+  False Path -through G7/B G3/A -setup -hold
//   |+  False Path -through nclk7 G3/A -setup -hold
```

Related Topics

read_sdc

report_clocks

report_false_paths

report_multicycle_paths

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_tla_loop_handling

Context: all contexts

Mode: setup

Specifies how to simulate feedback loops that contain transparent latches (TLAs).

Usage

set_tla_loop_handling [ON | OFf]

Description

By default, TLAs in feedback loops are simulated using the same iterative algorithm used to simulate loops through combinational logic. This can increase test coverage; however, it requires feedback loop analysis to be performed whenever DRC is run, which may result in a slightly longer DRC run time. When this command is disabled, feedback loop analysis is performed only after flattening; however, TLAs that are in feedback loops are treated as TIEX gates during simulation, which can reduce test coverage.

Note

Be aware that extra processing time is required to support loop simulation through TLAs. In some circumstances, the improved fault detection yields sufficient reduction in ATPG effort that overall runtimes decrease, but you should anticipate an increase in runtimes in most cases.

Any TLA that is in a feedback loop is treated as a place to break the loop (similar to the FB buffer gate used for combinational loops). Strongly connected sets of gates are identified as feedback networks. You can display feedback networks by using the report_feedback_paths command.

You can use the -Environment switch of the write_patterns command to save the value of the -Iteration switch. You can use the report_environment command to see the current value of the set_tla_loop_handling command.

Arguments

- ON

  An optional literal that enables TLA loop simulation. This is the default.

- OFf

  An optional literal that disables TLA loop simulation.

Related Topics

report_environment

report_feedback_paths

set_loop_handling

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_tool_options

Context: all contexts

Mode: all modes

Sets certain options for Tessent Shell.

Usage

set_tool_options [-reapply_settings_after_reelaboration {on | off}]
[-allow_vhdl_2008 {on | off}]
[-instantiate_unconnected_pins_in_written_design {on | off}]

Description

This command sets certain options in Tessent Shell.

Arguments

- -reapply_settings_after_reelaboration on | off

  When using set_current_design on the current design again the design is re-elaborated. When this setting is set to "on" Tessent Shell preserves certain settings, if possible. These are the settings done with the following commands:

  - add_clocks
  - add_black_boxes
  - add_output_masks
  - add_primary_inputs
  - add_primary_outputs
  - add_input_constraints
  - add_tied_signals
  - set_output_masks

  Note that all settings are removed independent of this setting if you set a different current design.

- -allow_vhdl_2008 on | off

  VHDL 2008 has a different set of IEEE STD packages compared to previous VHDL formats. You must specify to Tessent Shell if any blocks contain VHDL 2008 constructs in order to load 2008 packages. This has to be done before issuing any read_verilog/read_vhdl commands.

- -instantiate_unconnected_pins_in_written_design on | off

  An optional switch and Boolean that specifies how the tool instantiates unconnected pins. The default is on, meaning that the tool instantiates pins even if they have no connection.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When this switch is set to off and the tool is in -no_rtl mode, the unconnected pins are not used in the following:

- Read in instances
- Created instances
- For additional pins that are created by a replace_instances command

When this switch is set to off and the tool is in -rtl mode, the unconnected pins are not used in the following:

- Created instances
- For additional pins that are created by a replace_instances command

In -rtl mode, read in instances are not changed because of format preservation.

Related Topics

get_tool_option

set_tcl_shell_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_trace_flat_model_options

Context: all contexts

Mode: setup, analysis

Modifies the default behavior of the trace_flat_model command.

Usage

set_trace_flat_model_options [-tag_condition *tag_attribute_expression*]
[-stop_condition *stop_attribute_expression*]
[-controllability {controlling | constant | unblocked | connected}]
[-stop_at_first_tag {ON | OFf}] [-max_levels {unlimited | *number*}]
[-latch {normal | transparent}] [-map_tag_to_design_module_boundary {ON | OFf}]

Description

Modifies the default behavior of the trace_flat_model command.

This command sets options as global defaults. These defaults are always applied if they are not specified in the current trace_flat_model execution.

This command removes the requirement for you to repeatedly specify trace configuration information for each call to the trace_flat_model command.

Arguments

- -tag_condition *tag_attribute_expression*

  An optional switch and string pair that specifies the tag condition for the trace. All of the reached design objects fulfilling the defined tag condition are returned. The default is an empty string; in this case, the trace stop points are returned. In a non-default case, the trace stop points are not returned if the tag condition expression is not true for these objects. The format for the *tag_attribute_expression* is identical to the -filter option used for the get_pins command.

- -stop_condition *stop_attribute_expression*

  An optional switch and string pair that defines the stop condition for the trace. The default is an empty string; in this case, the trace is stopped at the gate_pin failing the specified controllability requirements. Endpoints such as primary I/Os and pins of memory or DFF primitives are always stop points even when not explicitly specified.

- -controllability {controlling | constant | unblocked| connected}

  An optional switch and literal pair that defines the controllability requirements before traversing a gate.

  controlling— Gates are only traversed if the other inputs have simulated constant values making a single input directly controlling the output without an inversion,

  constant — Only pins with a constant value are traversed. This mode is useful to find the source of a constant value.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

unblocked — Gates are only traversed if the other inputs do not have simulated constant values making the input blocked from the output pin. Only unblocked combinational paths are traversed.

connected—With this value, gates are always traversed when reached.

- -stop_at_first_tag {ON | OFf}

  An optional switch and literal pair that specifies whether to interrupt tracing when the first tag object with the specified controllability is found. This is useful when checking for at least one occurrence of a given condition in the fanin or fanout of a pin without wasting time to find them all.

- max_levels {unlimited | *number*}

  An optional switch and value pair that defines the maximum level of gates to trace. The default is unlimited but you can specify a positive integer to limit the level count.

- -latch {normal | transparent}

  An optional switch and literal pair that specifies the behavior of tracing through a latch.

  normal — A latch is traced through only if the simulated value in the current simulation context makes the latch satisfy the specified controllability requirements. In case of "-controllability connected" all latch input pins - including the latch enable pin - are traced through.

  transparent — All latches are traced through the data input, set and reset pins regardless of the value at the latch enable pin. The latch enable pin is not traced through.

- -map_tag_to_design_module_boundary {ON | OFf}

  An optional switch and literal pair that specifies whether to map the tag points to gate_pins outside of library cells. When this switch is ON, the trace_flat_model command maps the tag points to gate_pins outside of library cells and never inside the cells. Library cells can be either cells from a Tessent Cell library or `celldefine Verilog modules.

Examples

The following example script shows how the set_trace_flat_model_options command uses attribute expressions to constrain the trace for level 1 clock gaters.

```
set stop_condition_expr " (isClockGater == true) || (isRegister == true) "
set tag_condition_expr "isClockGater == true"

set_trace_flat_model_options -tag_condition  ${tag_condition_expr}  \
   -stop_condition ${stop_condition_expr}
foreach_in_collection pin [all_clocks] {
    set CGs [trace_flat_model -from $pin -direction forward]
        if {[sizeof_collection $CGs] > 0} {
            puts "Level1 clock gaters for clock ${pin} are:"
            foreach_in_collection CG ${CGs} {
                puts "  ${CG}"
            }
        }
    }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

get_trace_flat_model_option

trace_flat_model

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_trace_report

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether the tool displays gates in the scan chain trace.

Usage

set_trace_report **OFf** | **ON**

Description

Specifies whether the tool displays gates in the scan chain trace.

The set_trace_report command controls whether the tool displays all of the gates in the scan chain trace during rules checking.

Arguments

- **OFf**

  A literal that specifies for the tool not to display gates in the scan chain trace. This is the default behavior upon invocation of the tool.

- **ON**

  A literal that specifies for the tool to display gates in the scan chain trace during rules checking.

Examples

Example 1

The following example displays the gates in the scan chain trace during rules checking:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 outdata4
set_trace_report on
set_system_mode analysis
```

Example 2

The following example displays the gates in the scan chain trace during rules checking:

```
add_clocks 0 clock
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 outdata4
set_trace_report on
set_system_mode analysis
```

Related Topics

add_scan_chains

report_scan_chains

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_transcript_style

Context: unspecified, all contexts

Mode: all modes

Specifies the style in which the tool transcribes commands.

Usage

set_transcript_style {**off** | **full** | **tool_commands_only** | **input_only**}
[-result_collection_limit {unlimited | *limit*}]
[-argument_collection_limit {unlimited | *limit*}]
[-one_collection_entry_per_line {on | off }]

Description

You can control the tool's transcript in several ways. Use the set_transcript_style command to do this.

Arguments

- **off**

  A keyword that turns off all command transcription so that the commands run as though included using the Tcl "source" command. This option enables you to execute scripts that do not echo the tool commands, which can be helpful running scripts that use the report commands as required for introspection.

- **full**

  A keyword that transcribes all commands read from a dofile before any Tcl evaluation is done. This includes all Tcl commands and constructs.

  Since the command interpreter accumulates input until receiving a complete Tcl command before sending to Tcl for evaluation, entire if/else, for, and foreach constructs transcribe as a single item. In addition, when non-introspection tool commands are embedded within another command, they transcribe as they are about to execute. (Introspection commands are commands that simply return information about the design or tool state and therefore have no effect on operation of the tool. Tool commands included within the catch_output command are also considered introspection commands since they do not write to the transcript.)

  As in a standard Tcl shell, any result returned from each top-level Tcl command transcribes when you run the tool in interactive mode. In batch mode, this option does not transcribe command results.

- **tool_commands_only**

  A keyword that transcribes tool commands, including all arguments, as the commands run. The command string is transcribed after being evaluated by Tcl so all variable and command references are resolved. For commands in a loop or if/else construct, the tool transcribes each execution of each command plus any output produced so that the context is evident.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This option does not transcribe Tcl flow control constructs, pure Tcl commands, and introspection commands nor their results. This option essentially specifies the legacy behavior of the tools.

- **input_only**

  A keyword that transcribes all input from the dofile (both Tcl and tool commands) before sending to Tcl for evaluation. Since the command interpreter accumulates input until receiving a complete Tcl command before sending to Tcl for evaluation, entire if/else, for, and foreach constructs transcribe as a single item. Embedded tool commands are not transcribed again when they execute.

- -result_collection_limit unlimited | *limit*

  A switch and string or integer pair that specifies number of collection entries show in the interactive mode when a collection is returned. The default is 50.

- -argument_collection_limit unlimited | *limit*

  A switch and string or integer pair that specifies the number of collection entries show in the listing when a collection is passed to a command as an argument. The default is 1.

- -one_collection_entry_per_line on | off

  A switch and literal pair that show the collections on arguments and results with one entry per line. The default is off.

Examples

The following example consists of a dofile and the transcription results of the various style options.

Dofile:

```
set foo 2
set myclk(1) CLK1
set myclk(2) CLK2
set pfile mytest.proc
if { $foo == 2 } {
   add_clocks 0 $myclk(1)
   add_clocks 0 $myclk(2)
}
add_scan_groups g1 $pfile
```

Full-style transcript:

```
//  command: set foo 2
//  command: set myclk(1) CLK1
//  command: set myclk(2) CLK2
//  command: set pfile mytest.proc
//  command: if { $foo == 2 } {
//                     add_clocks 0 $myclk(1)
//                     add_clocks 0 $myclk(2)
//                   }
//  sub-command: add_clocks 0 CLK1
//  sub-command: add_clocks 0 CLK2
//  command: add_scan_groups g1 $pfile
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tool_commands_only-style transcript:

```
//  command: add_clocks 0 CLK1
//  command: add_clocks 0 CLK2
//  command: add_scan_groups g1 mytest.proc
```

Input_only-style transcript:

```
//  command: set foo 2
//  command: set myclk(1) CLK1
//  command: set myclk(2) CLK2
//  command: set pfile mytest.proc
//  command: if { $foo == 2 } {
//                     add_clocks 0 $myclk(1)
//                     add_clocks 0 $myclk(2)
//                   }
//  command: add_scan_groups g1 $pfile
```

Related Topics

get_transcript_style

no_transcript

set_logfile_handling

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_transient_detection

Context: dft -edt, dft -scan, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis (dft -scan context only)

Specifies whether the tool detects all zero width events on the clock lines of state elements.

Usage

set_transient_detection **OFf** | {**ON** [-Verbose | -NOVerbose]}

Description

The set_transient_detection command sets the simulator to detect all zero width events on the clock lines of state elements. If the zero width event causes a change of state in the state element, the tool sets that state element to X.

If Off is specified, DRC simulation treats all events on state elements as valid. Because the simulator is a zero-delay simulator, it is possible for DRC to simulate zero width, monostable circuits with ideal behavior which is rarely matched in silicon. The resulting zero width output pulse from the monostable circuit is also treated as a valid clocking event for other state elements.

Arguments

- **OFf**

  A literal that specifies for the tool to set transient detection off.

  Caution

  This argument is only for special cases where you understand the potential hazards from transients on the clock lines and know how to handle them. Turning transient detection off without taking care of the hazards can result in bad patterns that mismatch when verified in a timing based simulator. Be sure you understand the consequences and are willing to accept the risks before turning transient detection off.

- **ON**

  A literal that specifies for the tool to set transient detection on. This is the invocation default.

- -Verbose

  An optional switch that specifies for the tool to display a message identifying each time a state element is set to X due to transient detection. This is the invocation default and is valid only when transient detection is enabled.

- -NOVerbose

  An optional switch which specifies for the tool not to display a message identifying each time a state element is set to X due to transient detection. This switch is valid only when transient detection is enabled.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Transient detection is enabled in verbose mode by default. Unless you use the set_transient_detection command to modify the default, you see a message similar to the following for each transition that is caused by a transient, or zero-width, pulse:

```
//  Warning: 0 width pulse detected on pin 2 of
/my_design/my_inst/my_latch/ (463228) in procedure my_procedure at time 0.
```

You can eliminate these messages, while retaining the advantages of transient detection, by issuing this command:

set_transient_detection on -noverbose

Or you can disable transient detection, causing DRC simulation to treat all events on state elements as valid.

set_transient_detection off

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_transition_holdpi

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies for the tool to freeze all primary input values other than clocks and RAM controls during multiple cycles of pattern generation.

Usage

set_transition_holdpi **ON | OFf**

Description

Specifies for the tool to freeze all primary input values other than clocks and RAM controls during multiple cycles of pattern generation.

The set_transition_holdpi command lets you turn this feature on while the fault type is in "transition". This is useful in cases where it is not practical to maintain high data rates to the primary input pins of the device under test on a tester. Primary inputs may still change from pattern to pattern however.

Note

Because launch off shift transition patterns require the scan enable signal (which is not a clock or RAM control) to transition within a single pattern, the tools ignore this command when generating this type of pattern.

Arguments

- **ON**

  A literal that specifies for the tool to hold all primary input values (other than clocks and RAM controls) during multiple cycles of pattern generation.

- **OFf**

  A literal that specifies for the tool not to hold all primary input values. This is the invocation default.

Examples

The following example holds all primary input values.

```
set_transition_holdpi on
```

Related Topics

report_primary_inputs

set_pathdelay_holdpi

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_tristate_gating

Context: dft -scan, dft -test_points

Mode: setup

Specifies how tri-state devices are controlled during scan chain shifting.

Usage

set_tristate_gating {**OFf** | **ON** | **Busdrivers** | **Scan** | ***primary_input_or_output...***}
[-Control {SEn | TEn}] [-Force_gating]

Description

Specifies how tri-state devices are controlled during scan chain shifting.

The set_tristate_gating command ensures tri-state devices are controlled to either prevent bus contention or be turned on during testing. If the necessary test logic already exists, none is inserted.

When enabled, test logic is inserted that controls tri-state devices as follows:

- **Multiple tri-state gates driving nets (bus net)** — One gate is turned on and the rest are disabled during testing.
- **Single tri-state gates driving nets (primary net or an internal net)** — All single tri-state gates are turned on for testing.

You can also specify which signal (SEN or TEN) controls the enable lines of tri-state devices.

By default, when the enable signal of a tri-state device is directly controlled by a primary input, by TIE0, or by TIE1, no gating is necessary and a force statement for the primary input is added to the load_unload procedure in the new procedure file. This behavior can be overridden by using the -Force_gating switch.

Arguments

- {**OFf** | **ON**| **Busdrivers**| **Scan**| ***primary_input_or_output...***}

  Required literal or repeatable string that specifies which tri-state devices to control during scan shifting. The default setting is off. Options include:

  OFf — no test logic is inserted to control tri-state devices. Default setting.

  ON — test logic is inserted when necessary to control all tri-state devices.

  Busdrivers — test logic is inserted to control tri-state devices driving bus nets.

  Scan — test logic is inserted to control tri-state devices used as scan inputs/outputs.

  primary_input_or_output — specifies a primary input or output pin. Test logic is inserted to control the tri-state devices driving the specified primary output pin(s) or driven by the specified primary input pin(s).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Control SEn | TEn

  An optional switch and literal pair that specifies the enable signal used to control tri-state devices. Literal options include:

  SEn — Specifies the scan_enable signal. Default setting.

  TEn — Specifies the test_enable signal.

- -Force_gating

  An optional switch that adds test logic to the enable lines of tri-states devices when these lines are directly controlled by primary inputs, or by TIE1, or by TIE0. When the enable line is directly controlled by a primary input, the tool adds the force statement for this primary input to the load_unload procedure in the procedure file.

Examples

Example 1

The following example uses the set_tristate_gating command to insert test logic and make all tri-state devices driving bus nets controllable via the SEN signal.

```
add_clocks 0 clk
set_tristate_gating on
set_bidi_gating scan
set_system_mode analysis
add_scan_mode unwrapped -si_connections c1 bidi_in1/X \
          -so_connections blkB1/blkA/utri2/A
analyze_scan_chains
```

Related Topics

report_test_logic

report_control_signals

set_bidi_gating

set_test_logic

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_tsdb_options

Context: all contexts

Mode: all modes

Modifies options for how a file path located within the design source dictionary file is interpreted.

Usage

set_tsdb_options
[-directory_path_map {*original_path_prefix new_path_prefix ...*} [-append]]

Description

Use this command to define a directory path map list so the read_design command can continue to work even when you relocate the TSDBs. See the example in the root_directory section of the TSDB chapter for a detailed explanation of the use of absolute, relative, and symbolic paths to achieve this.

As described in the above-referenced example, you only need to specify a directory path map when the location of the chip repository is different from one site to the next. You can use a common link in all sites to point to the local chip repository and read the design through the link. The absolute paths stored in the *<design_name>.design_source_dictionary* file maintain the use of soft links and avoid the need to use mapping between sites. See the dft_inserted_designs section for more details on the *<design_name>.design_source_dictionary* file.

Arguments

- -directory_path_map {*original_path_prefix new_path_prefix ...*}

  An optional switch and repeatable value pair to specify pairs of file path prefixes to map the original to the new file location when using the content of the *<design_name>.design_source_dictionary* file (see dft_inserted_designs) with the read_design, write_design_import_script, and run_testbench_simulations commands.

  When using newer TSDBs, the path to the files within the chip repository is absolute, so the directory path map you use is always an absolute path prefix.

  If you have an older TSDB generated with relative paths, you use relative path prefixes for the original names you want to map. It is best and easier to map those original relative path prefixes to absolute paths prefixes.

  If you specify an absolute path prefix for a new location, the tool immediately validates it and reports an error if it does not find it.

  If you specify a relative path for the new location, the tool does not validate it, as those are not supposed to be valid relative to the current working directly. Instead, it is validated against the location of the given *<design_name>.design_source_dictionary* file when you run the read_design command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the new path is invalid relative to the design source dictionary file location, the tool ignores it. It reports a warning even if the original path matched a path in the dictionary.

- -append

  If you specify the optional -append switch, the new pair list you specify is appended to the existing list. If you do not specify the -append switch, the new pair list you specify overwrites the existing list that was previously specified.

Examples

Refer to Figure 15-6 of the root_directory section of the TSDB chapter for an example that makes use of this command in a real design flow. Below are examples showing various scenarios:

- Map the beginning of an absolute path starting with */ny/designs* to */la/designs*:

```
> set_tsdb_options -directory_path_map {/ny/designs /la/designs}
> get_tsdb_option -directory_path_map
/ny/designs /la/designs
```

- To deal with a legacy TSDB written with relative paths, this example maps a relative path *../../../designs* to */la/designs*. The use of the -append switch retains the previously set map and appends to it:

```
> set_tsdb_options -directory_path_map {../../../designs /la/
designs} \
                   -append
> get_tsdb_option -directory_path_map
/ny/designs /la/designs ../../../designs /la/designs
```

Related Topics

get_tsdb_option

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_tsdb_output_directory

Context: unspecified, dft, patterns

Mode: setup

Sets the TSDB output directory used by the process_dft_specification command.

Usage

set_tsdb_output_directory ***directory_path***

Description

Sets the TSDB output directory used by the process_dft_specification command.

The TSDB output directory defaults to *./tsdb_outdir* until the set_tsdb_output_directory command has been invoked. When either the get_tsdb_output_directory or set_tsdb_output_directory command is invoked, the tool verifies that the specified path is a writable directory if it already exists, or that the parent directory is a writable directory if it does not.

You specify a value of *<current_design>_<design_id>.tsdb* when a block undergoes several DFT insertion passes, and you do not intend to use the same TSDB output directory for all of them. This can happen if different individuals complete different insertion passes, possibly in different sites, as is the case in the Example in the root_directory section of the TSDB chapter. If all RTL insertion passes use the same TSDB output directory, *<current_design>_rtl.tsdb* is more appropriate. The example mentioned above explains what the tool does within the dft_inserted_designs directory to ensure that the TSDB can be relocated and still support the read_design command automation.

When you subsequently open the TSDB and run the set_current_design command, the tool automatically reads in the most recent *module_name.format*_interface file using the "read_verilog -interface_only" or "read_vhdl -interface_only" command if the corresponding *module_name* does not already exist in memory. This enables you to only load the modules associated with the current design and have the interface view of all child physical or sub-blocks loaded automatically.

Note

The set_current_design command does not automatically load a child block that has the interface_has_external_dependencies property set to 1 in the *.tsdb_info* file of the child block. The interface_has_external_dependencies property is set to "1" if the port list of the interface has external dependencies such as typedef packages or System Verilog interfaces. If you want to load that child block, use the "read_design *child_block* -view interface" command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***directory_path***

  A required string value used to specify the TSDB output directory which is populated by commands such as process_dft_specification. The specified directory_path can be a relative path with respect to the current working directory or an absolute path. In either case, the directory must be a writable directory when it exists, or the effective parent directory where it is to be created must be a writable directory. The set_tsdb_output_directory command does not create the directory; it is created the first time a command needs to store information into it such as when the process_dft_specification command is invoked.

Examples

Example 1

The following example set the TSDB output directory to "./dir1".

set_tsdb_output_directory dir1

Example 2

The following example shows a possible error than can be generated when setting the TSDB output directory.

set_tsdb_output_directory dir2/dir1

```
//  Error: The parent directory of the specified tsdb_output_directory
//  '/home/user1/dir2' does not exist.
```

Related Topics

get_tsdb_output_directory

process_dft_specification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_visualizer_preferences

Context: unspecified, all contexts

Mode: setup, analysis

Controls the Tessent Visualizer display preferences for the Flat Schematic and Hierarchical Schematic windows.

Usage

set_visualizer_preferences **-color_theme** {**dark** | **high-contrast** | **light**}

Description

Controls the Tessent Visualizer display preferences for the Flat and Hierarchical Schematic windows. Available preferences are “dark,” “light,” and “high-contrast.”

Arguments

- **-color_theme** {**dark** | **high-contrast** | **light**}

  A switch and literal that specify the color theme to use for the schematic tabs.

Examples

The following example changes the color theme for the flat and hierarchical tabs to “light.”

```
set_visualizer_preferences -color_theme light
```

Related Topics

analyze_drc_violation

open_visualizer

report_gates

set_gate_report

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_wrapper_analysis_options

Context: dft -scan

Mode: setup, analysis

This command sets up parameters for the wrapper cells analysis.

Usage

set_wrapper_analysis_options
[-output_fanin_flop_threshold {32 | *integer* | unlimited}]
[-input_fanout_flop_threshold {32 | *integer* | unlimited}]
[-output_fanin_libcell_levels_threshold {32 | *integer* | unlimited}]
[-input_fanout_libcell_levels_threshold {32 | *integer* | unlimited}]
[-non_scan_depth_threshold {2 | *integer* | unlimited}]
[-register_ports_exceeding_non_scan_depth {on | off }]
[-register_undriven_output {on | off}]
[-allow_internal_segments_as_wrapper {on | off]}
[-capture_window_behavior { auto | shift | invert_hold | programmable_hold }]
[-dedicated_wrapper_cells_location {inside_power_isolation | outside_power_isolation}]
[-exclude_ports *port_spec*]
[-clear_excluded_ports]
[-register_ports_reaching_blackboxes {*instance_or_module_spec* | all | off}]
[-identify_existing_dedicated_wrappers {off | on}]

Description

This command sets up parameters for the wrapper cells analysis.

Arguments

- -output_fanin_flop_threshold {32 | *integer* | unlimited}

  An optional switch and integer or literal pair that specifies the maximum number of sequential elements reachable during backward tracing from a PO. The default is 32.

  For the scan segments encountered during the backward tracing from POs, there is a special cost function to average the number of the backward hit flops between the total number of POs reaching this scan segment and the length of this scan segment.

- -input_fanout_flop_threshold {32 |*integer* | unlimited}

  An optional switch and integer or literal pair that specifies the maximum number of sequential elements reachable during forward tracing from a PI. The default is 32.

  For the scan segments encountered during the forward tracing from PIs, there is a special cost function to average the number of the forward hit and the backward flops (feedback flops) between the total number of PIs reaching this scan segment and the length of this scan segment.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -output_fanin_libcell_levels_threshold {32 | *integer* | unlimited}

  An optional switch and integer or literal pair that specifies the maximum number of levels of library cells to be traversed during the backward tracing from a PO until the first sequential element is reached. Buffers and inverters are not counted towards the number of levels. The limit set by this switch is for the maximum levels of combinational logic (without taking into account either buffers or inverters). The default is 32.

- -input_fanout_libcell_levels_threshold {32 | *integer* | unlimited}

  An optional switch and integer or literal pair that specifies the maximum number of levels of library cells to be traversed during the forward tracing from a PI until the first sequential element is reached. Buffers and inverters are not counted towards the number of levels. The limit set by this switch is for the maximum levels of combinational logic (without taking into account either buffers or inverters). The default is 32.

- -non_scan_depth_threshold {2 | *integer* | unlimited}

  An optional switch and integer or literal pair that controls the depth of non_scan sequential elements traced by the tool. When the threshold is exceeded, the behavior depends on the -register_ports_exceeding_non_scan_depth switch. The default is 2.

  Note

  The thresholds specified by the -output_fanin_flop_threshold, -input_fanout_flop_threshold, -output_fanin_libcell_levels_threshold, and -input_fanout_libcell_levels_threshold switches will not apply to ports that have been excluded from registration with the "set_dedicated_wrapper_cell_options off" command.

- -register_ports_exceeding_non_scan_depth {on | off }

  An optional switch and literal pair that controls the behavior of the tool when the non-scan depth threshold is exceeded. If it is set to on, a dedicated wrapper cell is added to the port that is being traced. If it is off, the branch that exceeds the threshold is ignored and the tracing continues to search for shared wrapper cell candidates.

- -register_undriven_output {on | off}

  An optional switch and literal pair that specifies whether an output port having no functional source should receive a dedicated wrapper cell.

  - on — undriven outputs receive dedicated wrapper cells except for specific ports that are configured with "set_dedicated_wrapper_cell_options off -ports {…}".
  - off — undriven output ports do not receive any wrapper cells except for specific ports that are configured with "set_dedicated_wrapper_cell_options on -ports {…}". This is the default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -allow_internal_segments_as_wrapper {on | off}

  An optional switch and literal pair that specifies whether to identify the defined scan segments as Input or Output wrapper elements during the forward and backward tracing from the Primary IOs.

  - on — identify encountered scan segments as Input or Output wrapper elements. This is the default.
  - off — do not identify encountered scan segments as wrapper elements. Ports that reach scan segments have a dedicated wrapper cell inferred unless these ports are specified to not infer dedicated wrapper cells for them (see set_dedicated_wrapper_cell_options)

- -capture_window_behavior { auto | shift | invert_hold | programmable_hold }

  Note

  The tool overrides the settings of this switch for dedicated wrapper cells when you use the set_dedicated_wrapper_cell_options command to specify an -input_module_name or -output_module_name. The tool uses that module specification to implement the dedicated wrapper cell and it affects the capture window behavior. The structure of that module defines the capture window behavior.

  An optional switch and literal pair that enables you to specify the hardware configuration to implement the capture window behavior of wrapper cells. With “shift”, wrapper cells (except for the dedicated ones used for set/reset ports) are shifting during the capture window of the internal (or external) test mode. In contrast, with “invert_hold | programmable_hold”, wrapper cells are holding during the capture window of the internal (or external) test mode. This makes it easier to close timing, because the SO-SI connections are not at-speed paths. No holding loop is ever added to shared wrapper existing segments, because it could introduce routing and timing issues.

  - auto — This is the default. It selects “shift” only when you have specified a multiple scan_en scheme (that is, when you specify “set_scan_enable -single_global_scan_enable off”); otherwise, it selects “invert_hold”.
  - shift — This option specifies that the wrapper cells are shifting during the capture window of the internal (or external) test mode.
  - invert_hold — This option specifies that the wrapper cells are holding, with *inverting* feedback, during the capture window of the internal (or external) test mode.
  - programmable_hold — This option specifies that the wrapper cells are holding during the capture window of the internal (or external) test mode. The type of feedback (direct or inverting) is controlled by the static DFT signal “wrapper_toggle_en”, giving ATPG the ability to request that they hold or toggle.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

If the design has test points, some control points may become shared output wrapper cells. Those cells will always hold their value regardless of the capture window behavior setting.

- -dedicated_wrapper_cells_location {inside_power_isolation | outside_power_isolation}

  An optional switch and literal pair that specifies where to insert the Input and Output dedicated wrapper cells with relation to the encountered power isolation cells.

  - inside_power_isolation — insert Input dedicated wrapper cells after the encountered power isolation cells and the Output dedicated wrapper cells before the encountered power isolation cells. This is the default.
  - outside_power_isolation — insert Input wrapper cells immediately after the PI ports being registered and insert the dedicated Output wrapper cells immediately before the PO ports being registered.

- -exclude_ports *port_spec*

  An optional switch and string that specifies that all ports in port_spec are to be excluded from wrapper analysis without regard to their previous state (off/on/auto). Ports can only be removed from exclusion with -clear_excluded_ports switch.

  port_spec can be a Tcl list of one or more ports or a collection of one or more port objects.

  If one or more ports in the port spec has already been set with "set_dedicated_wrapper_cell_options on|off -ports {…}" a note alerts you that this setting has been overridden.

```
// Note: Excluding and overriding set_dedicated_wrapper_cell
// configurations for the following port(s):
// <port1>, <port2>, <port3> …
```

Note

Excluded ports are ignored during wrapper analysis, but any previously identified X-bounding hardware are added when the insert_test_logic command is executed. However, for all other ports, the wrapper analysis results effectively overrides the X-bounding results.

- -clear_excluded_ports

  An optional switch that clears the list of ports that have been excluded from wrapper analysis. These ports are now included in wrapper analysis and receive shared, dedicated, or no wrapper cells based on future and global "set_dedicated_wrapper_cell_options {on|off|auto}" configurations.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -register_ports_reaching_blackboxes {*instance_or_module_spec* | all | off}

  An optional switch and a list of instance or module objects or a literal that directs the tool to add a dedicated wrapper cell to ports connected to the specified instances or instances of specified modules. The argument choices are as follows:

  - *instance_or_module_spec* — A Tcl list or a collection of one or more modules and/or instances. All objects must either be blackbox instances or modules, or interface-only modules. When wrapper analysis tracing from a port reaches one of the specified instances or an instance of one of the specified modules, the tool will add a dedicated wrapper cell to the port.
  - all — Adds dedicated wrapper cells to ports connected to any blackbox instance.
  - off — Disables this feature. This is the default.

  Note

  This command switch is cumulative, and every time you use it, it adds to the list of blackbox modules and instances. Using the off literal clears the list.

- -identify_existing_dedicated_wrappers {off | on}

  An optional switch and literal pair that enables identification of pre-existing dedicated wrapper cells as part of the analysis performed by the command “analyze_wrapper_cells”. The default is off.

Examples

Example 1

In the following example, all Primary IOs except for excluded_1 receive dedicated wrapper cells, including undriven IOs.

```
set_wrapper_analysis_options -exclude_ports excluded_1
set_dedicated_wrapper_cells on
set_wrapper_analysis_options -register_undriven_output on
```

Example 2

In the following example:

Ports A, B, C, D, excluded_1 are driven.

Ports undriven_1, undriven_2 are undriven.

Ports C, D, undriven_1 receive dedicated wrapper cells

Ports A, B, undriven_2, excluded_1 do not receive dedicated wrapper cells.

```
set_wrapper_analysis_options -exclude_ports excluded_1
set_wrapper_analysis_options -register_undriven_output on
set_dedicated_wrapper_cells off -ports {A B undriven_2}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

set_dedicated_wrapper_cell_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_write_patterns_options

Context: patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: analysis, setup

Creates the subset of ports or vector callbacks to use with the write_patterns command.

Usage

Common Usage

set_write_patterns_options [-additional_port_list *additional_port_list*]
[-existing_used_ports *existing_used_ports*] [-reset_used_ports]
[-continuous_expect_during_payload {{*port_name* H | L} …}]
[-delete_continuous_expect_during_payload]
[-delete_selective_tck_stretching_timeplates_after_write {on | off}]

Usage for Vector Callbacks

set_write_patterns_options [-vector_callback *vector_callback*
[-trigger {before_first | after_last | every_cycle | test_setup | test_end}]
[-received_ports *received_ports*]
[-returned_ports *returned_ports*]
[-replace]]
[-delete_vector_callbacks]

Description

Creates the subset of ports or vector callbacks to use with the write_patterns commands. See "Vector Creation and Modification" in the *Tessent Scan and ATPG User's Manual* for additional information.

Common Usage

This command is used to create the subset of ports to use in the subsequent write_patterns commands.

The *additional_port_list* argument value is a Tcl list of object names or a collection of objects. The valid object types are ports, pseudo-port, pin, or gate_pin. A pin must map to a pseudo-port, and a gate pin must map to a pseudo-port or port. The action is to add port names to the used subset, assuming that it is starting from an empty subset. Multiple instances of this command are not cumulative. Each issuance of the command first removes any existing subset, and starts over from the full port list. Issuing the -existing_used_ports option with an empty *obj_spec* such as {} creates an empty subset.

To add ports to the port list that do not exist in the original design data, use the -additional_port_list switch to specify a list of lists of strings which specify the name to add and its port direction. Port directions can be one of input, output, or inout. Optionally, the clock attribute and an optional off state for a clock can also be specified. If no off state is specified, an off state of 0 is assumed. If "clock" is not specified, the additional port is not treated as a clock

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

and the P state value cannot be assigned to it. If “clock” is specified, the additional port is treated as a clock, the P state value can be used, and the timing for the port is taken from the “pulse_clock” statement in the timeplate instead of “force_pi” statement. Issuing the command with only the -additional_port_list switch adds ports only to the default port list supplied by the design and setup information. If any additional ports are added that match the name of an existing port, an error is issued.

This example instructs the write_patterns commands to only include the ports starting with ijtag_ and two new ports called myNewInput and myNewOutput[1:0].

```
set_write_pattern_options -existing_used_ports [get_ports ijtag_*] \
   -additional_port_list [list myNewInput input myNewOutput[1:0] output]
```

Vector Callback Usage

This command adds a vector callback proc name to the list of vector callback procs that are used in the subsequent write_patterns commands. The switch -vector_callback must specify the name of a defined Tcl proc that is passed a vector set and returns a vector set. An optional trigger can be specified as an activation criterion for this Tcl proc. If no trigger is specified, the every_cycle trigger is the default and this callback is used for all vectors. Only one callback can be specified with each trigger.

If another callback is specified with the same trigger as a previously specified callback, this new callback replaces the older one if -replace is used otherwise an error is issued. If a new callback is added with the same name as a previous one, it replaces the previous one if -replace is used, otherwise an error is issued.

Note

If the vector callbacks assume a fixed sequence of TAP states, Siemens EDA recommends disallowing the IJTAG retargeting engine to skip the run-test/idle TAP state between consecutive scan loads. Before opening a pattern set, use the “set_ijtag_retargeting_options -allow_skip_rti off” command to prevent skipping the run-test/idle TAP state.

When using vector_callback procs and writing patterns, the data fed to the vector callback procs must be the internal view pattern data, and therefore, any external_mode cycles from a named capture, or external_mode cycles from PLL_cycles or external_capture are not used. When you enable vector callbacks, the tool issues the following warnings:

```
// Warning:  The set_write_pattern_options -vector_callback command causes
// write_patterns to use the internal_mode view of the pattern data.
// Set_external_capture_options -PLL_cycles is ignored in that case.
```

Or:

```
// Warning:  The set_write_pattern_options -vector_callback command causes
// write_patterns to use the internal_mode view of the pattern data.
// Set_external_capture_options -capture_proc is ignored in that case.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- -additional_port_list *additional_port_list*

  An optional switch and value pair that specifies a list of lists of strings which specify the name to add and its port direction using the following syntax:

  -additional_port_list {{**port_name input** | **output** | **inout** [clock [0|1]]} …}

- -existing_used_ports *existing_used_ports*

  An optional switch and value pair that specifies an absolute list of the subset of existing design ports to use when writing patterns. Refer to "Controlling Ports Used in Pattern and Testbench Generation" on page 1457 in the process_patterns_specification command description for additional information and usage notes.

  Note

  Removing active ports on a design may cause miscompares when you read the patterns back into the tool and re-simulate. If you remove active ports that the pattern data uses, the write_patterns command issues a warning to that effect.

- -reset_used_ports

  An optional switch used to reset the port list back to the original port list supplied by the design and setup information contained in the tool.

- -continuous_expect_during_payload {{*port_name* H | L} …}

  An optional switch and Tcl list of string and literal pairs that specifies a port name and an expect value to the set of continuous expect ports. The tool adds the expect values to the pattern payload data, and every test cycle after the setup procs complete (test_setup and ssn_setup) contains a measure for that value on the port until the start of the end procs (test_end and ssn_end). The possible expect values are H and L. You must specify the pairs as Tcl lists, and you must specify the list of pairs as a Tcl list; this means that both pairs of braces are required, even for a single item. For example:
  "-continuous_expect_during_payload {{myOut H} {locked L}}" or
  "-continuous_expect_during_payload {{locked L}}".

  You cannot specify this for any of the following:

  - an input port
  - a port that has an input constraint or tied value
  - a clock port
  - a port that is also a scan pin or scan control signal
  - a port that is forced or measured in test procedures

  Each time you use this option, it replaces the expect ports with the ones specified. If you specify the same port more than once or if you specify an invalid port, the tool reports an error.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -delete_continuous_expect_during_payload

  An optional switch that removes all expect ports and values specified with the -continuous_expect_during_payload argument.

- -delete_selective_tck_stretching_timeplates_after_write {on | off}

  An optional switch and value pair that, by default, removes all generated timeplates for selective TCK stretching when the write_patterns command completes. If you use vector callbacks and "write_patterns -vector_set" with selective TCK stretching, you must specify off for this switch and retain the timeplates, otherwise the tool reports an error.

- -vector_callback *vector_callback*

  An optional switch and value pair that adds a vector callback proc name to the list of vector callback procs that are used in subsequent write_patterns commands.

- -trigger {before_first | after_last | every_cycle | test_setup | test_end}

  An optional switch and literal used with the -vector_callback switch that specifies the activation criterion for the vector callback procs. Only one callback can be specified with each trigger. Choose from the following:

  before_first — A literal that specifies running the callback proc before the first vector in the pattern_set.

  after_last — A literal that specifies running the callback proc after the last vector in the pattern_set.

  every_cycle — A literal that specifies running the callback proc for every cycle in the pattern_set.

  test_setup — A literal that specifies running the callback proc for test_setup vectors.

  test_end — A literal that specifies running the callback proc for test_end vectors

  If no trigger is specified, the every_cycle trigger is the default and this callback is used for all vectors.

  A callback added with the activation criterion of before_first or after_last is not passed a vector set when it is called. These can be used to add additional vectors and/or annotations to the vector set, occurring before the first vector or after the last vector. When adding callback procs using these two triggers, it is not possible to specify the -received_ports list as no vector set is passed to these callbacks.

- -received_ports *received_ports*

  An optional switch and Tcl list of port names or gate_id or a collection of ports or gate that is used as the callback port list when passing a vector set to the callback. You use this switch in conjunction with the -vector_callback switch.

- -returned_ports *returned_ports*

  An optional switch and Tcl list of port names or gate_id or a collection of ports or gate that is used as the callback port list when returning a vector set from the callback. You use this switch in conjunction with the -vector_callback switch.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -replace

  An optional switch that replaces a previously specified callback with a new callback. You use this switch in conjunction with the -vector_callback switch.

- -delete_vector_callbacks

  An optional switch that removes all existing vector callbacks.

Related Topics

get_write_patterns_options

report_write_patterns_options

write_patterns

Vector Creation and Modification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_xbounding_options

Context: dft -scan, dft -test_points

Mode: setup, analysis

Sets up the parameters for X bounding insertion. The analysis is triggered by the analyze_xbounding command.

Usage

set_xbounding_options [-xbounding_enable *xbounding_enable_name*]
[-connect_to {existing_scan_cell | new_scan_cell | constant_value}]
[-scan_drive_limit {unlimited | *scan_drive_limit*}]
[-mcp_bounding_enable *mcp_enable_name*]
[-exclude_sdc_cross_domain_paths {off | on}]
[-test_clock {off | on}] [-exclude *pin_or_port*]
[-use_scan_cells {off | on }] [-bound_clock_gater_enables { on | off }]

Description

Sets up the parameters for X bounding insertion. The analysis is triggered by the analyze_xbounding command.

You use this command to set various options that control how the tool performs X-bounding analysis. These options include:

- The name of the enable signals that activate the bounding logic.
- The type of bounding (new scan cells, existing scan cells, or constant values) that is used to block the X source.
- Determine bounding of cross-clock domain paths as specified by the SDC file.

The actual analysis must be triggered by issuing the analyze_xbounding command.

If you issue this command without any arguments, all the defined and default settings are reported.

Arguments

- -xbounding_enable xbounding_enable_name

  An optional switch and literal pair used to specify the name of the of the X-bounding enable signal.

- -connect_to {existing_scan_cell | new_scan_cell | constant_value}

  An optional switch and literal pair used to specify whether the X sources are controlled by existing or new scan cells or whether the X source is blocked with a constant value.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Choose from one of the following:

existing_scan_cell — Specifies using existing scan cells to control the X sources. This is the default.

new_scan_cell — Specifies using new scan cells to control the X sources. When using new scan cells for X bounding, the scan cell model specified either by the add_cell_models command or the cell_type Tessent Cell library attribute is used.

constant_value — Specifies using either an AND gate to block the X source with a constant 0 or an OR gate to block the X source with a constant 1. The specific bounding value (0 or 1) is determined by performing a simple analysis of the logic that is forced to the constant value.

- -scan_drive_limit {unlimited | *scan_drive_limit*}

  An optional switch and literal pair or switch and integer pair that specifies how many Mux-type control points one scan cell can drive for bounding. The number of control points can be unlimited or an integer between 1 and 100. Having one scan cell drive a large number of Mux control points can reduce test coverage, due to structural dependency. The invocation default is 1.

  Note

  This switch is ignored when used with -connect_to new_scan_cell.

- -mcp_bounding_enable *mcp_enable_name*

  An optional switch and string pair that specifies the name of an alternate enable signal for bounding at-speed paths. This provides independent control of these bounding muxes as the tool must bound false paths and multicycle paths.

- -exclude_sdc_cross_domain_paths {off | on}

  An optional switch and literal that specifies whether the tool bounds cross clock-domain paths that are specified in the SDC file.

  A cross clock-domain path occurs when memory elements driven by one clock domain, feed logic that is observed (captured) by memory elements that are operated by a different clock.

  Cross clock-domain paths are typically specified as false paths in the SDC file. In that case, these paths are identified as potential X sources and bounding muxes are inserted near the capture point.

  In some cases, this might constrain the clock sequences that are applied during Logic BIST such that these paths are never sensitized. However, if your hardware is designed such that certain cross clock_domain interactions are not tested during Logic BIST, and you choose to not test these the cross clock_domain paths by constraining the capture sequences, then there is no need to insert bounding muxes.

  The -exclude_sdc_cross_domain_paths switch instructs the tool to ignore false paths that are specified in the SDC file during analysis.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

In order to guarantee that all cross clock domains are targeted for X-bounding, you must specify all cross clock domain paths as false paths in the SDC file. This can be done indirectly by using the set_clock_groups command to assign each clock to a separate group.

- -test_clock {off | on}

  An optional switch and literal pair that determines the clock signal for the newly inserted scan cells that drives the test_mode input of the X-bounding muxes. This option must be used in conjunction with the -connect_to new_scan_cell option.

  off — A literal that specifies that the clock signal for the newly created scan cells is chosen to match the clock domain of the flops in the fanout cone of the X-bounding mux.

  on — A literal that specifies that the newly created scan cells is driven by the test clock signal as defined with the “set_scan_signals -tclk” command.

- -exclude *pin_or_port*

  An optional switch and string pair that specifies the pins/ports that are guaranteed to have a known value during fault simulation and never propagate an X value to the MISR.

  The value of *pin_or_port* can be a Tcl list of one or more pins/ports, or a collection of one or more pin/port objects.

  You can use the -exclude switch multiple times, but the tool only recognizes the last usage.

  Note

  The tool does not insert X-bounding muxes at the specified pins/ports, or at the logic that is driven by these pins. During fault simulation the X-sources are not ignored, which means you must manually apply the appropriate initialization and constraints in order to avoid X's reaching the MISR during fault simulation.

- -use_scan_cells {off | on }

  An optional switch and literal pair that controls the tool behavior for the following specific scenario:

  - X-bounding is done on a pre-scan netlist.
  - the "-connect_to new_scan_cells" option was used.
  - you do not issue the "analyze_scan_chains" command after analyze_xbounding.

  When this switch is off (this is the default) the tool inserts non-scan flops to drive the test_mode input of the X-bounding muxes.

  When the switch is on, the tool inserts scan_cells. However, since analyze_scan_chains was not called, these new scan flops are not stitched into chains.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -bound_clock_gater_enables { on | off }

  An optional switch and literal pair that controls X-bounding of false or multi-cycle paths that reach the functional enable input of clock gaters that drive scannable cells.

  When set to on (this is the default), the tool adds X-bounding logic at the test_enable input of the clock gater to force it to always be enabled such that the potential X at the functional enable does not have any impact.

  When set to off, the tool does not perform any bounding of such paths.

Examples

Example 1

In the following example, when you issue this command without any arguments, it reports the current settings:

set_xbounding_options

```
Xbounding enable name           : lbist_en
Connect to                      : existing scan cell
Scan drive limit                : 1
Exclude sdc cross domain paths  : OFF
MCP bounding enable name        : lbist_en
Use test clock                  : OFF
Exclude pin or port list : DataOut_i[0]
                         : DataOut_i[1]
                         : DataOut_i[2]
```

Example 2

The following command specifies non-default enable names for the primary enable signal as well as the false and multicycle path enable prior to performing the X-bounding analysis:

**set_xbounding_options -xbounding_enable lbist_en **
-mcp_bounding_enable mcp_path_enable

analyze_xbounding

Related Topics

analyze_xbounding

insert_test_logic

report_xbounding

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_xclock_handling

Context: all contexts

Mode: setup, analysis

Specifies whether the sequential element model outputs X when any of its clock inputs become X.

Usage

set_xclock_handling {**Retain** | **X**} [-Pessimistic_simulation {ON | OFf}]

Description

Specifies whether the sequential element model outputs X when any of its clock inputs become X.

By default, the flip-flop and latch default models are used for sequential elements. These default models retain their output values when their clock value becomes X so long as the other input values do not cause the stored output value to change, regardless of whether the clock was at a 0 or 1. The set_xclock_handling command lets you override this behavior during pattern simulation, causing the stored output value to become X.

Table 8-23 lists the Q capture value of the level-sensitive sequential element models based on the X values on input pins, the Q value, and on the set_xclock_handling arguments. Table 8-24 lists the outputs of edge-triggered sequential element models based on the clock transition, the D value, and on the set_xclock_handling arguments.

This command has no effect during DRC simulation. The default flip-flop and latch behavior is used during DRC simulation.

Table 8-23. X Handling for Level-Sensitive Ports

Level-Sensitive Sequential Element Ports		Q capture value when using the Retain argument	Q capture value when using the X argument
latch enable = X	Q=D	Q	X
	Q≠D	X	X
set = X	Q=0	X	X
	Q=1	1	X
reset = X	Q=0	0	X
	Q=1	X	X

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 8-24. X Handling for Edge-Triggered Ports

Edge-triggered Sequential Element Inputs		Output when using "Retain -pessimistic_simulation OFF / ON"		Output when using "X -pessimistic_simulation OFF / ON"	
Clock event	**D value before clock event**	**OFF**	**ON**	**OFF**	**ON**
X→X X→1 0→X	D=Q	Q	Q	X	X
	D≠Q	X	X	X	X
1→X	D=Q	Q	Q	Q	X
	D≠Q	Q	X	Q	X
X→0	D=Q	Q	Q	Q	Q
	D≠Q	Q	Q	Q	Q

Arguments

- **Retain** | **X**

  A required literal that uses the default sequential element model (by default) or uses a modified sequential element model. For information about the effect of using these arguments, refer to Table 8-23 and Table 8-24.

- -Pessimistic_simulation {ON | OFf}

  An optional switch that affects the output of the edge-triggered sequential element model when the clock port becomes X, as shown in Table 8-24.

Examples

The following example sets the outputs of edge-triggered sequential elements to X unless any of their clock inputs transition from X to 0 during pattern simulation:

```
set_xclock_handling x -pessimistic_simulation on
```

Related Topics

get_xclock_handling

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_z_handling

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup

Specifies the simulation handling for high impedance signals on internal and external tri-state nets.

Usage

set_z_handling {**Internal** ***state***} | {**External** ***state***}

Description

Specifies the simulation handling for high impedance signals on internal and external tri-state nets.

The set_z_handling command specifies how to handle the high impedance state for internal and external tri-state nets during pattern simulation. If the high impedance value can be made to behave as a binary value, certain faults may become detectable. If you do not use this command to set the Z handling, the default value is X.

This command does not impact expected values on tri-state and bidirectional pins in the final saved patterns. The command changes the behavior of how Z values are simulated on internal nets. By default, Z values are treated as X values when fed into another internal gate, such as an AND gate. However, a Z value on a tri-state output pin is not converted to an X by default.

Arguments

- **Internal** ***state***

  A literal pair that specifies how the tool simulates high impedance values for internal tri-state nets. The literal choices for the ***state*** option are as follows:

  X — A literal that specifies to treat high impedance states as an unknown state. This is the default behavior upon invocation of the tool.

  0 — A literal that specifies to treat high impedance states as a 0 state.

  1 — A literal that specifies to treat high impedance states as a 1 state.

- **External** ***state***

  A literal pair that specifies how the tool simulates high impedance values for external tri-state nets. The literal choices for the ***state*** option are as follows:

  X — A literal that specifies not to measure high impedance states; they cannot be distinguished from a 0 or 1 state. This is the default behavior upon invocation of the tool.

  0 — A literal that specifies to treat high impedance states as a 0 state that can be distinguished from a 1 state.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

1 — A literal that specifies to treat high impedance states as a 1 state that can be distinguished from a 0 state.

Z — A literal that specifies to uniquely measure the high impedance states; they can be distinguished from both a 0 or 1 state.

Examples

The following example specifies to treat high impedance values as 1 states when they feed into logic gates, and as 0 states at the output of the circuit, then performs a pattern simulation run:

```
set_z_handling internal 1
set_z_handling external 0
set_system_mode analysis
create_patterns
```

Related Topics

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

set_zhold_behavior

Context: dft -edt, patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Specifies whether ZHOLD gates retain their state values.

Usage

set_zhold_behavior **OFf** | **ON**

Description

The ZHOLD gate enables the tool to model tri-state nets so that they can retain the previous state value when the net goes to a high impedance (Z) value. ZHOLD gates normally require clock sequential patterns to utilize this capability. But, if a ZHOLD gate is set to a fixed binary value when the clocks are off, the ZHOLD gate can retain that value for combinational patterns. The combinational fault simulation does not consider the fault effect on the retained value.

By invocation default, the tool assumes the retained value is always the clock off_state value for both the good machine and the faulty machine simulation.

The tool cannot assume a state from a previous scan pattern or from a load operation. The amount of time the ZHOLD gate can hold its value is limited to the number of sequential clock cycles.

Arguments

- **OFf**

  A literal that specifies not to let ZHOLD gates retain values.

- **ON**

  A literal that specifies for the tool to use ZHOLD gates to retain values subject to restrictions that the rules checker identified. This is the default.

Examples

The following example specifies that the ZHOLD gates cannot retain their previous values:

```
set_zhold_behavior off
```

Related Topics

report_gates

set_learn_report

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

setenv

Context: unspecified, all contexts

Mode: all modes

Sets a shell environment variable within the tool environment.

Usage

setenv ***variable_name*** [*value*]

Description

Sets a shell environment variable within the tool environment.

The setenv command sets or changes the value of a shell environment variable within the current tool environment, enabling the tool to access the variable for the rest of the session just as if the variable had been set from the shell. This command is useful if you ever find that the tool requires an environment variable that was not set prior to invocation.

Note

A shell variable you set with this command is set only with respect to the current tool session. The command does not alter the value of the variable in the invocation shell.

To unset the variable before you exit, use the unsetenv command. To view the value of the variable, use the command to pass an appropriate shell command (echo or env for example) to the operating system for execution.

Arguments

- ***variable_name***

  A required string that specifies the name of the variable.

- *value*

  An optional string that specifies the value of the variable.

Examples

The following example assigns a value to the environment variable SSH_AGENT_PID, then displays its value:

```
setenv SSH_AGENT_PID 66216
system {echo $SSH_AGENT_PID}
```

```
66216
```

Related Topics

system

unsetenv

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

shmoo_cdp_test

Context: patterns -silicon_insight

Mode: all modes

Generates a two-dimensional shmoo plot for a characterization and debug package (CDP) test.

Usage

shmoo_cdp_test ***test_name*** **-ate_callbacks_file** ***ate_callbacks_file***
-x_ate_callback ***x_ate_callback*** **-x_label** ***x_label*** **-start_x_value** ***start_x_value***
-stop_x_value ***stop_x_value*** **-x_resolution** ***resolution_value***
-y_ate_callback ***y_ate_callback*** **-y_label** ***y_label*** **-start_y_value** ***start_y_value***
-stop_y_value ***stop_y_value*** **-y_resolution** ***resolution_value***
[-report_result] [-pass_symbol *pass_symbol*] [-fail_symbol *fail_symbol*]
[-x_header_placement top | bottom]

Description

The shmoo_cdp_test command uses a CDP test and an ATE callbacks file to generate a text-based, non-graphical shmoo plot of two characteristics. The callbacks file includes procs to test for pass and fail given the x and y values, similar to the following:

```
proc x_ate_callback { x_value } {
  # Set <x_value> by controlling GPIB instrument (SID) or ATE resource (ATE-Connect)
}
proc y_ate_callback { y_value } {
  # Set <y_value> by controlling GPIB instrument (SID) or ATE resource (ATE-Connect)
}
```

Arguments

- ***test_name***

  A required string that specifies the name of the CDP test to generate a shmoo plot for.

- **-ate_callbacks_file** ***ate_callbacks_file***

  A required switch and string pair that specifies the filename containing the user-provided ATE callbacks in Tcl.

- **-x_ate_callback** ***x_ate_callback***

  A required switch and string pair that specifies the ATE callback proc name to call to set the parameter for the x-axis.

- **-x_label** ***x_label***

  A required switch and string pair that specifies the label for the x-axis.

- **-start_x_value** ***start_x_value***

  A required switch and real number pair that specifies the start value for the x-axis.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- **-stop_x_value *stop_x_value***

  A required switch and real number pair that specifies the stop value for the x-axis.

- **-x_resolution *resolution_value***

  A required switch and real number pair that specifies the step between intervals displayed for the x-axis.

- **-y_ate_callback *y_ate_callback***

  A required switch and string pair that specifies the ATE callback proc name to call to set the parameter for the y-axis.

- **-y_label *y_label***

  A required switch and string pair that specifies the label for the y-axis.

- **-start_y_value *start_y_value***

  A required switch and real number pair that specifies the start value for the y-axis.

- **-stop_y_value *stop_y_value***

  A required switch and real number pair that specifies the stop value for the y-axis.

- **-y_resolution *resolution_value***

  A required switch and real number pair that specifies the step between intervals displayed for the y-axis.

- -report_result

  An optional switch that prints the shmoo result table. By default, the tool does not display the result table.

- -pass_symbol *pass_symbol*

  An optional switch and single character to represent a “pass” result in the shmoo plot. By default, a space ( ) represents a “pass” result.

- -fail_symbol *fail_symbol*

  An optional switch and single character to represent a “fail” result in the shmoo plot. By default, an asterisk (*) represents a “fail” result.

- -x_header_placement top | <u>bottom</u>

  An optional switch and literal pair that sets the placement of the x-axis information in the shmoo plot. By default, the x-axis appears at the bottom of the plot.

Examples

The following example shows a shmoo plot of frequency versus voltage:

```
shmoo_cdp_test glue_logic -ate_callbacks_file ../my_shmoo_procs.tcl \
        -x_ate_callback gd_set_clock -x_label freq(MHZ) -start_x_value 100.0 \
        -stop_x_value 900.0 -x_resolution 10.0 -y_ate_callback gd_set_power \
        -y_label Vcc(Volts) -start_y_value 1.0 -stop_y_value 2.5 -y_resolution 0.05 \
        -report_result
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Shmoo of 'freq(MHZ)' vs. 'Vcc(Volts)': (pass symbol=' ', fail symbol='*'):
//
//   Vcc(Volts)
//      3.38| ******
//      3.26| *******
//      3.14| ********
//      3.02| ********
//      2.90| *********
//      2.78| **********
//      2.66| ***********
//      2.54| ************
//      2.42| *************
//      2.30| **************
//      2.18| ***************
//      2.06| ******************
//      1.94| *******************
//      1.82| ********************
//      1.70| ************************
//      1.58| *************************
//      1.46| ****************************
//      1.34| *******************************
//      1.22| ************************************
//      1.10| ****************************************
//      0.98| ************************************************
//      0.86| *******************************************************
//      0.74| ****************************************************************
//      0.62| *********************************************************************************
//      0.50| **********************************************************************************
//
//            ================================================================================
//
//            1111111111222222222233333333334444444444555555555566666666667777777777888888888889
//            0123456789012345678901234567890123456789012345678901234567890123456789012345678901
//            0000000000000000000000000000000000000000000000000000000000000000000000000000000000
//            ................................................................................
//            0000000000000000000000000000000000000000000000000000000000000000000000000000000000
//                                                     freq(MHZ)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The callback file for this example would look similar to the following:

```
proc gd_set_clock { frq } {
  # Set clock frequency to '${frq}MHZ with execute_gpib_command
            .
            .
  execute_gpib_command "FREQ ${frq}" -address 15
            .
            .
  after 400
}
proc gd_set_power { v } {
  # Set Vcc power to '${v}V' with execute_gpib_command
            .
            .
  execute_gpib_command "SOURCE:VOLTAGE:LEVEL:IMMEDIATE:AMPLITUDE ${v}" -address 19
            .
            .
  after 500
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

shutdown_sid_tester

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Terminates the currently active SID tester process that was created by executing the launch_sid_tester command.

Usage

shutdown_sid_tester

Description

Terminates the currently active SID tester process that was created by executing the launch_sid_tester command.

Arguments

None

Examples

The following example shuts down the SID tester process.

```
shutdown_sid_tester
```

Related Topics

launch_sid_tester

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

simulate_clock_pulses

Context: all contexts

Mode: setup, analysis

Prerequisites: The flat model must already exist, and you must have previously used the add_clocks command to define specified gate_pin objects as clocks.

Pulses one or more clocks within the current simulation context.

Usage

simulate_clock_pulses ***obj_spec*** [-repetitions *number*]

Description

Pulses one or more clocks within the current simulation context.

The simulate_clock_pulses command pulses the specified clock(s) within the current simulation context. Before issuing this command, you must previously use the add_clocks command to define the specified gate_pin objects as clocks.

The default number of clock pulses is 1 unless you specify a different number with the -repetitions switch. If any forces have been queued with previous add_simulation_forces commands, the tool automatically simulates these forces before pulsing the clocks.

Arguments

- ***obj_spec***

  A required value that specifies one or more gate_pin objects.

- -repetitions *number*

  An optional switch and integer pair that specifies the number of clock pulses to simulate. If you do not use this switch, the default number of clock pulses is 1.

Examples

The following example shows the use of the simulate_clock_pulses command:

```
SETUP> add_clocks myInst1/OUT
SETUP> add_clocks myInst2/OUT -pseudo_port_name myClk
SETUP> simulate_clock_pulses [get_gate_pins {myInst1/OUT myClk} ] -repetitions 2
```

Related Topics

add_simulation_context

add_simulation_forces

copy_simulation_context

delete_simulation_contexts

delete_simulation_forces

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_simulation_context_list

get_simulation_value_list

report_simulation_contexts

report_simulation_forces

simulate_forces

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

simulate_forces

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Simulates the queued forces in the current simulation context.

Usage

simulate_forces [-latch {normal | transparent}]

Description

The simulate_forces command simulates all the queued simulation forces previously added with the add_simulation_forces command. This command operates only within the current simulation context.

Arguments

- -latch {normal | transparent}

  An optional switch and value pair that controls how latches are to be handled when propagating simulation values.

  A value on the D input of a latch propagates to the Q output of the latch when the enable pin of the latch is simulated to its active value. When the enable pins of the latch is simulated to its inactive value, the Q pin retains its value. The -latch switch controls the behavior when the enable pin of the latch simulates to an X.

  Choose one from the following literals:

  normal — The output of the latch goes to X if the D and Q values are different.

  transparent — The output of the latch is updated to match the value on the D pin.

  This switch is needed when you are trying to extract clock paths from the destination back to the source. Clock gaters exist along the clock paths. These clock gaters have latches that enable propagation of the clock signal. The enable value at the input of the latch must propagate through the latch to disable the clock gater and enable tracing of the clocks through the clock gaters.

  Note

  When using this option, it is important that you later come and verify the proper connection and polarity of the latches along the enable paths to make sure they were implemented correctly.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example shows the effect of adding a force on a particular gate_pin object, and then issuing the simulate_forces command to trigger the tool to simulate that force:

ANALYSIS> set_current_simulation_context myContext
ANALYSIS> add_simulation_forces [get_gate_pins I1] -value 1
ANALYSIS> report_simulation_forces

```
Value   Object
------  ------
1       I1
```

ANALYSIS> get_simulation_value_list [get_gate_pins I1]

```
{ X }
```

ANALYSIS> simulate_forces
ANALYSIS> get_simulation_value_list [get_gate_pins I1]

```
{ 1 }
```

Related Topics

add_simulation_context

add_simulation_forces

copy_simulation_context

delete_simulation_contexts

delete_simulation_forces

get_simulation_context_list

get_simulation_value_list

report_simulation_contexts

report_simulation_forces

simulate_clock_pulses

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

simulate_patterns

Context: patterns -scan, patterns -scan_diagnosis

Mode: analysis

Performs simulation by applying the specified pattern source.

Usage

simulate_patterns [-source {external | bist | random}] [-store_patterns {all | effective | none}] [-patterns_per_pass *patterns_per_pass*]

Description

The simulate_patterns command performs a good-machine simulation. It also performs a fault simulation when there is a current fault list. By default, the command simulates the external patterns previously loaded with the read_patterns command, and the tool retains the pattern order (disables pattern classification) after simulation. The command optionally stores the patterns to the internal pattern set.

The command issues a warning before performing a good-machine simulation when no patterns are stored.

When performing the simulate_patterns command, the tool analyzes the design to determine if split capture cycle or clock_off simulation is needed. If so, the tool turns these functions on automatically. When the pattern source is set to BIST and a named capture procedure (NCP) is used, the tool checks the NCP to determine the necessary sequential depth and other environment settings (such as multiple load or hold_pi) before starting fault simulation. For more information, refer to "Example 2" on page 2984.

Arguments

- -source {external | bist | random}

  An optional switch and literal pair that specifies which pattern source to simulate. By default, the command simulates the external patterns previously loaded with the read_patterns command.

  Note

  External chain patterns are not simulated, and they do not contribute to test coverage for any fault types.

  The "bist" option specifies that the tool simulate only BIST patterns. When running from BIST mode (LFSR is defined or in EDT_LBIST mode) and no external patterns are loaded, the default pattern source is BIST. There must be at least one defined linear feedback shift register (LFSR). The existence of such an LFSR means that the design successfully passed the BIST rules checking when leaving Setup mode.

  You can use the -store_patterns switch to store the BIST patterns when performing good simulation; however, the "-store_patterns effective" option is not permitted. These patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

are different from normal patterns in that they specify the excess values that occur on short scan chains during the load and unload process. Also, the last pattern does not contain an unload of the scan chains.

The "random" option specifies the random patterns, capture clock, and observation point specified previously. You can use this option to create patterns for random pattern testable faults. For fault simulation, you can use this option to evaluate the expected random pattern test coverage.

- -store_patterns {all | effective | none}

  An optional switch and literal pair that specifies whether the simulated patterns should be added to the internal pattern set. By default, the command stores no patterns. However, you can optionally store all patterns or store only the patterns that were effective at detecting additional faults.

- -patterns_per_pass *patterns_per_pass*

  An optional switch and integer pair that simulates the specified number of patterns per pass. The value of <*patterns_per_pass*> must be from 1 to 64. If this argument is not specified, the default value is 64.

Examples

Example 1

The following example performs a good-machine simulation using the current BIST patterns and then adds those patterns to the internal pattern set:

```
simulate_patterns -source bist -store_patterns all
```

Example 2

The following example applies an NCP named "cap_depth3" and then initiates a BIST pattern fault simulation. Before running the simulation, the tool first analyzes the NCP and then both automatically changes the sequential depth to 3 and issues a "set_split_capture_cycle on" command.

```
add_bist_capture_range cap_depth3 0 511
simulate_patterns -source bist
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Warning: Current sequential depth is not large enough to apply named capture procedures.
//           Automatically increase the sequential depth to be 3.
//  largest sequential test depth = 1, largest controllability depth = 1,
//  largest observability depth = 1
//  largest test depth is on /tck (57) (control = 0, observe = 1)
// | ---------------------------------------------------------------------------------
// |          Current split capture setting:      Off
// |                        DRC requirement:      Yes (C3)
// |                                Calling:      set_split_capture_cycle on
// |
// |   Current clock off simulation setting:      Off (optimal)
// | ---------------------------------------------------------------------------------
//
//  ---------------------------------------------------------------------
//  Simulation performed for #gates = 9002  #faults = 25443
//  Run fault simulation without storing patterns.  pattern source = 512 BIST patterns
//  ---------------------------------------------------------------------
//  #patterns  test      #faults  #faults    # eff.    # test         process
//  simulated  coverage  in list  detected   patterns  patterns       CPU time
…
//   64         47.85%    12158     13285       64         0          0.50 sec
//   128        53.66%    10121      2037       56         0          0.56 sec
//   192        56.52%     9118      1003       38         0          0.60 sec
…
```

Related Topics

delete_patterns

read_patterns

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

sizeof_collection

Context: unspecified, all contexts

Mode: all modes

Returns the number of objects in a collection.

Usage

sizeof_collection ***collection***

Description

Returns the number of objects in a collection.

The sizeof_collection command is an efficient way to determine how many objects are in a collection.

Arguments

- ***collection***

  A required string that specifies the collection for which to get the number of objects. The returned size for an empty collection is 0.

Examples

Example 1

The following example shows a simple way to find out how many instances of cell module the current design has.

puts [sizeof_collection [get_instances -filter {type == “cell”}]

```
56781
```

Example 2

The following example shows the size of a collection as it gets appended to.

set a “”
puts [sizeof_collection $a]

```
0
```

append_to_collection a b[7]

```
{b[7]}
```

puts [sizeof_collection $a]

```
1
```

append_to_collection a {b c d}

```
{b[7] b c d}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

puts [sizeof_collection $a]

```
4
```

Related Topics

add_to_collection

append_to_collection

compare_collections

copy_collection

filter_collection

foreach_in_collection

index_collection

is_collection

range_collection

remove_from_collection

sort_collection

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

sort_collection

Context: unspecified, all contexts

Mode: all modes

Sorts a collection based on the value of one or more attributes, resulting in a new sorted collection.

Usage

sort_collection ***collection criteria*** [-descending] [-numerical_sort]

Description

Sorts a collection based on the value of one or more attributes, resulting in a new sorted collection.

The sort is in ascending order by default. In an ascending sort, objects that have Boolean attributes set to false are first followed by Boolean objects that have attributes set to true. The sorting considers the default values of the attributes when the attribute has never been set on the object.

To sort the objects in reverse order, specify the -descending option. To sort the objects numerically, instead of lexically, specify the -numerical_sort option.

Arguments

- ***collection***

  A required string that specifies the collection to be sorted.

- ***criteria***

  A required value that specifies an attribute name or a Tcl list of attribute names to use as sort keys. The order of the attribute name in the list controls the order of priority while sorting.

- -descending

  An optional switch that sorts the collection in reverse order. By default, the sort proceeds in ascending order.

- -numerical_sort

  An optional switch that sorts the collection numerically.

  This is the sorted collection with the switch:

```
core1_28x16/a[1]
core1_28x16/a[2]
core1_28x16/a[3]
core1_28x16/a[10]
core1_28x16/a[20]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

By default, the sort proceeds lexically. This is the sorted collection without the switch:

```
core1_28x16/a[10]
core1_28x16/a[1]
core1_28x16/a[20]
core1_28x16/a[2]
core1_28x16/a[3]
```

Examples

The following example illustrates the use of two attributes to sort a collection. Note how the first attribute has higher priority than the second attribute in the sorting.

register_attribute -name att2 -value_type string -default a
register_attribute -name att1 -value_type string -default a

set_attribute_value {ba bb} -name att1 -value b

```
{ba bb}
```

set_attribute_value {ab bb} -name att2 -value b

```
{ab bb}
```

sort_collection [get_inst {aa ab ba bb}] {att1 att2}

```
{aa ab ba bb}
```

sort_collection [get_inst {aa ab ba bb}] {att2 att1}

```
{aa ba ab bb}
```

sort_collection [get_inst {aa ab ba bb}] {att2 att1} -descending

```
{bb ab ba aa}
```

Related Topics

add_to_collection

append_to_collection

compare_collections

copy_collection

filter_collection

foreach_in_collection

index_collection

is_collection

sizeof_collection

range_collection

remove_from_collection

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

start_silicon_insight

Context: patterns -silicon_insight, patterns -ijtag -silicon_insight, patterns -scan -silicon_insight

Mode: setup, analysis

Starts Tessent SiliconInsight and initializes the data structures.

Usage

start_silicon_insight [-gui]

Arguments

- -gui

  Starts the Tessent SiliconInsight GUI.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

stop_silicon_insight

Context: patterns -silicon_insight

Mode: setup

Stops Tessent SiliconInsight.

Usage

stop_silicon_insight

Description

Closes the CDP and exits the sid_tester process when you are running Tessent SiliconInsight Desktop, SimDUT or “NoTester.” When you are running on an ATE, this command closes the connection to the test program.

Arguments

None

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

system

Context: unspecified, all contexts

Mode: all modes

Passes the specified command to the operating system for execution.

Usage

system ***os_command***

Description

Passes the specified command to the operating system for execution.

The system command executes one operating system command without exiting the currently running application.

Note

There have been isolated incidents in which the “system echo” command resulted in memory overrun. If this occurs, the recommendation is to use the Tcl “puts” command instead.

Arguments

- ***os_command***

  A required string that specifies any legal operating system command.

Examples

Example 1

The following example invokes a synthesis script to synthesize generated RTL, and reloads the synthesized view:

```
write_design -output_file rtl/my_module.v
system “cd synth_work; ./synth_script”
read_verilog gate/my_modul.v -force
set_current_design mytop
```

Example 2

The following example performs an ATPG run, then displays the current working directory without exiting the tool:

```
set_system_mode analysis
create_patterns
system pwd
```

Related Topics

setenv

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

unsetenv

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 9
Command Dictionary (T-Z)

The Command Dictionary describes the application commands for Tessent Shell. For quick reference, the commands appear alphabetically with each beginning on a separate page.

The command usage line syntax conventions are common to all products documented in this manual. See "Conventions for Command Line Syntax" on page 97.

Command Descriptions 2997
trace_flat_model 3000
uncompress_layout 3009
uniquify_instances 3010
unregister_attribute 3012
unregister_static_dft_signal_names 3014
unregister_tcl_command 3015
unsetenv 3016
update_icl_attributes_from_design 3017
update_implication_detections 3019
verify_patterns 3021
verify_tcd_scan 3025
write_atpg_setup 3027
write_cell_library 3030
write_config_data 3034
write_core_description 3037
write_core_timing_constraints 3039
write_design 3040
write_design_import_script 3052
write_design_source_dictionary 3054
write_diagnosis 3056
write_edt_files 3059
write_failing_paths 3071
write_failures 3074
write_fault_sites 3079
write_faults 3083
write_flat_model 3096
write_gds 3100
write_icl 3101
write_loops 3103
write_memory_repair_dictionary 3104
write_modelfile 3108
write_patterns 3109

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_pdl 3140
write_primary_inputs 3142
write_primary_outputs 3143
write_procedure_testbench 3144
write_procfile 3148
write_scan_order 3150
write_scan_setup 3157
write_schematic_dofile 3159
write_test_point_dofile 3160
write_tsdb_data 3162

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Command Descriptions

This section describes the Tessent Shell commands alphabetically, T through Z. The beginning of each command description shows the contexts and modes that support the command and provides the following information:

Context: Lists each context and sub-context that supports the command. “All contexts” means that the command is supported in all available contexts. “Unspecified” means that you do not have to set a context to use the command.

Mode: Lists each mode that supports the command. “All modes” means that the command is supported in all available modes. In some cases, a particular mode supports the command only within certain contexts, and this is noted in parentheses after the mode type.

Table 9-1. Commands T Through Z

Command	Description
trace_flat_model	Traces the flat model, taking into account the current simulation context.
uncompress_layout	Uncompresses a Layout Database.
uniquify_instances	Uniquifies the module definition of the specified instance objects found in obj_spec.
unregister_attribute	Unregisters a user-defined attribute that was registered using the register_attribute command.
unregister_static_dft_signal_names	Unregisters previously registered static DFT signal names.
unregister_tcl_command	Unregisters a command that was previously registered using the register_tcl_command.
unsetenv	Unsets a shell environment variable within the tool environment.
update_icl_attributes_from_design	Updates ICL port, instance, and module attributes from the gate views of a design. This command affects all ICL ports, instances, and modules in the current design.
update_implication_detections	Performs an analysis on the undetected and possibly detected faults to determine if any of those faults can be classified as detected-by-implication.
verify_patterns	Performs pattern verification for diagnosis, and creates, updates, or loads the Diagnosis Startup Cache for use with the diagnosis tool.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 9-1. Commands T Through Z (cont.)

Command	Description
verify_tcd_scan	Reads the specified tcd_scan file for the core specified in the file or, in the case of a file with multiple cores, the core specified with the -core switch, and runs syntactic and semantic checks. The command verifies that all existing scan segments specified the tcd_scan file are traceable.
write_atpg_setup	Writes a test procedure file and a dofile that describes the chains that were created during scan insertion. In addition, for multi-mode scan insertion, the tool also generates a dofile that can be used to verify all of the defined scan modes.
write_cell_library	Writes out a single library file of all library files loaded by the read_cell_library command.
write_config_data	Writes the configuration data presently in memory into a file.
write_core_description	Writes out the core description corresponding to the current chip level.
write_core_timing_constraints	Creates timing (SDC) files that provide the constraints related to the scan path and ATPG settings in the core design.
write_design	Writes the current design in Verilog netlist format to the specified file.
write_design_import_script	Generates a script that can be processed by a synthesis tool to synthesize an RTL design that has been DFT inserted.
write_design_source_dictionary	Writes a design-source dictionary into a file. This file contains the file and directory paths relative to the output file location.
write_diagnosis	Writes diagnosis reports in text, comma separated values (CSV), and layout (physical) marker location formats.
write_edt_files	Creates the files that implement the EDT logic.
write_failing_paths	Reports, for each diagnosed symptom from a failure log, the scan pattern number, suspect(s) and paths.
write_failures	Injects faults and/or writes failing pattern results to a file.
write_fault_sites	Writes path definitions or bridge entries from the internal list to a file.
write_faults	Writes fault information from the current fault list to a file.
write_flat_model	Saves the flattened circuit model, the scan trace, and all DRC-related information to a specific binary file with options to encrypt the file, password protect it, and prohibit (or hide) commands from running on it.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 9-1. Commands T Through Z (cont.)

Command	Description
write_gds	Writes out the all the physical information in the currently open LDB in the GDS format into the specified output file.
write_icl	Writes out ICL modules created and/or read in with the read_icl command to the specified file.
write_loops	Writes a list of all the current feedback loops to a file.
write_memory_repair_dictionary	Writes the CompressBisrChain configuration file.
write_modelfile	Writes all internal states for a RAM or ROM gate into the file that you specify.
write_patterns	Saves the current test pattern set in memory to a file in a specified format.
write_pdl	Writes out PDL iProcs associated with ICL modules to a file.
write_primary_inputs	Writes primary inputs to the specified file.
write_primary_outputs	Writes primary outputs to the specified file.
write_procedure_testbench	Writes a testbench for verifying correct operation of internal signals used in procedures.
write_procfile	Writes existing procedure and timing data to the named test procedure file.
write_scan_order	Writes out a ScanDEF file that describes the newly created scan chains by defining a number of segments. Each segment defines some part of the chain and describes one or more scan cells (instance path, scan in pin, and scan out pin) within that segment. The layout tool uses this information to determine which cells are interchangeable when optimizing the scan chain structure to meet timing constraints and minimize layout congestion.
write_scan_setup	Writes out a dofile containing all necessary steps for scan insertion.
write_schematic_dofile	Writes out a dofile to recreate the current state of a schematic window from the command line.
write_test_point_dofile	Writes out a dofile containing the test points.
write_tsdb_data	Populates the contents of the Tessent Shell Data Base (TSDB).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

trace_flat_model

Context: all contexts

Mode: setup, analysis

Prerequisites: This command can only be used after the design is flattened to the simulation model. This occurs when you first exit setup mode to enter analysis mode, or when you run the create_flat_model command.

Traces the flat model, taking into account the current simulation context.

Usage

trace_flat_model **-from** ***from_obj_spec*** [-to *to_obj_spec*] **-direction** {**forward** | **backward**}
[-tag_condition *tag_attribute_expression*] [-stop_condition *stop_attribute_expression*]
[-controllability {controlling | controlling_allow_reconvergence | constant | unblocked | connected}] [-stop_at_first_tag {On | Off}] [-max_levels {unlimited | *number*}]
[-max_combinational_cells {unlimited | *number*}] [-max_tags {unlimited | *number*}]
[-latch {normal | transparent}] [-map_tag_to_design_module_boundary {ON | OFF}]
[-store_polarity_to_tagged_objects *polarity_var_name*]
[-store_polarity_along_paths_to_tagged_objects *path_polarity_var_name*]
[-store_paths_to_tagged_objects *tag_path_var_name*]
[-store_path_to_stop_point *stop_path_var_name*]
[-store_starts_to_tagged_objects *starts_to_tagged_objects_var_name*]

Description

For information about simulation contexts, refer to the set_current_simulation_context command description.

The tool begins tracing at the objects specified with the -from option. The direction is either forward or backward. You can use equations based on attributes found on gate_pin objects to specify the gate_pins that you want the tracing to stop on, or the gate_pin objects you want to tag when reached.

Note

When you give the gate number as the start point of a trace, the tool uses the output gate_pin object as the start point.

This command returns a collection of reached gate_pins for which the tag condition evaluates to true. When you specify the -tag_condition argument, this command returns a collection of all gate_pin objects the trace stopped on. You can also provide a Tcl variable in which to store the path between the start object and the tagged objects. This enables you to display them in Tessent Visualizer, as shown in two of the examples in the Examples section.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

This command only returns the start point when it is reached again during the trace, that is, when there is a feedback path.

When using this command in setup mode, you may see the following error message because no simulation data is available:

```
Error: Controllability 'unblocked' is the default, but no simulation
data exists in the current simulation context ...
```

To avoid this error, you must specify the "-controllability connected" trace option.

Arguments

- **-from *from_obj_spec***

  A required option that specifies a Tcl list of one or more names of gate_pin, pin, or port objects, or a collection of one or more gate_pin, pin, or port objects. When using pin names or objects, these pins must have been preserved in the flat model.

  Pins that are preserved in the flat model are either all pins on instances of type cell, or pins on instances of type design on which an attribute is set to a non-default value and the set_attribute_option -preserve_boundary_in_flat_model option of that attribute is set to true. These attributes must have been defined on the pin in setup mode prior to creating the flat model.

  Note

  By default, when the tool flattens the design, hierarchical pins are not kept as gate_pin elements. If you need to work with a hierarchical pin, you need to instruct the tool to keep it as part of the flat model by using the "preserve_boundary" option as shown here:

  set_attribute_value *object_spec* -name preserve_boundary

  The tracing is performed starting from each object specified in the from_obj_spec argument.

  If the direction is forward and the start object has no fanout, a null collection is returned. If the direction is backward and the start object has no fan-in or is tied to a constant, a null collection is returned when the -map_tag_to_design_module_boundary switch is set to on. This is because the tieX, tie0, or tie1 gate in the fan-in does not correspond to a design module boundary.

  When tracing from an inout port or pin, you must use the "get_gate_pins -of_pins | -of_ports" command with the "-direction source | destination" option because you are looking for the fan-in or fanout of the inout port or pin. See the -of_pins switch description of the get_gate_pins command for more information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -to *to_obj_spec*

  An optional switch and value pair that directs the trace operation to stop at the objects specified by *to_obj_spec* and to tag them, if reached. The *to_obj_spec* argument can be a Tcl list of one or more names of gate_pin, pin, or port objects, or a collection of one or more gate_pin, pin, or port objects. When using pin names or objects, these pins must have been preserved in the flat model.

  You can use this switch to specify objects as stop points in addition to the objects specified with the -stop_condition switch, and to specify which objects are tagged in addition to the objects specified with the -tag_condition switch.

  When tracing to an inout port or pin, you must use the "get_gate_pin -of_pins | -of_ports" command with the "-direction source | destination" option because you are tracing forward or backward. See the -of_pins switch description of the get_gate_pins command for more information.

- **-direction** {**forward** | **backward**}

  A required string that specifies whether tracing should be performed forward or backward from the defined start object(s).

- -tag_condition *tag_attribute_expression*

  An optional switch and string pair that specifies the tag condition for the trace. All of the reached design objects fulfilling the defined tag condition are returned. The default is an empty string; in this case, the trace stop points are returned. In a non-default case, the trace stop points are not returned if the tag condition expression is not true for these objects. The format for the *tag_attribute_expression* is identical to the -filter option used for the get_pins command.

- -stop_condition *stop_attribute_expression*

  An optional switch and string pair that defines the stop condition for the trace. The default is an empty string; in this case, the trace is stopped at the gate_pin failing the specified controllability requirements. The pins of memory or dff primitives are always stop points even when not explicitly specified.

- -controllability {controlling | controlling_allow_reconvergence | constant | unblocked | connected}

  An optional switch and literal pair that defines the controllability requirements before traversing a gate.

  controlling— Gates are only traversed if the other inputs have simulated constant values making a single input directly controlling the output. Inversions are possible and information about them is stored in the variable specified by the -store_polarity_to_tagged_objects argument.

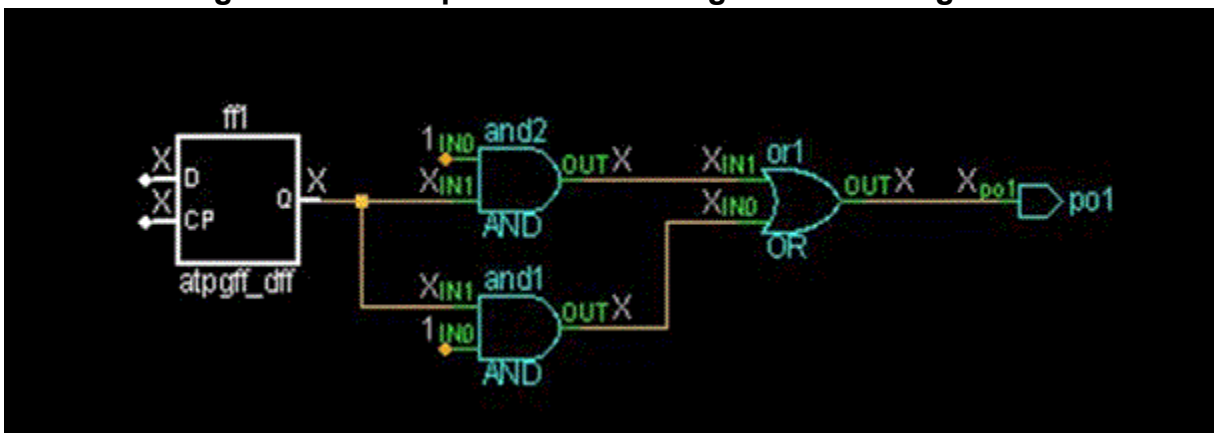
  controlling_allow_reconvergence — Supports tracing reconvergent controlling paths. When this option is specified, the tool first tries to perform the trace with controllability 'controlling' (see description above). If this trace succeeds, the tool behavior is identical with a trace performed with the controllability controlling. If this trace fails, an unblocked backward trace from the target is internally performed to

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

find all the possible controlling source candidates. These candidates are then investigated by simulating 0 and 1 to them and observing the effect on the destination.

Figure 9-1 shows an example of a reconvergent controlling path from ff1/Q to po1.

Figure 9-1. Example of a Reconvergent Controlling Path



Information about inversions is stored in the variable specified by the -store_polarity_to_tagged_objects switch.

Note

In the case where the reconvergence analysis was necessary to find the controlling source, the variable specified by the -store_paths_to_tagged_objects switch returns an empty collection.

Unnecessarily specifying controllability controlling_allow_reconvergence instead of controlling has negative impact on tool's performance.

constant — Only pins with a constant value are traversed. This mode is useful to find the source of a constant value.

unblocked — Gates are only traversed if the other inputs do not have simulated constant values making the input blocked from the output pin. Only unblocked combinational paths are traversed.

connected—With this value, gates are always traversed when reached. You should use this option when this command is used in setup mode because no simulation data exists yet.

- -stop_at_first_tag {On | Off}

  An optional switch and literal pair that specifies whether to interrupt tracing when the tool finds the first tag object with the specified controllability. This is useful when checking for at least one occurrence of a given condition in the fan-in or fanout of a pin and you do not need to find them all.

- -max_levels {unlimited | *number*}

  An optional switch and value pair that defines the maximum levels of gates to trace. The default is unlimited but you can specify a positive integer to limit the level count. The gates

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

are the primitive gates as seen in the flat model. An AND-OR gate has two levels of primitive gates in the flat model and counts as two levels of gates for this limit. If you want to only count cells irrespective of their complexity, use the -max_combinational_cells switch instead.

- -max_combinational_cells {unlimited | *number*}

  An optional switch and value pair that defines the maximum number of combinational cells to trace. The default is unlimited. Each combinational cell is counted as one cell except for inverters and buffers which do not count at all. Even if a cell has several levels of primitives inside it, it is not counted as more than one cell. To limit the search to a maximum number of primitive gates, use the -max_levels switch instead.

- -max_tags {unlimited | *number*}

  An optional switch and value pair that directs the trace operation to stop when the number of tagged objects reaches the specified limit. The default is unlimited. This is useful when you are coding your own custom DRC using the register_drc command and you only want to detect an initial quantity of violations.

- -latch {normal | transparent}

  An optional switch and literal pair that specifies the behavior of tracing through a latch.

  normal — A latch is traced through only if the simulated value in the current simulation context makes the latch satisfy the specified controllability requirements. In case of "-controllability connected" all latch input pins - including the latch enable pin - are traced through.

  transparent — All latches are traced through the data input, set and reset pins regardless of the value at the latch enable pin. The latch enable pin is not traced through.

- -map_tag_to_design_module_boundary {ON | OFF}

  An optional switch and literal pair that specifies whether to map the tag points to gate_pins outside of library cells. When this switch is ON, the command maps the tag points to gate_pins outside of library cells and never inside the cells. Library cells can either be cells from a Tessent Cell library or `celldefine Verilog modules.

- -store_polarity_to_tagged_objects *polarity_var_name*

  An optional switch and string pair that specifies the name of a Tcl variable in which to store a list of 0 or 1 values to reflect the presence of an inversion between the start object and the tag object(s). If an inversion exists between the start and tag object(s), the value 1 is stored in the list; otherwise, the value 0 is stored. This option can only be used with the -controllability controlling option; inversion is meaningless in all other cases.

- -store_polarity_along_paths_to_tagged_objects *path_polarity_var_name*

  An optional switch and string pair that specifies the name of a Tcl variable in which to store a list of 0 or 1 values. The values reflect the presence of inversions between the start object and the tag object(s). If an inversion exists between the start and tag object(s), a value of 1 is stored in the list; otherwise, the value 0 is stored. This switch provides a list of 0 and 1 values representing the local polarity of each node that the corresponding collection found

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

by the -store_paths_to_tagged_objects switch. This option can only be used with the -controllability controlling option; inversion is meaningless in all other cases.

- -store_paths_to_tagged_objects *tag_path_var_name*

  An optional switch and string pair that specifies to store a list of paths in the variable *tag_path_var_name*. Each path holds a collection containing all of the traversed gate_pins between the start object and the tagged object(s). The list is in the order specified by the -direction option and includes the start object and the reached tagged object.

  Note

  In the case where the reconvergence analysis was necessary to find the controlling source, the variable specified by this switch returns an empty collection.

- -store_path_to_stop_point *stop_path_var_name*

  An optional switch and string pair that specifies to store the path between the start object and the gate_pin where the trace stopped independent of whether the stop point is a tagged object. This switch is only valid when -direction is specified as backward and -controllability is specified as controlling or constant. In this case, only one start object is supported and only one stop point can ever be found. This mode is useful to trace a clock source and display the path to the stop point if an appropriate source is not reached.

- -store_starts_to_tagged_objects *starts_to_tagged_objects_var_name*

  An optional switch and string pair that specifies to store the start point from which all tagged objects were reached from. The return value is a single collection containing the start points from which the tagged objects were seen from.

Examples

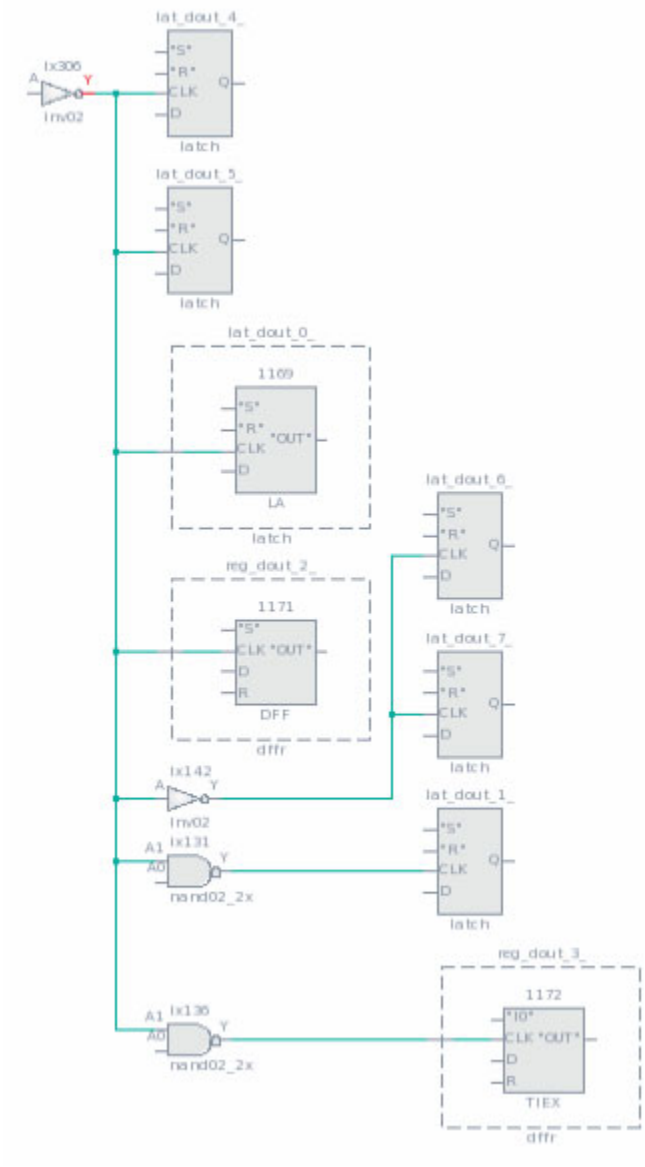
Example 1

The following example traces the unblocked fanout of a pin. Because no -tag_condition is specified, the tool automatically tags every node with no unblocked fanout. The paths to the tag elements are stored in the Tcl variable tag_paths. The foreach command iterates on the list of paths and the add_schematic_path command displays the paths in the Tessent Visualizer Flat Schematic. Figure 9-2 shows the resulting Flat Schematic.

```
set start_pin [get_pins core_i/test_design_i/ix306/Y]
trace_flat_model -from $start_pin -direction forward \
        -store_paths_to_tagged_objects tag_paths
defer_schematic_updates {
        foreach path $tag_paths {
                add_schematic_path $path -direction forward
        }
}
add_schematic_objects $start_pin -highlight red
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 9-2. Flat Schematic Display After add_schematic_path Command



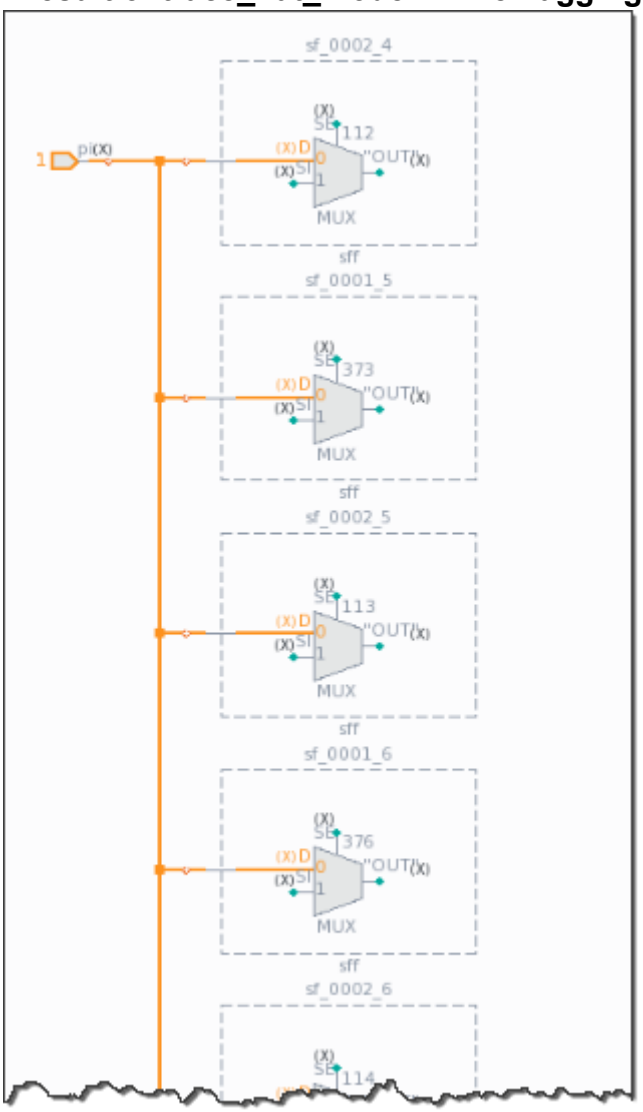
Example 2

The following example performs a controlling walk forward from a primary input and collects any sff instances traversed along the way. The paths from the start point to all tagged elements are displayed in the Flat Schematic, as shown in Figure 9-3. Notice how the set_gate_report drc command is set to stable_after_setup; this corresponds to the simulation conditions used by the trace_flat_model command. The tagged instances have a controlling value on the select pin. Those without a controlling value on the select pin were not traversed and therefore their output pins were not tagged.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
trace_flat_model -from pi \
        -tag_condition {module_name == sff} \
        -direction forward \
        -store_paths_to_tagged_objects tag_paths -controllability controlling
set_gate_report simulation_context stable_after_setup
defer_schematic_updates {
        foreach path $tag_paths {
                add_schematic_path $path -direction forward
        }
}
```

Figure 9-3. Result of trace_flat_model While Tagging Instances



Example 3

The following example uses the trace_flat_model command to find paths between output and inout pins of modb to pins of modd. Note how the -stop_at_first_tag option is set to on, ensuring that only one path from each modb pin is reported.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_system_mode setup
set_attribute_value [get_instances -of_module {modb modd}] -name preserve_boundary
set_system_mode analysis
trace_flat_model -from [get_pins u4/* -direction {output inout}] -direction forward \
                 -stop_condition preserve_boundary \
                 -tag_condition {module_name == modd} \
                 -stop_at_first_tag on \
                 -store_paths_to_tagged_objects paths
if {[llength $paths] > 0} {
  puts "pins of modb fanning out to modd:"
    foreach path $paths {
      puts [get_single_name [index_collection $path 0]]
    }}
```

Related Topics

get_trace_flat_model_option

set_current_simulation_context

set_trace_flat_model_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

uncompress_layout

Context: patterns -scan_diagnosis

Mode: analysis

Uncompresses a Layout Database.

Usage

LDBs created after v2013.2: uncompress_layout

LDBs created with v2013.2 and before: uncompress_layout ***layout_database_name***

Description

Uncompresses a Layout Database.

You must open the Layout Database using the open_layout command before performing the uncompress operation.

See "Layout Database Compression and Decompression" in the *Tessent Diagnosis User's Manual* for complete information.

Arguments

- ***layout_database_name***

  A required string that specifies the location to and the name of the Layout Database to be created during the uncompress operation.

Related Topics

compress_layout

create_layout

open_layout

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

uniquify_instances

Context: all contexts

Mode: insertion

Uniquifies the module definition of the specified instance objects found in obj_spec.

Usage

uniquify_instances ***obj_spec*** [-module_uniquification_suffix *module_uniquification_suffix*] [-silent]

Description

Uniquifies the module definition of the specified instance objects found in ***obj_spec***.

If the instances specified by ***obj_spec*** have a module definition that is used more than once below the current design, the module name is made unique by appending the specified -module_uniquification_suffix value to its name.

This command returns a collection of all modules associated with the instances in the ***obj_spec*** irrespective of whether the module name was uniquified or not.

Arguments

- ***obj_spec***

  A required value that specifies the instances whose module name is to be uniquified. The ***obj_spec*** value must be a Tcl list of one or more instance names, or a collection of one or more instances as generated by the get_instances command

- -module_uniquification_suffix *module_uniquification_suffix*

  An optional switch and value pair that specify a string to append to the module name if the instance's module definition is used more than once below the current design. The default module_uniquification_suffix is "_#" where # begins at 1 and increments by one until a unique module name is found.

- -silent

  An optional argument that specifies to ignore specified instance objects that do not exist or instances that are not of type design, and to not generate error messages.

Examples

Example 1

The following example requests all instances found within u2 to be uniquified. All instances of design modules in this example are instances of module moda and modb. From the result of the command, you can see that there are only three instances of moda within the current design and they are all within the u2 instance. Once the first two instances were uniquified to moda_1 and moda_2, the third instance was not changed as it was now unique. Because modb was uniquified, this means that it is used in other instances of the current design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

uniquify_instances [get_instances u2/* -of_type design]

```
{moda moda_1 moda_2 modb_1}
```

Example 2

The following example overwrites the default module_uniquification_suffix of "_#" to "_uniq#" so that it is easier to identify which modules were uniquified:

uniquify_instances [get_instances u2/* -of_type design] -module_unification_suffix _uniq#

```
{moda moda_uniq1 moda_uniq2 modb_uniq1}
```

Example 3

The following example appends to the my_design_modules collection the collection of modules returned by the uniquify operation. Note that by definition, uniquify_instances returns a collection with a different module object for each instances object because uniquified modules are only ever instantiated once.

**append_to_collection my_design_modules **
[uniquify_instances [get_instances u2/* -of_type design]]

Related Topics

create_instance

delete_instances

get_common_parent_instance

rename_instance

replace_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

unregister_attribute

Context: unspecified, all contexts

Mode: all modes

Unregisters a user-defined attribute that was registered using the register_attribute command.

Usage

unregister_attribute **-name *attribute_name*** [-object_types *type_list*] [-silent]

Description

Unregisters a user-defined attribute that was registered using the register_attribute command.

Following de-registration, the attribute is unavailable for use.

Pre-defined attributes cannot be de-registered.

If ***attribute_name*** does not exist, or if a design object type in *type_list* does not exist, an error message displays unless the -silent is specified.

Arguments

- **-name *attribute_name***

  A required switch and value pair that specifies the name of the attribute to de-register.

- -object_types *type_list*

  An optional switch and value pair that specifies the object type to de-register. The *type_list* string is a Tcl list of one or more object types (for example, module, instance, port, pin, net, and so on). The *type_list* should contain only object types where the attribute was registered, that is, a subset of what was specified with the register_attribute command.

  If this argument is not specified, by default, the new attribute is de-registered for all object types.

- -silent

  An optional switch that suppresses error messages if the given attribute_name is not registered.

Examples

The following example de-registers the attribute added_by for only the net object type.

```
unregister_attribute -name added_by -object_type net
```

Related Topics

get_attribute_list

get_attribute_option

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_attribute_value_list

register_attribute

report_attributes

reset_attribute_value

set_attribute_options

set_attribute_value

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

unregister_static_dft_signal_names

Context: dft, patterns

Mode: setup

Prerequisites: You cannot unregister static dft signal names after the add_dft_signals command is run.

Unregisters previously registered static DFT signal names.

Usage

unregister_static_dft_signal_names {***name*** ...} | **-all_user_defined**

Description

Unregisters previously registered static DFT signal names. After you have run the add_dft_signals command, it is no longer possible to unregister a static DFT signal. Only DFT signal previously registered with the register_static_dft_signal_names command can be unregistered. The built-in names as well as the logic test control signals registered using the tcd_scan metadata cannot be unregistered.

Arguments

- ***name***

  A repeatable string value that specifies the names of the DFT signals to unregister. The string can also be a well-formatted Tcl list of names. These names must correspond to DFT signal names that were registered using the register_static_dft_signal_names command.

- **-all_user_defined**

  A Boolean switch specifying that all DFT signal names previously registered using the register_static_dft_signal_names command are to be unregistered.

Examples

The following example registers a DFT signal name then unregisters it:

```
register_static_dft_signal_names clockb_select
unregister_static_dft_signal_names clockb_select
```

Related Topics

register_static_dft_signal_names

report_dft_signal_names

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

unregister_tcl_command

Context: unspecified, all contexts

Mode: all modes

Unregisters a command that was previously registered using the register_tcl_command.

Usage

unregister_tcl_command ***command_name***

Description

Unregisters the command that was previously registered using the register_tcl_command.

An error is generated if the command was registered before the last lock_current_registration command invocation. Commands loaded as part of a plugin are automatically locked.

Arguments

- ***command_name***

  A required string that specifies the command to unregister.

Examples

The following example unregisters the commands ABC and DEF. The command ABC was registered prior to the last lock_current_registration command so it cannot be unregistered:

unregister_tcl_command ABC

```
//  Error: Command locked and can't be un-registered
```

unregister_tcl_command DEF

Related Topics

lock_current_registration

register_tcl_command

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

unsetenv

Context: unspecified, all contexts

Mode: all modes

Unsets a shell environment variable within the tool environment.

Usage

unsetenv ***variable_name***

Description

Unsets a shell environment variable within the tool environment.

The unsetenv command unsets a shell environment variable within the current tool environment. When you only need a variable unset within the tool, this command saves you the time and trouble of exiting, issuing the UNIX unsetenv command in the shell, and re-invoking.

Note

A shell variable you unset with this command is unset only with respect to the current tool session. The command does not alter the value of the variable in the invocation shell.

To set or change the value of a variable, use the setenv command. To view the value of the variable, use the system command to pass an appropriate shell command (echo or env for example) to the operating system for execution.

Arguments

- ***variable_name***

  A required string that specifies the name of the variable.

Examples

The following example unsets the SSH_AGENT_PID environment variable within the current tool environment, then confirms that it is unset:

```
system {echo $SSH_AGENT_PID}
```

```
66216
```

```
unsetenv  SSH_AGENT_PID
system {echo $SSH_AGENT_PID}
```

Related Topics

setenv

system

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

update_icl_attributes_from_design

Context: all contexts

Mode: all modes

Updates ICL port, instance, and module attributes from the gate views of a design. This command affects all ICL ports, instances, and modules in the current design.

Usage

update_icl_attributes_from_design [-verbose]

Description

This command updates the following ICL module attributes:

- **tessent_design_gate_ports —** This attribute refers to the design port corresponding to the ICL port. The attribute value is the name of the design port as it appears in the netlist.
- **tessent_design_instance —** This attribute refers to the design instance corresponding to the ICL instance. The attribute value is the hierarchical name relative to the parent ICL instance.
- **tessent_design_rtl_gate_port_mapping —** This attribute maps the RTL name to the netlist name for those ports that appear in the following attribute name lists:
  - forced_high_input_port_list
  - forced_low_input_port_list
  - forced_high_output_port_list
  - forced_low_output_port_list
  - forced_high_internal_input_port_list
  - forced_low_internal_input_port_list
  - tessent_ground_port_list
  - tessent_power_port_list
  - tessent_clock_domain_labels

  The attribute value is a list of (RTL name, netlist name) pairs for these ports.

The "write_design -tsdb" command automatically invokes the update_icl_attributes_from_design command. The insert_test_logic also invokes this command after insertion.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- [-verbose]

  An optional switch that issues warnings when a port name cannot be found or matched in the netlist.

Related Topics

icl_port

icl_instance

icl_module

write_design

insert_test_logic

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

update_implication_detections

Context: dft -edt, patterns -scan

Mode: analysis

Performs an analysis on the undetected and possibly detected faults to determine if any of those faults can be classified as detected-by-implication.

Usage

update_implication_detections

Description

Performs an analysis on the undetected and possibly detected faults to determine if any of those faults can be classified as detected-by-implication.

The update_implication_detections command is automatically issued after the simulate_patterns command executes a fault simulation when using external patterns or when using internal patterns in non-Atpg. The update_implication_detections command is also automatically issued as part of the create_patterns command.

By invocation default, only scan-path-associated faults for the detected-by-implication classification are analyzed. The following faults are classified as detected-by-implication when the update_implication_detections command is issued:

- A stuck-at-1 fault on the set input line of a transparent latch, scan latch, scan D flip-flop, shadow, copy, or sequential cell when the tool detects the stuck-at-1 fault on the output.
- A stuck-at-1 fault on the reset input line of a transparent latch, scan latch, scan D flip-flop, shadow, copy, or sequential cell when the tool detects the stuck-at-0 fault on the output.
- A stuck-at-0 fault on a clock input line of a transparent latch, scan latch, scan flip-flop, shadow, copy, or sequential cell when the tool detects both the stuck-at-0 and stuck-at-1 faults for the associated data line.
- A stuck-at-0 fault on a data input line of a transparent latch, scan latch, scan D flip-flop, shadow, copy or sequential cell when the tool detects the stuck-at-0 fault and no other ports can capture a 1.
- A stuck-at-1 fault on a data input line of a transparent latch, scan latch, scan D flip-flop, shadow, copy, or sequential cell when the tool detects the stuck-at-1 fault on the data output and no other port can capture a 0.

Arguments

None

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

report_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

verify_patterns

Context: patterns -scan_diagnosis

Mode: analysis

Performs pattern verification for diagnosis, and creates, updates, or loads the Diagnosis Startup Cache for use with the diagnosis tool.

Usage

verify_patterns {**-Diagnosis** |
-Create_startup_cache ***startup_cache_name*** [-Force_open] |
-Update_startup_cache ***startup_cache_name*** [-Force_open] |
-Load_startup_cache ***startup_cache_name***}

Description

Performs pattern verification for diagnosis, and creates, updates, or loads the Diagnosis Startup Cache for use with the diagnosis tool.

See "Pattern Verification and the Diagnosis Startup Cache" in the *Tessent Diagnosis User's Manual* for complete information.

Arguments

- **-Diagnosis**

  A required switch that only runs pattern verification without storing the pattern verification results into Diagnosis Startup Cache. The diagnosis tool outputs the pattern verification results to stdout and in the tool's transcript.

  If the pattern verification passes, then Tessent Diagnosis does not re-run pattern verification following the diagnose_failures command, unless there changes to the current pattern set.

- **-Create_startup_cache** ***startup_cache_name***

  A required switch and string pair that runs pattern verification and creates a new Diagnosis Startup Cache. If ***startup_cache_name*** already exists, then the tool produces an error.

  If the pattern verification passes, then the tool creates a new Diagnosis Startup Cache with the given filename. If the pattern verification does not pass, then the tool outputs the pattern verification results to stdout and does not create the Diagnosis Startup Cache.

- **-Update_startup_cache** ***startup_cache_name***

  A required switch and string pair that loads an existing Diagnosis Startup Cache and compares the contents of the Diagnosis Startup Cache (netlist, patterns, and so on) to this information in the current Tessent Diagnosis session. If ***startup_cache_name*** does not exist, then the tool produces an error.

  If the specified Diagnosis Startup Cache does not contain the current pattern sets or does not match the current tool version, then the tool runs pattern verification.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the pattern verification passes, then the tool updates the specified Diagnosis Startup Cache. If the pattern verification does not pass, then the tool outputs the pattern verification results to stdout and does not update the Diagnosis Startup Cache.

- **-Load_startup_cache *startup_cache_name***

  A required switch and string pair that loads an existing Diagnosis Startup Cache. If ***startup_cache_name*** does not exist, then the tool produces an error and exits.

  If the specified Diagnosis Startup Cache contains the current pattern sets and matches the current tool version, then the tool loads the Diagnosis Startup Cache. Otherwise, the tool produces an error.

- -Force_open

  An optional switch used with either -Create_startup_cache or -Update_startup_cache that unlocks a locked Diagnosis Startup Cache.

Examples

Example 1

This example demonstrates creating a Diagnosis Startup Cache for future use in diagnosis with Tessent Diagnosis.

In the Tessent Diagnosis scan diagnosis point tool, the following command sequence loads the test patterns, performs the pattern verification, and creates a Diagnosis Startup Cache named *my_startup_cache*:

```
read_patterns pat1
read_patterns pat2 -append
verify_patterns -create_startup_cache ./my_startup_cache
exit
```

Subsequently, you re-invoke Tessent Diagnosis, load the patterns and newly created Diagnosis Startup Cache (*my_startup_cache*), and perform diagnosis using the following command sequence:

```
read_patterns pat1
read_patterns pat2 -append
verify_patterns -load_startup_cache ./my_startup_cache
diagnose_failures log1
diagnose_failures log2
......
exit
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

This example command sequence illustrates updating an existing Diagnosis Startup Cache (*my_startup_cache*) for future use by Tessent Diagnosis:

```
read_patterns pat1
read_patterns pat2 -append
read_patterns pat3 -append
verify_patterns -update_startup_cache ./my_startup_cache
exit
```

Subsequently, you re-invoke Tessent Diagnosis, load the patterns and updated Diagnosis Startup Cache (*my_startup_cache*), and perform diagnosis using the following command sequence:

```
read_patterns pat1
read_patterns pat2 -append
read_patterns pat3 -append
verify_patterns -load_startup_cache ./my_startup_cache
diagnose_failures log1
diagnose_failures log2
......
exit
```

Example 3

The following example provides the command sequence you would perform to use an existing Diagnosis Startup Cache with the Tessent Diagnosis Server:

1. Create the Diagnosis Startup Cache (*my_startup_cache*) with the following commands using the Tessent Diagnosis scan diagnosis point tool:

   ```
   read_patterns pat1
   read_patterns pat2 -append
   verify_patterns -create_startup_cache ./my_startup_cache
   exit
   ```

2. In the Tessent Diagnosis Server, add the patterns and load the Diagnosis Startup Cache (*my_startup_cache*) using the following command sequence:

   ```
   add_pattern  monitor1 pat1
   add_pattern monitor1 pat2
   add_startup_cache monitor1 ./my_startup_cache
   ...
   ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

In this example, Tessent Diagnosis does not create a Diagnosis Startup Cache or re-run pattern verification during diagnosis:

```
read_patterns pat1
read_patterns pat2 -append
verify_patterns -diagnosis  // assume pattern verification passed
diagnose_failures log1 // does not re-run pattern verification
```

Related Topics

diagnose_failures

read_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

verify_tcd_scan

Context: dft -scan

Mode: setup

Reads the specified tcd_scan file for the core specified in the file or, in the case of a file with multiple cores, the core specified with the -core switch, and runs syntactic and semantic checks. The command verifies that all existing scan segments specified the tcd_scan file are traceable.

Usage

verify_tcd_scan -tcd_scan ***filename*** [-core *core* …] [-active_child_scan_mode *mode*] [-output *output_file*]

Description

Use this command to verify manually-created simple tcd_scan files required as inputs for scan stitching. You can use it for Gate-level Verilog modules, celldefine models and Tessent library models. Do not use it to verify TCD files that are generated by Tessent scan stitching tools.

If you have more than one scan mode in the TCD file, you must specify the mode to be verified with the -active_child_scan_mode switch.

The command also verifies at least one scan segment specified in the tcd_scan file. A temporary wrapper with a unique name is created to perform this test and if a tracing error occurs, the wrapper is kept for debug. It is deleted if no error occurs.

Note

The command does not support modules of type “OCC” that may require a test setup procedure for correct tracing. It also does not support verification of chains in cores that are connected to an EDT instance. These chains are assumed to be stitched and are automatically verified during scan stitching at the parent level. The tool only verifies scan segments that are connected to the boundary of the container.

Arguments

- -tcd_scan ***filename***

  A required switch and string pair that specifies the tcd_scan file to be read.

- -core *core* …

  An optional switch and string or repeatable string that specify one or more cores for which to verify the tracing of scan segments as configured by the tcd_scan file. These cores may be Tessent library models or Verilog modules.

  If -core is not specified the default is to check all cores defined in the tcd_scan file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -active_child_scan_mode *mode*

  An optional switch and string pair that specifies a specific value for the active_child_scan_mode attribute to be set on the top module (and inherited by the models and modules being tested).

- -output *output_file*

  An optional switch and string pair that specifies to print a tcd_scan file named output_file that includes the time and Tessent version on which it was successfully tested.

  During the process of verifying the tcd_scan file, the tool automatically identifies and fixes small discrepancies in the output file. To avoid overwriting the tcd_scan file being tested, the output file cannot have the same name as the tcd_scan file.

Examples

The following example shows how to verify a core.

```
set_context dft -scan
read_verilog A.v
read_cell_lib my_lib.lib
verify_tcd_scan -tcd A.tcd_scan -core A -output A_out.tcd_scan
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_atpg_setup

Context: dft -scan

Mode: analysis, insertion

Prerequisites: Scan chains must be inserted.

Writes a test procedure file and a dofile that describes the chains that were created during scan insertion. In addition, for multi-mode scan insertion, the tool also generates a dofile that can be used to verify all of the defined scan modes.

Usage

write_atpg_setup ***basename*** [-Replace] [-NO_merge] [-NO_Edt] [-All_internal_clocks]

Description

Writes a test procedure file and a dofile that describes the chains that were created during scan insertion. In addition, for multi-mode scan insertion, the tool also generates a dofile that can be used to verify all of the defined scan modes.

You can use these files to identify the scan chains when loading the scan design into Tessent FastScan. By default, the test procedure and the dofile for newly inserted scan chains are merged with those for existing scan chains.

The order of the commands in the test procedure file determine the different levels of test procedure file information produced by this command.

- If you execute write_atpg_setup without first using the read_procfile command, a test procedure file that contains default timeplates and the procedures needed for ATPG setup (for example, shift, load_unload, and test_setup) are created.
- If you run read_procfile and load a test procedure file that contains timeplate definitions, non-scan procedure (for example, capture) definitions, and empty or template scan procedures, and then execute the write_atpg_setup command, a more complete test procedure file is created.

  Note

  If the test procedure file loaded with the read_procfile command contains scan procedures (for example, shift and load_unload), they are replaced by ones generated with the write_atpg_setup command based on test logic inserted during the current session. The tool issues a W14 warning each time it replaces a procedure. However, it uses timeplates specified in the original procedure file for the newly-generated procedures. The only exceptions are for the test_setup procedure. Instead of being replaced, the events created by the write_atpg_setup command are appended to the end of the test_setup procedure.

By default, the tool automatically detects a SoC DFT flow and writes out EDT-specific commands to the ATPG setup files.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The dofile defines a Tcl procedure named “tessent_scan_setup” that accepts a parameter that is used to specify the name of the scan mode that you want to set up. For example, to define the scan chains that are accessible in “unwrapped” mode, you must add a call to tessent_scan_setup with the parameter set to “unwrapped”, as shown below:

```
dofile setup.dofile
tessent_scan_setup unwrapped
```

Finally, for designs that had pre-inserted scan chains, the default is that these scan chains are not defined regardless of the selected mode. However, there is an optional argument to the tessent_scan_setup procedure that can be set to “1" if you want to also define these chains for the subsequent step. The example for this is shown below:

```
dofile setup.dofile
tessent_scan_setup unwrapped 1
```

In addition to generating the setup dofile, the tool also generates a verification dofile that enters the patterns -scan context, sets up each scan mode, and triggers a system mode transition to verify that the inserted scan chains can be traced in that mode. By default, this procedure does not include any pre-defined chains. For designs with pre-existing chains, two additional procedures are created: verify_all_scan_modes_with_existing_chains and verify_all_scan_modes_without_existing_chains. You can choose the desired procedure after sourcing the verification dofile.

Hybrid TK/LBIST Flow Specifics

The regular scan insertion dofile includes additional commands that specify the pin names for scan enable and BIST enable signals. The tool automatically includes logicBIST-related commands either X-bounding or bounding of timing exceptions paths is implemented. Refer to the *Hybrid TK/LBIST Flow User’s Manual* for complete information.

Arguments

- ***basename***

  A required string that specifies the root name for the test procedure and dofile. The files produced are ***basename***.testproc and *basename.dofile*.

- -Replace

  An optional switch that replaces the contents of a file that already exists. By default, files are not replaced.

- -No_merge

  An optional switch that prevents merging the test procedure and the dofile of the newly inserted scan chains with those of the existing scan chains. By default, the test procedures and dofiles are merged.

- -NO-Edt

  An optional switch that specifies that EDT-specific commands are not written to the ATPG setup files. By default, the tool automatically detects a SoC DFT flow and writes out EDT-

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

specific commands to the ATPG setup files. For more information, see the set_edt_options command.

- -All_internal_clocks

  An optional switch that specifies to write all internally defined clocks to the timeplate, to the ATPG dofile, and to the shift procedure.

Examples

The following example writes the test procedure, the dofile, and the new netlist for the inserted scan chains to the specified filenames:

```
add_clocks 1 clk1
add_clocks 0 clk0
set_system_mode analysis
insert_test_logic

write_atpg_setup scan -replace
write_design -output_file scan.v
```

Related Topics

insert_test_logic

read_procfile

set_edt_options

write_design

write_procfile

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_cell_library

Context: all contexts

Mode: all modes

Writes out a single library file of all library files loaded by the read_cell_library command.

Usage

write_cell_library ***new_lib_filename*** [-replace]
[-INCLUDE_CREATED_dft_cell_selections | -ONLY_DFT_cell_selections]

Description

Writes out a single library file of all library files loaded by the read_cell_library command, or alternately (if not using Tessent Shell) loaded at tool invocation with the -library switch; this single library file can subsequently be loaded by the read_cell_library command or at tool invocation with the -library switch to reproduce the internal library state that would exist if the same set of original files was loaded.

Note

Library cells that were read into the tool using the read_verilog command with associated Verilog compiler directives like `celldefine and `endcelldefine are not included in the output of the write_cell_library command.

A dft_cell_selection wrapper is optional information that can be in the same file as the Tessent cell library models, or in a separate file. Use the dft_cell_selection wrapper to specify the cells (which are also models in the cell library) to use for DFT test logic applications such as scan insertion. The get_dft_cell command also accesses information from the dft_cell_selection wrapper.

For example, if you need a 2-input AND cell and there is one specified in the dft_cell_selection wrapper, the tool uses that referenced model if you need such a cell for the test logic being inserted into your netlist, or if you use the “get_dft_cell and -inputs 2” command.

If no dft_cell_selection wrapper is provided, a default is created when a command is issued which causes elaboration, such as get_dft_cell or set_current_design, and is given the name “default”. A model is selected with an appropriate learned simulation_function (“and” in the example described above), and an attempt is made to determine which of those models (if any) is best. Note that there is no need to provide the simulation_function which is written out for informational purposes when you issue write_cell_library, as it is discarded when the file is parsed. Models with a cell_type specified in the model in the tessent cell library file (such as “cell_type = and”) are preferred over models with no cell_type specified when the file is parsed. Also, any model already instantiated in the current design netlist is preferred over a model which has no instances in the current design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If both the default dft_cell_selection wrapper and models are written out, the learned cell_type is also written in appropriate models. These should also be checked and corrected if the file is to be read back in for use in subsequent runs of some test tool requiring a tessent cell library. You can find these cells by searching for the string "// Learned cell_type", as illustrated next.

Note

All parsed input is considered to be user input, regardless of comments or names. If you do not change the default dft_cell_selection name from "default", then that is the name for your selected dft cells. If you do not check and edit the default dft_cell_selection wrapper, but instead read it back as written, all the learned cell_type is considered to be user-provided and approved cell_type, not learned. This is true despite any comment in the parsed library file saying a cell_type was learned—that is ignored because when parsed, the cell_type is now user-specified, not learned. These cells no longer defer to other cells whose cell_type was originally specified in the library models.

The following examples illustrate the output resulting from a model having its cell_type learned rather than specified when the model was parsed, combined with -include_default_dft_cell_selection. Note that the simulation_function is always learned and is always written out, but it is informative only and is discarded and recalculated each time the file is parsed. A cell_type must have a compatible simulation_function (for example, simulation_function should be mux for cell_type mux or cell_type clock_mux) to be used for test logic, even if you set the cell_type.

If the -include_default_dft_cell_selection switch is specified, and a default dft_cell_selection wrapper exists:

```
model TC4MUX2XC
  (Y, D0, D1, S0)
(
  cell_type = mux;  // Learned cell_type
  simulation_function = mux;
  ....
```

If the -include_default_dft_cell_selection switch is not specified, or a default dft_cell_selection wrapper does not exist:

```
model TC4MUX2XC
  (Y, D0, D1, S0)
(
  simulation_function = mux;
```

Arguments

- ***new_lib_filename***

  A required string that specifies the location of the file to which the library contents are written in cell/model format. If this file is subsequently loaded using "read_cell_library new_lib_filename" or alternately (if not using Tessent Shell) using "-library

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

new_lib_filename" at invocation, the internal library state that existed when the file was written is effectively restored.

- -replace

  An optional switch that specifies to replace the contents of the file specified by the new_lib_filename argument if the file already exists.

- -INCLUDE_CREATED_dft_cell_selections

  An optional switch that writes out the created dft_cell_selection information. If no dft_cell_selection is parsed, the default dft_cell_selection is written at the top of the tessent cell library file, before the models are written. Otherwise, the parsed dft_cell_selections is written, including the augmented dft_cell_selection information. Models with a learned cell_type also have the cell_type written in the model. If this switch is not specified, only user specified cell_type are written in the model.

- -ONLY_DFT_cell_selections

  An optional switch that lets you write out only the dft_cell_selections to a file. If no parsed dft_cell_selection exists, in which case a default dft_cell_selection is written, we recommend that you change the name from "default" and adjust the default dft_cell_selection to reference the cells that you prefer, and then use the result in future tool invocations to achieve deterministic selection of the cells used for test logic. The file containing the edited dft_cell_selection wrapper can then be given to the read_cell_library command as a Tessent cell library filename.

Examples

Example 1

The following example assumes the tool is invoked with the -library switch and the library files are loaded. The write_cell_library command writes the cells/models contained in those files to the new library file, *new_library.tcelllib*. If the *new_library.tcelllib* file already exists, the file is not written.

```
$ fastscan -library mylib.tcelllib
SETUP> write_cell_library new_library.tcelllib
```

Example 2

The following example assumes the tool is invoked without the -library switch and the library files are loaded after invocation with the read_cell_library command. The write_cell_library command writes the cells/models contained in those files to the new library file, *new_library.tcelllib*. If the *new_library.tcelllib* file already exists, its contents are replaced with the new library data.

```
$ tessent -shell
SETUP> read_cell_library pad_lib.tcelllib
SETUP> read_cell_library standard_lib.tcelllib
SETUP> read_cell_library my_lib.tcelllib
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
SETUP> write_cell_library new_library.tcelllib -rep
```

Related Topics

delete_cell_library

read_cell_library

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_config_data

Context: unspecified, all contexts

Mode: all modes

Writes the configuration data presently in memory into a file.

Usage

write_config_data ***filename*** [-wrappers *wrapper_spec*] [-specify_all] [-indent *integer*] [-replace] [-original_hierarchy] [-no_banner]

Description

When using the -wrappers option, the data written to the file can be limited to some specific wrappers.

Arguments

- ***filename***

  A required string option that specifies the name of the file into which to write the configuration data.

- -wrappers *wrapper_spec*

  An optional switch and value pair that specifies a Tcl list of one or more names of configuration wrappers, or a collection containing one or many configuration wrapper objects. Only the specified wrapper elements are written to the file.

  Just as with the report_config_data command, if *wrapper_spec* only contains a single wrapper of type CSV, the format of the file is changed such that a true CSV file is written out and ready to be read in to tools that support the CSV format such as Excel. This behavior is disabled if you use the -original_hierarchy option. A true CSV file has no wrapper and the line termination character is the carriage return instead of the semicolon, as is the case in a normal configuration data file.

- -specify_all

  An optional switch that specifies that unspecified properties and singular wrappers all show up in the written file with the unspecified properties specified to their default values. By default, unspecified properties and singular wrappers do not appear in the output file.

- -indent *integer*

  An optional switch and value pair that specifies to change the amount of white spaces used for indentation. By default, sub-elements are indented by two characters with respect to their parent wrapper.

- -replace

  An optional Boolean switch that suppresses the error that normally is issued if the filename already exists. When the -replace option is used, the tool silently deletes the existing file before creating the new one.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -original_hierarchy

  By default, the wrappers specified with the -wrappers options become root wrappers in the output file. You can use this option to preserve their original hierarchy and include all parent wrappers above them. When you write out a sub-wrapper without preserving the original hierarchy, you can then read it back into any valid parent wrapper using the "report_config_data -in_wrapper" command and option.

- -no_banner

  An optional Boolean option that suppresses the banner that is normally written at the top of the file. The banner format is the following:

```
//-------------------------------------------------
//  File created by: Tessent Shell
//          Version: 2014.1
//       Created on: Mon Jan 20 12:12:28 PST 2014
//-------------------------------------------------
```

  Notice that the banner is also removed when the file is output in the true CSV format. See the description of the -wrappers option for more info about the CSV format.

Examples

The following example shows the effect of all options.

#shows the configuration data currently in memory
report_config_data

```
tmp(1) {
  subWrapper1 {
    prop1 : v1;
  }
  subWrapper2 {
    prop1 : v1;
  }
}
```

 # Standard write to a file
write_config_data file1
cat file1

```
//-------------------------------------------------
//  File created by: Tessent Shell
//          Version: 2014.1-prerelease
//       Created on: Mon Jan 20 12:47:09 PST 2014
//-------------------------------------------------
tmp(1) {
  subWrapper1 {
    prop1 : v1;
  }
  subWrapper2 {
    prop1 : v1;
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

 # Limit write to one sub wrappers
write_config_data file2 -wrappers tmp(1)/subWrapper1
cat file2

```
//-------------------------------------------------
//  File created by: Tessent Shell
//          Version: 2014.1-prerelease
//       Created on: Mon Jan 20 12:47:09 PST 2014
//-------------------------------------------------
subWrapper1 {
  prop1 : v1;
}
```

 # Preserve the original hierarchy
write_config_data file3 -wrappers tmp(1)/subWrapper1 -original_hierarchy
cat file3

```
//-------------------------------------------------
//  File created by: Tessent Shell
//          Version: 2014.1-prerelease
//       Created on: Mon Jan 20 12:47:09 PST 2014
//-------------------------------------------------
tmp(1) {
  subWrapper1 {
    prop1 : v1;
  }
}
```

#Remove the banner. Show error when file exist and -replace is not used.
catch {write_config_data file3 -no_banner}

```
//  Error: File file3 cannot be overwritten unless you use the option -replace.
```

write_config_data file3 -no_banner -replacecat file3

```
tmp(1) {
  subWrapper1 {
    prop1 : v1;
  }
  subWrapper2 {
    prop1 : v1;
  }
}
```

Related Topics

report_config_data

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_core_description

Context: patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: Retargetable pattern generation is enabled in patterns -scan_retargeting context (set_current_mode -type internal).

Writes out the core description corresponding to the current chip level.

Usage

write_core_description ***filename*** [-replace]

Description

Writes out the core description corresponding to the current design.

The tool only writes out the current mode of the current core. It also writes out the core descriptions of any of its core instances so that the definition of the current core's current mode is self-contained.

The core description file can be generated in the following cases:

- After ATPG for the internal mode of a core, so that information can be fed forward for pattern retargeting.
- For the top level during pattern retargeting, so it can be used for reverse-mapping of chip-level failures back to the core level for diagnosis.
- For logicBIST in the patterns -ijtag context when generating chip-level patterns.

This command works in the -scan_retargeting context because this file is required as input for failure mapping.

Hybrid TK/LBIST Flow Specifics

Writes the description of the core for the logicBIST mode of operation. This includes description of the core pins, scan chain information, and EDT/logicBIST hardware registers (essentially, all information to capture the core-level logicBIST mode operation). Refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- ***filename***

  A required switch that specifies the pathname of the file to which the core description is written.

- -replace

  An optional switch that specifies to overwrite the existing file *filename*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The "Retargeting Example" in the*Tessent Scan and ATPG User's Manual* demonstrates the use of this command in a design context.

Related Topics

add_core_instances

delete_core_instances

read_core_descriptions

report_core_descriptions

report_core_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_core_timing_constraints

Context: dft -edt, patterns -scan

Mode: analysis

Creates timing (SDC) files that provide the constraints related to the scan path and ATPG settings in the core design.

Usage

write_core_timing_constraints ***filename_prefix*** [-Replace]

Description

Creates timing (SDC) files that provide the constraints related to the scan path and ATPG settings in the core design.

If launch-off capture (broadside) transition patterns are used, fast capture constraints are generated, otherwise slow capture constraints are generated.

For more information on the timing files, see "SDC Timing File Generation" in the *Tessent TestKompress User's Manual*.

Arguments

- ***filename_prefix***

  A required string that specifies the naming prefix for the timing files.

- -Replace

  An optional switch that replaces the contents of the file if the *filename* already exists.

Examples

The following example outputs timing files for the scan path and ATPG settings in the core design, using "atpgm" as the filename prefix:

write_core_timing_constraints atpgm

```
//  Writing atpgm_core_shift_sdc.tcl
//  Writing atpgm_slow_capture_sdc.tcl
```

Related Topics

write_edt_files

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_design

Context: all contexts

Mode: all modes

Writes the current design in Verilog netlist format to the specified file.

Usage

Usage 1: Creating interface modules

write_design -output_file *filename* [-banner *banner*] [-modules *modules_list*]
-interface [-verbose] [-replace]

Usage 2: Writing out created modules in rtl context

write_design -output_file *filename* [-banner *banner*] [-created | -modules *modules_list*
[-hierarchical]] [-exclude_modules *exclude_modules*][-verbose] [-replace]

Usage 3: Writing out read-in modules in rtl context

write_design {-output_directory *directory* | -original_location [-extension_suffix *suffix*]}
[-modified | -all_read | -modules *modules_list* [-hierarchical]]
[-exclude_modules *exclude_modules*] [-write_out_updated_f_files] [-verbose]
[-file_map *file_map_var*]
[-mark_edited_macro_regions]
[-replace]

Usage 4: Writing out read-in or created modules in no_rtl context

write_design -output_file *filename* [-banner *banner*]
[-modified | -created | -modules *modules_list*
[-hierarchical] [-exclude_modules *exclude_modules*] [-verbose] [-replace]

Usage 5: Writing out the graybox view of the current design

write_design -output_file *filename* [-banner *banner*] [-graybox]
[-file_dictionary dictionary_name] [-verbose] [-replace]

Usage 6: Writing out the modified design in the dft_inserted_designs directory of the TSDB

write_design -tsdb [-create_ijtag_graybox {on | off}]
[-mark_edited_macro_regions] [-verbose]

Usage 7: Writing out the graybox view of the current design in the dft_inserted_designs directory

write_design -tsdb [-graybox]
[-variant_scan_graybox_id *variant_scan_graybox_id*]
[-file_dictionary dictionary_name] [-verbose]

Usage 8: Creating a dft_inserted_designs directory in the TSDB with a softlink to the read-in concatenated netlist

write_design -tsdb -softlink_netlist [-create_ijtag_graybox {on | off}] [-verbose]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Usage 9: Creating a simulation wrapper

write_design -simulation_wrapper -output_file *filename*
[-verbose] [-replace]

Description

Writes the current design in Verilog netlist format to the specified file.

Use the -modules switch to write only specific modules. The default is to write the top module with all the modules in the hierarchy below except the created modules and the modules below them. The file organization of the read in files is never changed when using the -rtl mode of the set_context command. See the section "Difference Between -rtl and -no_rtl Mode" for the difference between -rtl and -no_rtl mode when you write out the design using the write_design command.

Note

You cannot use the write_design command to overwrite files containing RTL modules that were read in with the read_verilog command. See the section "Difference Between -rtl and -no_rtl Mode" below for an explanation.

Note

The write_design command is disabled if you read in a flat model using the read_flat_model command.

Note

Verilog modules originally defined inside `celldefine and `endcelldefine directives are not included in the output netlist. For more information about using `celldefine to define cell boundaries, refer to the "Fault Locations" discussion in the *Tessent Scan and ATPG User's Manual*.

Usage 1 — This usage case writes out the interface definition of the specified modules only. The port and parameter definitions are saved but everything else is deleted from the module. This has no impact on the module definition in memory and only affects the content of the file being written out. If you do not specify the -modules option, the interface of the current design is written out.

Usage 2 — This usage case writes out the created modules in the rtl context. Created modules are modules created using the create_module command. These modules do not have a source file associated with them, so you must specify one using the -output_file option. You can write all created modules into a single file using the -created option. This only works if all of your created modules have a common language because modules from different languages (VHDL versus Verilog) must be written into different files.

If you do not specify either -modules or -created options, the tool infers the following: "-modules [get_current_design] -hierarchical." In this case, the tool generates an error if the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

current design is a created module and you did not specify the -output_file switch. When you specify the -modules option, you can only specify created modules; otherwise, an error is generated.

When you specify the -hierarchical option, read-in modules are automatically skipped.

Usage 3 — This usage case writes out read-in modules in the rtl context. Because RTL can contain many directives that affect how things are later compiled, the tool preserves the content and ordering of the RTL files when writing them out. For this reason, you can only write out read-in modules using either the -output_directory or the -original_location switch. You cannot use the -output_file option for read-in modules in the rtl context.

By default, the tool does not write out the read-in -f files. You must specify the optional -write_out_updated_f_files switch to write out the -f files in addition to the read-in RTL files.

You can use the -all_read option to decompile and write all files previously read with read_verilog or read_vhdl, even if they contain no module definitions, such as files containing only the definitions of System Verilog interfaces, `define statements, or files containing no modules instantiated below the current design.

You can use the -modified option to quickly write out only the modified modules, but if you also specify created modules, the tool does not write those out. However, you can write out the created modules using a separate write_design command invocation with the -output_file option specified.

If the -modules or -modified options are not specified, the tool infers the following: "-modules [get_current_design] -hierarchical." In this case, the tool generates an error if the current design is a created module and you used the -output_directory or the -original_location switches. When you specify the -modules option, you can only specify read-in modules in this usage.

For more explanation on how read-in modules are written out in the RTL context, see the "Difference Between -rtl and -no_rtl Mode" section below.

Note

Use the -mark_edited_macro_regions switch to flag changes in the design files with comments. This enables you to more easily identify changes made by the tool. Inactive regions also have markers to identify them as inactive.

Usage 4 — This usage case writes out read-in or created modules in the no_rtl context. In the no_rtl context, the design modules are written out with all compiler directives removed. The dependencies on the file order is thus removed, allowing the concatenation of any modules into common files. You can merge both read-in and created modules into the same file.

In the no_rtl context, the recommended use model is to write out the entire design hierarchically into a single file, which is the default behavior when the -modules, -created or -modified option are not specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Usage 5 — This usage case writes out the graybox view of the current design into a single file. Only those instances and nets that have the preserve_in_graybox attribute set to true are kept in this file. All other nets and instances are removed. You can set this attribute using the set_attribute_value command, but this attribute is normally populated using the analyze_graybox command. See Usage 7 below for how to save the graybox model into the TSDB.

In the ijtag context, only the no RTL view is available. In the ijtag -rtl context, you create the IJTAG graybox view using the extract_icl command by specifying the "-create_ijtag_graybox on" switch.

Note

If you previously ran the "analyze_graybox -scan" command in the patterns -scan context, "write_design -graybox" generates a scan graybox. If you ran the "analyze_graybox -ijtag" command in the patterns -ijtag -no_rtl context, "write_design -graybox" generates an IJTAG graybox.

Usage 6 — This usage case creates a dft_inserted_designs directory with the modified design that you created using a custom script. If ICL is elaborated, it is written out to the new *dft_inserted_designs* directory with the tessent_design_id attribute updated to match the new *design_id* defined with the set_context command. The concatenated PDL file is also copied and the *.tcd* file and the *.tsdb_info* files are also created. Use this case when you want to run a custom editing script. Saving it using "write_design -tsdb" stores it inside the TSDB such that you can read it back later using the read_design command. When you specify the "-create_ijtag_graybox on" switch, an IJTAG graybox view is also created.

Note

Use the -mark_edited_macro_regions switch to flag changes in the design files with comments. This enables you to more easily identify changes made by the tool. Inactive regions also have markers to identify them as inactive.

Usage 7 — This usage case writes out the graybox view in the dft_inserted_designs directory. When using the -graybox switch, an error is generated if the analyze_graybox command was not called prior. The graybox view can be saved in a *dft_inserted_designs* directory that was previously created using the "write_design -tsdb" or "insert_test_logic -write_in_tsdb on" command. If the current "set_context -design_identifier" value does not match an existing dft_inserted_designs directory, a new one is created with all the associated files: the ICL, PDL, TCD, and the TSDB info files.

Usage 8 — The -softlink_netlist usage case is only usable in the -no_rtl mode right after the design was elaborated in the setup mode. It creates a dft_inserted_designs directory for the read-in netlist such that it can be easily loaded with read_design or by the run_testbench_simulations commands. This option is typically used to create a TSDB entry for the post-synthesis netlist when scan insertion is not done in Tessent Shell. The post-synthesis TSDB view is created by "insert_test_logic -write_in_tsdb on" when the scan insertion is done in Tessent Shell. It is also

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

used to create a TSDB entry for the post layout netlist such that the read_design command can be used in ATPG to load this netlist, create a graybox view when it is a wrapped core, and automate the design loading in run_testbench_simulations. When you specify the "-create_ijtag_graybox on" switch, an IJTAG graybox view is also created for the netlist.

Usage 9 — This usage case enables you to write the simulation wrapper into the file specified via the -output_file switch. See "Simulation Wrapper Creation" in the *Tessent Shell User's Manual* for more information.

Note

Manual generation of the simulation wrapper is not required in most flows.

Difference Between -rtl and -no_rtl Mode

There are many differences that exist between RTL and netlist files. Those differences are listed and contrasted in Table 9-2. Because of the characteristics of the RTL files mentioned in the table, the module organization of RTL files cannot be changed.

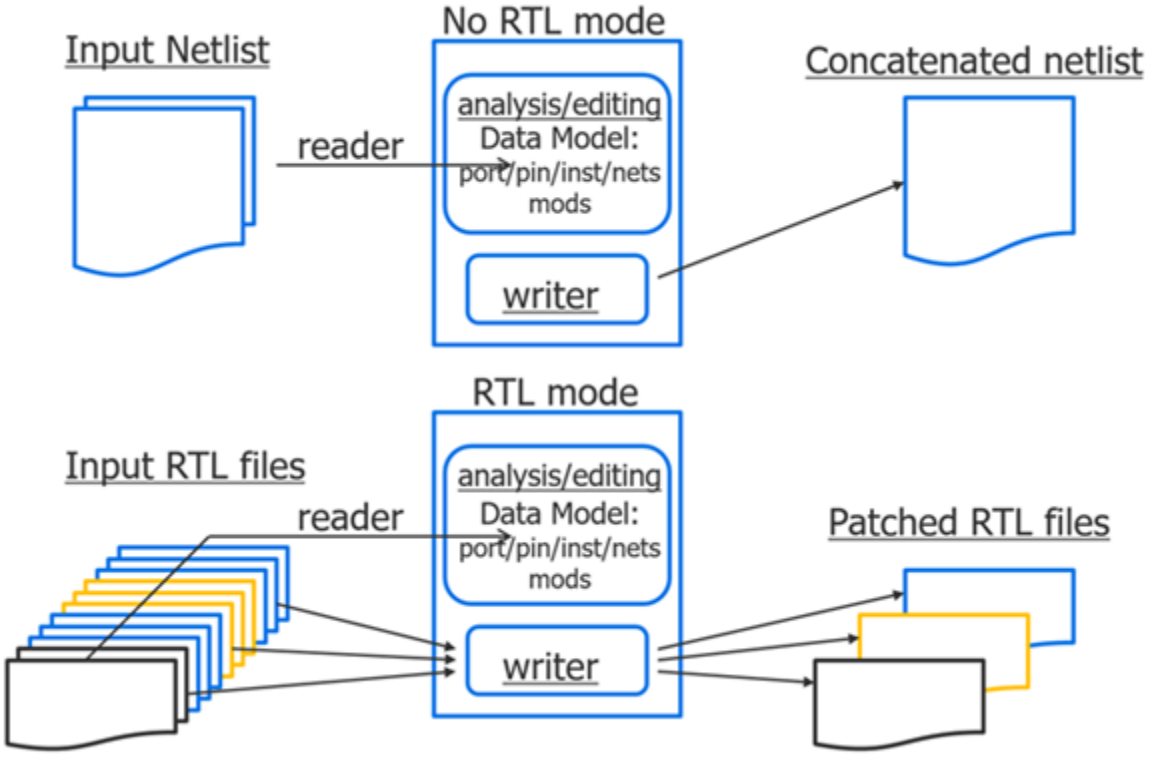
Figure 9-4 shows the differences that exist inside the tool when reading RTL vs netlist files. In both cases, the data model inside the tool only consists of the structural elements found inside the modules, instances, ports, pins, and nets. The different color used for the input files of the rtl mode illustrate the various languages and formats the files may have in rtl mode. In the netlist mode, the files are always with language Verilog and format 2001. When it comes time to write out the modified netlist, all the information is in the data model and a new netlist is reconstructed from scratch using the information inside the tool. In the rtl mode, only the edits made to the structural objects (module, instances, pin, port, net) are present in the internal data model. The effects on the RTL files are only reflected when the write_design command is used. The original files containing the modified modules are used and the edits are patched in a surgical fashion such that all white spaces, comments, and carriage returns in the original files are preserved. Only the pins, ports, and nets affected by the edits show modifications. Because each file came in with a unique language and format, all written out files preserve the unique language and format, as illustrated with the coloring of the files. Figure 9-5 shows the effect of doing two insertion passes on the same design. Some files were never modified. Some files were only modified by the first insertion pass. Some files were only modified by the second insertion pass. Finally, some files were modified by both insertion passes. The design view that represents the design as it exists after the second insertion pass has files located in the original location, in the dft_inserted_designs directory of the first insertion pass and in the dft_inserted_designs directory of the second insertion pass. Refer to the read_design, run_testbench_simulations and the write_design_import_script command descriptions to see how the tool provides powerful commands to help reload the edited RTL design files.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 9-2. Difference Between RTL and Netlist Files

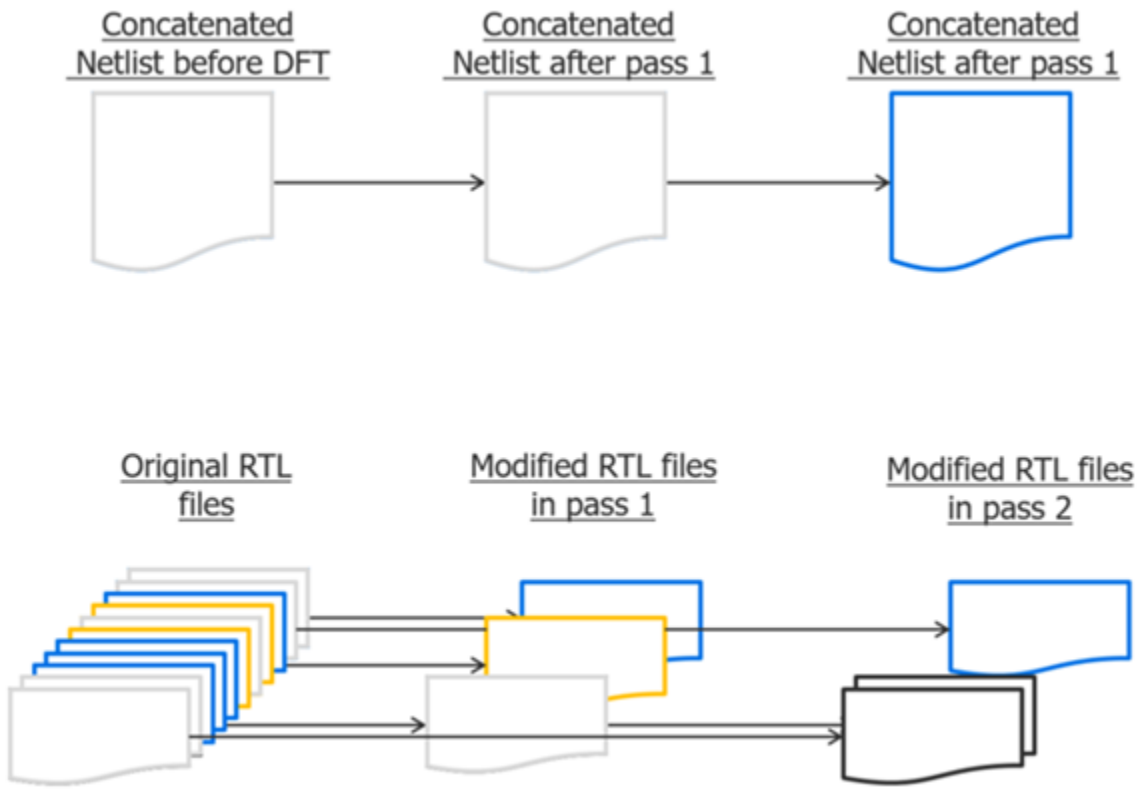
RTL	Netlist
9 distinct formats, no superset parser Verilog (1995 | 2001 | sv31a | sv2005 | sv2009) VHDL (1987 | 1993 | 2002 | 2008)	1 format Verilog 2001
Content of read-in file affects the files read afterward. Ex :`define, packages, SV definitions,...	Read order has no effect
File structure and read order must be preserved one format per file read order cannot be changed	Support concatenating all modules into one file
Typically one file per module or functional block manually edited and maintained revision control system like CVS or Git	One file per physical block
One module name can have multiple implementations parameters, generate blocks, logical libraries, ...	One module name, one implementation

Figure 9-4. Writing Out the Modified Design in rtl vs no_rtl Modes



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 9-5. Effect of Successive Design Editing Passes on the Design View



Logic Synthesis and Port Name Mapping

Logic synthesis can flatten or bit-blast ports, causing post-synthesis names to differ from those in the initial RTL. This process is automatic, and typically does not require user intervention. For more information, refer to "Automatic Design Mapping" in the *Tessent Shell User's Manual*.

Arguments

- -all_read

  An optional switch that decompiles and writes out all files previously read with read_verilog or read_vhdl, even if they contain no modules. This switch is available only in the rtl context.

- -banner *banner*

  An optional switch and string pair that specifies a header comment to be written as the first line of the output file.

- -created

  An optional switch that specifies to limit the written modules to only those created. Specifying this option is equivalent to using "-modules [get_modules -filter is_created]."

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -create_ijtag_graybox on | off

  An optional switch and literal pair available in Usages 6 and 8. It internally invokes the "analyze_graybox -ijtag" command followed by "write_design -output_file -graybox" to create an IJTAG graybox view. This view is used during pattern generation and simulations, as explained in Tessent SSN Workflows of the *Tessent Shell User's Manual*. See the create_patterns_specification and process_patterns_specification command reference pages for more information on how to use the IJTAG graybox view.

- -exclude_modules *exclude_modules*

  An optional switch and value pair that excludes modules from being written out. This argument is useful only when the -hierarchical option is inferred or specified. All modules below the excluded modules are also excluded if they are not specified in the -modules list or found below other non-excluded modules.

  In the -rtl context, when using multiple logical libraries and VHDL architectures, use the get_modules command to select modules from specific libraries or architectures. The tool only looks for a module name in the module_list in the default library with the default architecture.

  In the -rtl context, the tool always includes all modules in the output file even if the -exclude_modules switch excludes some of those modules. When one module in the file is in the -modules switch, the tool writes out all modules.

  In the -no_rtl context, you can control which modules to include in the output file. See the "Difference Between -rtl and -no_rtl Mode" section for more details.

- -extension_suffix *suffix*

  An optional switch and string pair that specifies the suffix to use to distinguish the written-out filenames from the read-in filenames when the -original_location option is used. The -extension_suffix defaults to "_ts."

- -file_dictionary *dictionary_name*

  An optional switch and string pair that specifies a Tcl variable name containing a Tcl dictionary. This dictionary contains information about the design files that were written out. When you restore an IJTAG graybox generated from an RTL design, you must load this dictionary. For a description of the file dictionary format, see the description of the *<design_name>.design_source_dictionary* component in dft_inserted_designs.

- -file_map *file_map_var*

  An optional switch and string pair that returns a list of modified files compared with the original design files during the design writing operation. The format is a Tcl list of lists with two entries per list; the first entry is the original design file path, and the second entry is the modified design file path. Use this switch when doing flows that require the generation and use of context views for designs that need uniquification. The *file_map_var* shows which files were modified, enabling you to update your design load script.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -graybox

  An optional switch and string pair that specifies that the content of the output files does not contain the complete module descriptions but only those nets and instances having the in_graybox attribute set to true (by the analyze_graybox command).

  Note

  If you previously issued the "analyze_graybox -scan" command in the patterns -scan context, "write_design -graybox" generates a scan graybox. If you issued the "analyze_graybox -ijtag" command in the patterns -ijtag context, "write_design -graybox" generates an IJTAG graybox.

  When writing the IJTAG graybox view in RTL context, the system mode is automatically switched back to setup mode in order to restore the full view of the design. You are notified with this message:

  ```
  //  Note: A transition back to setup mode has been performed. This
  was done in order to restore
  //       the internal circuit representation, which was affected by
  writing out the IJTAG graybox.
  ```

- -hierarchical

  An optional switch that writes out all child modules below the specified modules. This switch can only be used with the -modules switch. If the -modules switch includes created modules, read-in modules found below those modules are ignored. If the -modules switch includes read-in modules, created modules found below those modules are ignored. An error is generated of the -modules switch contains both read-in and created modules as those consist of different usage. See Usage 2 and 3 above for an explanation of the difference. See the "Difference Between -rtl and -no_rtl Mode" section for more details.

- -interface

  An optional switch that specifies to write out only the interface of the current design or the interfaces of the modules specified by the -modules option.

- -mark_edited_macro_regions

  An optional switch that adds comments to design files that were changed by the tool. See Usage 3 and 6 above for more details.

- -simulation_wrapper

  An optional switch that specifies that the simulation wrapper is written if "get_current_design -needs_simulation_wrapper" is true. See "Simulation Wrapper Creation" in the *Tessent Shell User's Manual* for more information.

- -modules *module_list*

  An optional switch and value pair that specifies the names of the modules to write out. The module_list value is a Tcl list of module names, or a collection of modules as returned by the get_modules command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In the rtl context when using multiple logical libraries and vhdl architectures, use the get_modules commands to select modules from a specific library or architecture. A module name in the module_list is only looked for in the default library with the default architecture.

In the rtl context, you cannot use the -output_file switch when writing out read-in modules. The written out file always contain the exact same modules list that existed in the read-in file. If you use the -modules switch to refer to a module, the source file that contains the read-in module is written out containing all modules that existed when it was read in. See the "Difference Between -rtl and -no_rtl Mode" section for an explanation of why the file structure cannot be modified for read-in files in rtl mode.

- -original_location

  A switch that specifies to write out the specified modules in the same directory that they were read in from. All files containing at least one of the specified modules in the -modules switch value are written out. The filename and directory location of the source file are used and the default or specified -extension_suffix value is appended to the file extension.

  This option preserves the same file and directory structure as the original RTL description. The modified RTL files can be loaded into the next tool using the original -y options by simply modifying the extension search list. You must, however, have write access in the source directories. The source file can be write protected but not the source directory.

  The commands process_dft_specification, "insert_test_logic -write_in_tsdb on", and "write_design -tsdb" all create a "dft_inserted_designs" TSDB container for the new design view. The modified RTL files are stored inside that directory and the *.design_source_dictionary* file contains the information to load that design view within the read_design, write_design_import_script, and the run_testbench_simulations commands. If you want the modified RTL files to be stored back beside the original file locations as it was done in the LV Flow, then issue the following command before leaving the insertion mode (for example, after having called the process_dft_specification command but before calling extract_icl):

  ```
  write_design -modified -original_location -extension_suffix _et
  ```

  The file dictionary is written into the TSDB when exiting the insertion pointing to the modified RTL files written back into their original location.

- -output_directory *directory*

  A switch and string pair that specifies the directory in which to write all specified modules. All files containing at least one of the specified modules is written out. Subdirectories within the specified output directory are created if the read-in files were stored in separate directories in order to avoid filename collisions. You can only use this switch in -rtl mode, and defines Usage 3 described above. See the "Difference Between -rtl and -no_rtl Mode" section for an explanation why the file structure is not modified when using the rtl context.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -output_file *filename*

  A switch and string pair that specifies the name of the file in which to write all specified modules. This switch can only be used in the rtl context when writing created modules as described in Usage 2 above. It can be used for any modules in the no_rtl context.

- -replace

  An optional switch that specifies to overwrite an existing file. When you use this switch, the error message that is normally generated if the file being written out already exists is suppressed.

- -softlink_netlist

  An optional switch that is only usable in the -no_rtl mode right after the design was elaborated in the setup mode. It must be used with the -tsdb switch to create a *dft_inserted_designs* directory containing a softlink to the read-in netlist, such that it can be easily loaded with read_design or by the run_testbench_simulations command. For more information about this switch, see Usage 8 described previously.

- -tsdb

  An optional switch that specifies to write the design and all its associated files in the Tessent Shell Data Base (TSDB). See the Usage 6, 7, and 8 above for more details. When the tool creates a new DFT-inserted design subdirectory under the current TSDB output directory, all the ICL variants present in the source TSDB are updated and written alongside the main ICL file. The tool also updates the main and variant IJTAG graybox views. This is particularly useful when many ICL variants were created in RTL mode before real synthesis of the design using the extract_icl command. The synthesized netlist is read together with the ICL from the last RTL insertion pass, and "write_design -tsdb [-softlink_netlist]" results in a post-synthesis TSDB DFT-inserted design that is complete.

- -variant_scan_graybox_id *variant_scan_graybox_id*

  An optional switch and value pair to associate a *variant_scan_graybox_id* with a scan graybox model. You can only use this switch in combination with the -graybox switch in the patterns -scan context. Typically, you need to create several scan grayboxes and differentiate them with a *variant_scan_graybox_id* if there are several external scan modes in a given design. You can then load the correct view using the "read_design -view scan_graybox -variant_scan_graybox_id *variant_scan_graybox_id*" command.

- -verbose

  An optional switch that specifies to print the name of each written file, one per line.

- -write_out_updated_f_files

  An optional switch to add the read-in -f files to the written RTL files. The -f files are modified to point to the new location of the written directories and files. This may change the directory structure of the written files if the -f files were located higher in the directory tree than the RTL files.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example writes out the current design and every other design module instantiated below it into a common file.

```
write_design -output_file design_out.v
```

Example 2

The following example generates an interface module description for the current design inside the file called *mod.shell*.

```
write_design -output_file mod.shell -interface
```

Example 3

The following example writes out all modules within the current design but stops on CoreA and CoreB. Separate commands are used to write out all modules within CoreA and CoreB. Note how the first command does not specify the -modules option, which means the current design is to be written out and the -hierarchical switch is inferred.

```
write_design -output_file myTop.v -exclude_modules {CoreA CoreB}
write_design -output_file CoreA.v -modules CoreA -hierarchical
write_design -output_file CoreB.v -modules CoreB -hierarchical
```

Example 4

The following example uses the -no_hdl option of the read_design command to load the information about *my_design* from the pre-synthesis TSDB directory and combine it with the post-synthesis netlist to create a complete gate-level TSDB directory. This enables the use of "read_design *my_design* -design gate" to create patterns.

```
set_context dft -design_id gate -no_rtl
read_cell_library cell_library
read_verilog path/my_design.vg
read_design my_design -design_id rtl -no_hdl -verbose
set_current_design my_design
write_design -tsdb -softlink_netlist -verbose
```

Refer to Example 2 in run_testbench_simulations for a usage example on how to use the write_design -tsdb -softlink_netlist command to create a TSDB dft_inserted_designs container for a netlist generated from synthesis or layout, and on which you want to use the run_testbench_simulations command to simulate the patterns described in your PatternsSpecification.

Related Topics

delete_design

set_current_design

read_verilog

read_vhdl

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_design_import_script

Context: dft, patterns

Mode: setup, analysis

Generates a script that can be processed by a synthesis tool to synthesize an RTL design that has been DFT inserted.

Usage

write_design_import_script [*filename*] [-replace] [-module *module_spec*]
[-design_id *design_id*]
[-format {dc_shell | genus}]
[-use_relative_path_to *directory*]
[-exclude_sub_blocks] [-silent]

Description

This command generates a script that can be processed by a synthesis tool to synthesize an RTL design that has been DFT inserted. The script analyzes all the source files that were read into Tessent Shell and reported into the *design_name.design_source_dictionary* for the current design that is generated into the Tessent Shell Data Base (TSDB). By default, all the sub-blocks of the design are also analyzed by the script, unless the option -exclude_sub_blocks is specified. When the *design_name.design_source_dictionary* has not been generated for the current design, the generated script uses the information in memory about the source files that were read into Tessent Shell when the command is issued.

Arguments

- *filename*

  An optional string that specifies the script *filename*. When the *filename* is a path to an existing subdirectory, the file is created in that subdirectory with the default name and the "-use_relative_path_to" value is assumed to be the subdirectory specified for *filename*. When *filename* is not specified or does not exist, the name defaults to:

  ```
  design_name.tool_name_import_script
  ```

- -replace

  An optional switch that specifies overwriting *filename* when the file exists.

- -module *module_spec*

  An optional switch and string pair that specifies a specific module name in an opened TSDB that has been DFT inserted. The default is to use the current design and current design identifier.

- -design_id *design_id*

  An optional switch and value pair that specifies a specific *design_id* when a module has been DFT inserted more than once.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -format dc_shell | genus

  An optional switch and value pair to specify which synthesis tool consumes the generated file content. The default value is “dc_shell”.

- -use_relative_path_to *directory*

  An optional switch and value pair that specifies a directory from which the generated script is processed. The files being referenced are relative to this directory. By default, the generated script includes the full paths to the files to process.

- -exclude_sub_blocks

  An optional switch that specifies to ignore the analysis of the sub-blocks instantiated in the main module in the generated script. By default and if this switch is not specified, the generated script considers the design source dictionary for the selected DFT inserted design and all the sub-blocks that are included in the selected design. The inclusion stops and does not include any module that is a physical block. All the TSDBs used to assemble the selected module must be opened in order for the sub-block information to be accessible.

- -silent

  An optional switch that suppresses informational messages usually displayed during the processing of the command.

Examples

Example 1

The following produces the report for the current design in memory, with full paths and including all the sub-blocks. The resulting script is displayed when in interactive mode and is reported in the Tessent Shell log file.

```
write_design_import_script
```

Example 2

The following produces the report for the design *my_core* with paths relative to the subdirectory *synth*/*my_core* and without the sub-blocks. The generated script is saved in the file *synth*/*my_core*/*analyze.tcl*.

```
file mkdir synth/my_core
write_design_import_script analyze.tcl -replace -module my_core \
    -exclude_sub_blocks -use_relative_path_to ./synth/my_core
```

Related Topics

run_synthesis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_design_source_dictionary

Context: dft, patterns

Mode: setup, analysis, insertion

Writes a design-source dictionary into a file. This file contains the file and directory paths relative to the output file location.

Usage

write_design_source_dictionary ***output_file*** [-dictionary *dictionary*] [-replace]

Description

Writes a design-source dictionary into a file. This file contains the file and directory paths relative to the output file location. You can subsequently source this written-out design-source dictionary to reload the design using the "read_design -from_dictionary $*design_source_dictionary*" command. Refer to the example to see how you use this command to package an RTL test case into a single directory. If you omit the -dictionary switch, then the design-source dictionary returned by the "get_design_sources -file_dictionary" command is written out. You can also choose to store the design-source dictionary returned by the "get_design_sources -file_dictionary" command into a variable, make edits to the dictionary stored into the variable using the Tcl "dict" command, and then write out the modified dictionary using the "-dictionary $var" switch, where "var" is the Tcl variable that holds the dictionary you modified.

Note

If you want to write out a design-source dictionary that reflects the state of the RTL design after you have edited the design in insertion mode, then you must first go to setup mode to synchronize the design in order for the file dictionary to reflect the files with your edits.

Arguments

- ***output_file***

  A required string argument that specifies the name of the file into which the tool writes the design-source dictionary. All of the file and directory paths contained in the design-source dictionary file are relative to the location of the output file.

- -dictionary *dictionary*

  An optional switch-string pair that specifies a Tcl dictionary the tool writes out. Typically, you do not use this switch. When unspecified, the tool writes the design-source dictionary that is returned by the "get_design_sources -file_dictionary" command. You can choose to store the design-source dictionary returned by the "get_design_sources -file_dictionary" command into a variable, make edits to the dictionary stored into the variable using the Tcl "dict" command, and then write out the modified dictionary using the -dictionary $var switch, where "var" is the Tcl variable that holds the dictionary you modified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -replace

  An optional Boolean switch that suppresses the error the tool normally issues when the specified ***output_file*** already exists. Instead of issuing an error, the tool silently replaces the file.

Examples

The following example demonstrates writing out the complete RTL source files into a single directory with an associated design-source dictionary.

You can subsequently tar the dictionary file and the RTL directory as a test case. When the dictionary file and the RTL directory are untarred at the destination, the file dictionary is sourced into the tool using the read_design -from_dictionary $design_source_dictionary command, which reads the design from the RTL directory. Use the -skip_existence_checks switch to suppress the error that the tool issues if the design you are loading uses the -y and/or -v libraries, or +incdir specifications. The design-source dictionary contains paths to the original path as well as the new path found within the RTL directory:

```
# To store the complete RTL file, use those commands:
set current_design_name [get_single_name [get_current_design]]
write_design -output_directory ${current_design_name}_rtl_files -all_read
write_design_source_dictionary ${current_design_name}.design_source_dictionary

# Tar up the file and the directory and untar it at the destination
# To reload the design, use those commands:set current_design_name my_design
source ${current_design_name}.design_source_dictionary
read_design -from_dictionary $design_source_dictionary -skip_existence_checks
set_current_design $current_design_name
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_diagnosis

Context: patterns -scan_diagnosis

Mode: analysis

Writes diagnosis reports in text, comma separated values (CSV), and layout (physical) marker location formats.

Usage

write_diagnosis [-SHORT] [-ENCoded] [-FORmat {TEXT | LAYout_marker | CSV}...] [-FILE *name* [-REPlace]] [-XMAP {V2LVS | CELL_HIERarchy}...]

Description

If you specify write_diagnosis with no arguments, Tessent Diagnosis creates an ASCII text diagnosis report and names the file using the *failure_filename* from the diagnose_failures command and puts the results file where *failure_filename* is located.

Arguments

- -SHORT

  An optional switch that changes the diagnosis report from the layout-aware diagnosis report format to the traditional diagnosis report for backward compatibility with other tools for reading in the diagnosis report.

  Using this option does not change the computed diagnosis result including layout-aware diagnosis information, but rather suppresses the display of the layout-aware information.

  Note

  This option performs the layout-aware diagnosis, but suppresses the display of this information.

- -ENCoded

  An optional switch specifying encoding a suspect's pin pathname, net name, and cell name in the diagnosis output file.

- -FORmat {TEXT | LAYout_marker | CSV}

  An optional switch and single or multiple literals specifying the report's output format. By default, the write_diagnosis command creates an ASCII text version of the report if you specify no arguments.

  Note

  To produce the location tables (XMAP tables), you must select the default ASCII text report format (-format TEXT) for the write_diagnosis command.

  If you use the -FORmat switch, then you must specify at least one of the following options:

  - TEXT — Specifies ASCII text format.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- LAYout_marker — Specifies layout coordinate format.
- CSV — Specifies comma separated value (CSV) format.

You can specify any combination of these literals. For example, the following syntax creates both text (*example.diag*) and CSV (*example.csv*) reports:

```
write_diagnosis -format text csv -file example
```

- -FILE *name*

  An optional switch and string pair specifying the filename (*name*) of the report. The tool uses the *name* value for the report filename, regardless of format, and identifies the format using the following filename suffixes:

  - *name*.csv — CSV format diagnosis report.
  - *name*.diag — ASCII text format diagnosis report.
  - *name*.lay — Layout markers.
  - *name*.enc.csv — Encoded CSV diagnosis report.
  - *name*.enc.diag — Encoded ASCII text diagnosis report.
  - *name*.enc.lay — Encoded layout markers.

- -REPlace

  An optional switch for overwriting an existing file.

- -XMAP {V2LVS | CELL_HIERarchy}

  An optional switch and single or multiple literals specifying the outputting of SPICE-mapped nets, Verilog module hierarchy, or both in the diagnosis report.

  If you use the -XMAP switch, then you must specify at least one of the following options:

  - V2LVS — Specifies outputting the SPICE-mapped nets.
  - CELL_HIERarchy — Specifies outputting Verilog module hierarchy.

  You can specify any combination of these switches. For example, the following syntax creates both the SPICE-mapped nets and the Verilog module hierarchy in the diagnosis report:

```
write_diagnosis -XMAP V2LVS CELL_HIERarchy
```

Examples

Example 1

The following example creates a CSV and ASCII text diagnosis report with the filename *foo* in the *bar* directory, which must exist:

write_diagnosis -format text csv -file bar/foo

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If you list the *bar* directory, then the following information is displayed:

```
% ls -l bar

...
-rwxrwxr-x   1 user account 1339 Aug 10 09:53 foo.csv
-rwxrwxr-x   1 user account 1768 Aug 10 09:53 foo.diag
```

Example 2

The following example creates encoded CSV, ASCII text, and layout markers with the filename *foo* in the *bar* directory, which must exist:

write_diagnosis -encoded -format text csv layout_marker -file bar/foo -replace

If you list the *bar* directory, the following information is displays:

```
% ls -l bar

...
-rwxrwxr-x   1 user account  1241 Aug 10 09:53 foo.enc.csv
-rwxrwxr-x   1 user account  1502 Aug 10 09:53 foo.enc.diag
-rwxrwxr-x   1 user account 40495 Aug 10 09:53 foo.enc.lay
```

Related Topics

report_diagnosis

diagnose_failures

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_edt_files

Context: dft -edt

Mode: analysis

Creates the files that implement the EDT logic.

Usage

write_edt_files ***filename_prefix*** [-Replace]
[-VErilog | -VHdl [-SIgnal_type STD_Logic | STD_Ulogic]]
[-SYnthesis_script {DC_Tcl | DC_Shell | RTL_Compiler}]
[-NO_Timing_constraints | -Timing_constraints]
[-Nortl_prefix | -RTl_prefix *prefix_string*]
[-Single_module_per_file]
[-UNIque_bypass_lockup_clock]
[-EXClude_modules *list_of_modules*]
[-NO_SYNOpsys_pragma [*pragma_list*]]
[-NO_WRITE_Design]
[-IJtag {Off | data_ports}]
[-TSdb [-DESIGN_Id *design_id*]]

Description

For more information, see "Creation of the EDT Logic" in the *Tessent TestKompress User's Manual*. By default, if no other switches are specified, the following files are written:

- EDT logic RTL (**_edt.v*)
- Tessent Core Description (TCD) file (**_edt.tcd*)
- ICL file (*_edt.icl*)
- PDL file (*_edt.pdl*)
- Test procedure file (**_edt.testproc*)
- Dofile (**_edt.dofile*)
- Design Compiler (DC) synthesis script (**_dc_script.scr*)
- Bypass dofile (**_bypass.dofile*)
- Bypass test procedure file (**_bypass.testproc*)

The following additional files are written out for the external flow:

- Core top file (**_edt_top.v*)
- Core blackbox file (**_core_blackbox.v*)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the set_lpct_controller command is enabled and issued, then the tool writes out the following additional file:

- LPCT TCD file (*_*edt_lpct.tcd*)

When the EDT logic is configured for multiple compression configurations, a separate test procedure and dofile are written for each configuration. For more information, see "Defining Dual Compression Configurations" in the *Tessent TestKompress User's Manual*.

By default, the language of the netlist read in is the format used for the EDT logic.

If you execute this command and you are using the internal IP location flow with Tessent Shell EDT insertion, the tool writes out all EDT files, inserts EDT IP into the design, and automatically changes the system mode from analysis to insertion.

If you execute this command and you are using the external IP location flow, the tool writes out all EDT files including the wrapper that instantiates the core design and EDT logic and remains in analysis mode. You can explicitly go to insertion mode, at which time you can further edit the core design and save the modifications. However, saving the design in insertion mode is only necessary if you have modified the design in insertion mode.

If tcd_scan is available for certain Tessent IP containing scan segments, the tool generates and writes CTL scan information in the TCD. See "Default Generation of CTL Files" in "Scan" on page 4794.

Arguments

- ***filename_prefix***

  A required string that specifies a name prefix for the EDT logic files. The filenames are composed of this prefix followed by an underscore (_) and a descriptive file-specific suffix. You can include the pathname to an existing directory as a component of the ***filename_prefix,*** and the files are written to that location. This option is mutually exclusive with the use of the -tsdb option.

- -Replace

  An optional switch that replaces the contents of each file that already exists.

- -VErilog

  An optional switch that writes the EDT logic in Verilog format.

- -VHdl

  An optional switch that writes the EDT logic in VHDL 93 format.

- -SYnthesis_script DC_Tcl | DC_Shell | RTL_Compiler

  An optional switch and set of literals that specify to generate a synthesis script:

  DC_Tcl — The tool generates the DC synthesis script in TCL format.

  DC_Shell — The tool generates the DC synthesis script in DC_Shell language format.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

RTL_Compiler — The tool generates the RTL Compiler synthesis script in TCL format.

- -SIgnal_type STD_Logic | STD_Ulogic

  An optional switch and literal pair that specifies the type of signals and ports to use (std_logic or std_ulogic) when writing out the EDT logic in VHDL format.

  Note

  When writing out the top-level wrapper (**_edt_top.vhd*) and the black box description of the core (**_core_blackbox.vhd*), the std_logic and std_ulogic declaration of signals and ports in the original netlist are preserved.

- -Single_module_per_file

  An optional switch that writes the EDT logic into multiple files with one Verilog module (or VHDL entity/architecture pair) in each file. Use this switch only if your synthesis flow requires the RTL files to be organized this way.

  The number of files written depends on the EDT logic setup. When the EDT logic is located outside the core, the top-level wrapper file may be split up if any pins share channel outputs. The DC synthesis script is modified to read in each of the RTL files individually.

  The -Single_module_per_file switch causes the tool to use a different naming convention for the output files as described in Table 9-3.

Table 9-3. Single Module Per File Naming Convention

Old Name	New Name	Output File
<filename_prefix>_edt_top .v (or .vhd)	{<top_module_name> | <rtl_prefix>}_edt_top.v (or .vhd)	Top-level wrapper (external flow only)
<filename_prefix>_edt.v (or .vhd)	{<top_module_name> | <rtl_prefix>}_edt.v (or .vhd)	EDT circuitry
<filename_prefix>_core_bl ackbox.v (or .vhd)	<top_module_name>_bla ckbox.v (or .vhd)	Blackbox description of core (external flow only)

  The additional *single module* files are named as follows:

  - {<top_module_name> | <rtl_prefix>}*_edt_*<module_name>*.v* (or *.vhd*)
  - {<top_module_name> | <rtl_prefix>}*_edt_pinshare_logic.v* (or *.vhd*) (external flow only)

  The RTL prefix string is used when constructing the preceding names only if you include the -Rtl_prefix switch; otherwise it uses the top-module name. If the command's *filename_prefix* argument includes a pathname component, the tool still writes all files to that location even though the filename prefix may not appear in all filenames.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Nortl_prefix

  An optional switch that disables the default internal RTL naming feature.

- -RTl_prefix *prefix_string*

  An optional switch and string pair that specifies a *prefix_string* to prepend to the module and instance names in the EDT logic RTL. The specified *prefix_string* must follow the normal naming rules for Verilog or VHDL identifiers.

- -NO_Timing_constraints | -Timing_constraints

  An optional switch that determines if SDC timing files are output. By default, no timing files are output. When you specify -timing_constraints, the tool reports both the fast capture and slow capture constraints. For more information, see "SDC Timing File Generation" in the *Tessent TestKompress User's Manual*.

- -UNIque_bypass_lockup_clock

  An optional switch that individually routes the clock pin of all scan cells involved with bypass lockup cells to the EDT logic. The clock pins created on the EDT logic boundary are named *edt_bypass_clk<n>* instead of top-level clock names.

- -EXClude_modules *list_of_modules*

  An optional switch and string pair that specifies to exclude from the complete EDT logic-inserted netlist, those modules specified by the *list_of_modules* argument; the tool excludes the modules specified by *list_of_modules* and any sub-module used by one of the specified modules, unless the sub-module is also used by another module that is to be written out. The *list_of_modules* is a list of strings that are separated by spaces.

  In the internal IP location flow with DC-based insertion, the tool writes out a Synopsys Design Compiler script to make EDT connections and to write out the modified design. Without the -EXClude_modules switch, the netlist is complete and contains a description of all the modules present in the input netlist. This switch excludes from the netlist the descriptions of the modules specified by the *list_of_modules* argument.

  During pattern generation, you need to provide the complete view of the EDT-inserted design including the module descriptions you excluded from the IP-creation output netlist with this command.

  Note

  If any of the modules excluded by this switch need to be modified during IP insertion, a warning message is issued but the modified modules are written out.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -NO_SYNOpsys_pragma [*pragma_list*]

  An optional switch that specifies to not write out pragmas when the tool writes out the RTL description of EDT logic in either Verilog or VHDL. The *pragma_list* can contain any of the following:

  - **sync_set_reset** — By default, when synthesis targets Synopsys Design Compiler (-SYnthesis_script DC_Tcl | DC_Shell), a pragma is written indicating to the synthesis tool that it is a reset signal and should not be merged with other signals.
  - **synthesis_off** — This pragma appears around VHDL config statements.

  If you do not specify a pragma list, the writing of all of these pragmas is suppressed.

  Refer to "Example 8" on page 3068 for an example of VHDL code including "sync_set_reset". Refer to "Example 9" on page 3070 for an example of using the write_edt_files command to suppress these pragmas.

- -NO_WRITE_Design

  An optional switch that directs the tool to not write out an EDT-inserted netlist when this command is executed. In insertion mode, you can edit the netlist further and save the design. You must save the design using the write_design command, and you must name the file using the naming convention *<filename_prefix>_edt_top_rtl.v* which is referenced by the synthesis script.

- -IJtag Off | Data_ports

  An optional switch and value pair that directs the tool to use IJTAG in the generated dofiles to describe the static configuration inputs of the TestKompress IP. Use this option if you are using a dofile to pass IP information from the IP creation phase to the pattern generation phase. These static configuration inputs set, enable, or disable certain features of the TestKompress IP: edt bypass, single chain bypass, low power and edt configuration.

  Note

  If you are using the Tessent Core Description (TCD) EDT flow, then you do not need to specify this option. See "IJTAG and the EDT IP TCD Flow" in the *Tessent TestKompress User's Manual* for more information. By default, the tool performs the IJTAG mapping automatically and writes out the ICL and PDL files.

  For the TCD EDT flow, you can disable the IJTAG mapping using the following command:

  ```
  set_procedure_retargeting_options -ijtag off
  ```

  For details on how to use the IJTAG files for TestKompress ATPG, see section "Setting up EDT IP for IJTAG Integration" in the "IJTAG Features of ATPG in Tessent Shell" section of the *Tessent IJTAG User's Manual*.

- -TSdb [-DESIGN_Id *design_id*]]

  An optional and optional switch and string pair that specifies to write the EDT files into the instruments directory of the Tessent Shell Data Base (TSDB). The name of the TSDB

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

directory defaults to *tsdb_outdir* and is specified using the set_tsdb_output_directory command.

The optional -DESIGN_Id *design_id* switch and string pair specifies the *design_id* value when creating the TSDB directory. Refer to "Tessent Shell Data Base (TSDB)" on page 4807 for complete information. The default *design_id* is that specified by the set_context command, or the default of that command if "set_context -design_identifier" is not used. In general, it is not recommended to specify -DESIGN_id as part of write_edt_files since this option overrides the value used by other TSDB-related commands used in that context.

Exporting your instruments using this method during post-scan insertion also enables the run_synthesis command to synthesize the generated IP.

The EDT Tessent Core Descriptions are also written out. See "Tessent Core Description (TCD)" in the *Tessent TestKompress User's Manual* for more information. EDT TCDs created using this switch are automatically loaded into the tool and do not need to be explicitly read.

This option is mutually exclusive with the use of *filename_prefix*. See Example 10 for a usage example.

Examples

Example 1

The following external flow example writes the files for EDT logic including bypass circuitry, and uses the filename prefix, *my_design* to name the output files:

write_edt_files my_design -verilog -replace -nortl_prefix

```
// Writing my_design_edt.v
// Writing my_design_edt_top.v
// Writing my_design_core_blackbox.v
// Writing my_design_dc_script.scr
// Writing my_design_edt.dofile
// Writing my_design_edt.testproc
// Writing my_design_bypass.dofile
// Writing my_design_bypass.testproc
// Writing my_design_edt.icl
// Writing my_design_edt.pdl
// Writing my_design_edt.tcd
```

The module and instance names in the RTL generated by the preceding example all begin with the *edt_* prefix.

Example 2

The following internal flow example writes out EDT logic in multiple files, one Verilog module per file. The EDT logic is placed internal to the design core, and the top-module name in the design is core1:

write_edt_files results/created -single_module_per_file -verilog

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Writing results/core1_edt_decompressor.v
//  Writing results/core1_edt_spatial_compactor_8.v
//  Writing results/core1_edt_controlled_decoder_3_to_8.v
//  Writing results/core1_edt_compactor.v
//  Writing results/core1_edt_bypass_logic.v
//  Writing results/core1_edt.v
//  Writing results/core1_edt_mux_2_to_1.v
//  Writing results/created_dc_script.scr
//  Writing results/created_edt.dofile
//  Writing results/created_edt.testproc
//  Writing results/created_bypass.dofile
//  Writing results/created_bypass.testproc
//  Writing results/created_edt.icl
//  Writing results/created_edt.pdl
//  Writing results/created_edt.tcd

write_design -output_file results/created_new.v
```

Example 3

The following external flow example shows how the -Rtl_prefix switch affects the filenames when the EDT logic is placed external to the design core, the EDT logic is written in multiple files, and the design has a top module named mtm2:

write_edt_files multiple -single_module_per_file -verilog -rtl_prefix designX_blockY

```
//  Writing designX_blockY_edt_decompressor.v
//  Writing designX_blockY_edt_spatial_compactor_8.v
//  Writing designX_blockY_edt_controlled_decoder_3_to_8.v
//  Writing designX_blockY_edt_compactor.v
//  Writing designX_blockY_edt.v
//  Writing designX_blockY_edt_pinshare_logic.v
//  Writing designX_blockY_edt_top.v
//  Writing mtm2_blackbox.v
//  Writing multiple_dc_script.scr
//  Writing multiple_edt.dofile
//  Writing multiple_edt.testproc
//  Writing designX_blockY_edt.icl
//  Writing designX_blockY_edt.pdl
//  Writing designX_blockY_edt.tcd
```

Example 4

The following internal flow example shows the files created when the tool inserts EDT logic into the design core:

write_edt_files results/created -verilog -rep

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Writing results/created_edt.v
//  Writing results/created_dc_script.scr
//  Writing results/created_edt.dofile
//  Writing results/created_edt.testproc
//  Writing results/created_bypass.dofile
//  Writing results/created_bypass.testproc
//  Writing results/created_edt.icl
//  Writing results/created_edt.pdl
//  Writing results/created_edt.tcd
//  ---------------------------------------------------------------------
//  Begin EDT logic insertion
//  ---------------------------------------------------------------------

//  Inserting EDT logic in the design.
//  EDT logic insertion completed.
```

write_design -output_file results/created_new.v

The output file *created_edt_top_rtl.v* contains the EDT logic inserted design without the EDT logic description.

Example 5

The following external flow example writes out timing files in addition to the default EDT files and specifies created for the naming prefix:

write_edt_files created -timing_constraints

```
//  Writing created_edt.v
//  Writing created_edt_top.v
//  Writing created_core_blackbox.v
//  Writing created_dc_script.scr
//  Writing created_edt.dofile
//  Writing created_edt.testproc
//  Writing created_bypass.dofile
//  Writing created_bypass.testproc
//  Writing created_edt_shift_sdc.tcl
//  Writing created_bypass_shift_sdc.tcl
//  Writing created_edt_slow_capture_sdc.tcl
//  Writing created_edt.icl
//  Writing created_edt.pdl
//  Writing created_edt.tcd
```

Example 6

The following internal flow example shows the files created when the tool inserts EDT logic into the design core and the -synthesis_script switch is specified with the rtl_compiler option. The file resulting from the -synthesis_script rtl_compiler switch and option is in **bold**.

write_edt_files created -verilog -synthesis_script rtl_compiler -replace

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Writing created_edt.v
//  Writing created_rtlc_script.scr
//  Writing created_edt.dofile
//  Writing created_edt.testproc
//  Writing created_bypass.dofile
//  Writing created_bypass.testproc
//  Writing created_edt.icl
//  Writing created_edt.pdl
//  Writing created_edt.tcd

//  ---------------------------------------------------------------------
//  Begin EDT logic insertion
//  ---------------------------------------------------------------------
//  Inserting EDT logic in the design.
//  EDT logic insertion completed.
//  Note: The synthesis script assumes the modified design is saved
//  in file 'created_edt_top_rtl.v'.
//  Update the synthesis script when saving to a different file.
```

write_design -output_file created_new.v

Example 7

In this example, assume the top level of the design has the three modules cpu_core, clkblk, and padring instantiated. The following commands write out the top-level module of the design but exclude the cpu_core, clkblk, and padring modules. After executing these commands, the output netlist contains only the top module description with the instances of these three modules:

write_edt_files created -replace
write_design -output_file created_edt_top_rtl.v -exclude_modules {cpu_core clkblk padring}

Assume the input design for this example is contained in two files: *top.v*, which contains only the top-level module, and *chip.v*, which contains the other three modules. The following example invokes the tool, sets the context for IP creation, loads the Verilog files, and runs the dofile, which is assumed to include the above write_edt_files and write_design commands:

tessent -shell
SETUP> set_context dft -edt
SETUP> read_verilog top.v chip.v
SETUP> read_cell_library tessent_cell.tcellib
SETUP> set_current_design
SETUP> dofile ip_creation.do

After performing logic synthesis of the EDT logic, the output file is *created_edt_top_gate.v*. The command-line invocation for pattern generation uses the synthesized EDT logic file *created_edt_top_gate.v* as well as the file *chip.v* that contains the description of the modules excluded during IP creation.

tessent -shell
SETUP> set_context patterns -scan
SETUP> read_verilog created_edt_top_gate.v chip.v
SETUP> read_cell_library tessent_cell.tcellib
SETUP> set_current_design
SETUP> dofile pattern_gen.do

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 8

In this example, the write_edt_file command writes out RTL with synopsys pragma:

write_edt_files created -replace

In Verilog, the pragma is included as a comment near the always block that describes the target registers as shown here:

```
//-----------------------------------------------------------------------
//  File created by: Tessent Shell
//          Version: 2020.2
//       Created on: Mon Apr 20 13:10:26 PDT 2020
//
//       IP version: 7
//-----------------------------------------------------------------------

module m8051_edt_decompressor (
   input  wire        edt_clock,
   input  wire        edt_update,
   input  wire [ 1:0] edt_channels_in,
   output reg  [15:0] edt_scan_in
);
   reg    [11:0] lfsm_vec;
   reg    [11:0] lfsm_vec_lockup;
   // synopsys sync_set_reset edt_update
   always @(posedge edt_clock)
   begin : lfsm
      if (edt_update == 1'b1) begin
         lfsm_vec <= 12'b000000000000;
      end
      else begin
         lfsm_vec[ 0] <= lfsm_vec[ 1];
         lfsm_vec[ 1] <= lfsm_vec[ 2] ^ edt_channels_in[1];
         lfsm_vec[ 2] <= lfsm_vec[ 3];
         lfsm_vec[ 3] <= lfsm_vec[ 4] ^ edt_channels_in[1];
         lfsm_vec[ 4] <= lfsm_vec[ 5];
         lfsm_vec[ 5] <= lfsm_vec[ 6] ^ edt_channels_in[1];
         lfsm_vec[ 6] <= lfsm_vec[ 7] ^ lfsm_vec[ 3];
         lfsm_vec[ 7] <= lfsm_vec[ 8] ^ edt_channels_in[0];
         lfsm_vec[ 8] <= lfsm_vec[ 9] ^ lfsm_vec[ 4];
         lfsm_vec[ 9] <= lfsm_vec[10] ^ edt_channels_in[0];
         lfsm_vec[10] <= lfsm_vec[11] ^ lfsm_vec[ 1];
         lfsm_vec[11] <= lfsm_vec[ 0] ^ edt_channels_in[0];
      end
   end
   ...
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In VHDL, the pragma is specified as an attribute from the Synopsys library in the declarations section of the architecture description as shown here:

```
//-----------------------------------------------------------------------
//  File created by: Tessent Shell
//          Version: 2020.2
//       Created on: Mon Apr 20 13:10:26 PDT 2020
//
//       IP version: 7
//-----------------------------------------------------------------------

LIBRARY ieee;
LIBRARY synopsys;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE synopsys.attributes.all;
USE work.all;
ENTITY m8051_edt_decompressor IS
 PORT (
      edt_clock       : IN    std_logic;
      edt_update      : IN    std_logic;
      edt_channels_in : IN    std_logic_vector(1 DOWNTO 0);
      edt_scan_in     : OUT   std_logic_vector(15 DOWNTO 0)
   );
END m8051_edt_decompressor;
ARCHITECTURE rtl OF m8051_edt_decompressor IS
   SIGNAL lfsm_vec          : std_logic_vector(11 DOWNTO 0);
   SIGNAL lfsm_vec_lockup   : std_logic_vector(11 DOWNTO 0);
   ATTRIBUTE sync_set_reset of edt_update : SIGNAL is "true";
BEGIN
   lfsm: PROCESS (edt_clock)
   BEGIN
      IF (edt_clock = '1') AND (edt_clock'EVENT) THEN
         IF (edt_update = '1') THEN
            lfsm_vec <= (OTHERS => '0');
         ELSE
            lfsm_vec( 0) <= lfsm_vec( 1);
            lfsm_vec( 1) <= lfsm_vec( 2) XOR edt_channels_in(1);
            lfsm_vec( 2) <= lfsm_vec( 3);
            lfsm_vec( 3) <= lfsm_vec( 4) XOR edt_channels_in(1);
            lfsm_vec( 4) <= lfsm_vec( 5);
            lfsm_vec( 5) <= lfsm_vec( 6) XOR edt_channels_in(1);
            lfsm_vec( 6) <= lfsm_vec( 7) XOR lfsm_vec( 3);
            lfsm_vec( 7) <= lfsm_vec( 8) XOR edt_channels_in(0);
            lfsm_vec( 8) <= lfsm_vec( 9) XOR lfsm_vec( 4);
            lfsm_vec( 9) <= lfsm_vec(10) XOR edt_channels_in(0);
            lfsm_vec(10) <= lfsm_vec(11) XOR lfsm_vec( 1);
            lfsm_vec(11) <= lfsm_vec( 0) XOR edt_channels_in(0);
         END IF;
      END IF;
   END PROCESS lfsm;
   ...
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 9

The following example shows two commands that do not include the synopsys pragma:

write_edt_files noprag1 -synthesis_script rtl_compiler

write_edt_files noprag2 -no_synopsys_pragma

Example 10

The following example uses the -tsdb option and default value of the set_tsdb_output_directory command. The module name prefix is "*current_design_design_id_tessent_*". See the set_context -design_identifier command for more information about the design identifier. See the instruments directory section for details about the instrument container directory.

write_edt_files -tsdb -replace

```
//  Writing ./tsdb_outdir/instruments/m8051_gate_edt.instrument/
m8051_gate_tessent_edt.v
//  Writing ./tsdb_outdir/instruments/m8051_gate_edt.instrument/
m8051_gate_edt.synthesis_dictionary
//  Writing ./tsdb_outdir/instruments/m8051_gate_edt.instrument/
m8051_gate_tessent_edt.icl
//  Writing ./tsdb_outdir/instruments/m8051_gate_edt.instrument/
m8051_gate_tessent_edt.pdl
//  Writing ./tsdb_outdir/instruments/m8051_gate_edt.instrument/
m8051_gate_tessent_edt.tcd
//  ----------------------------------------------------------------------
//  Begin EDT logic insertion
//  ----------------------------------------------------------------------
//  Inserting EDT logic in the design.
//  EDT logic insertion completed.
//  Writing modified netlist ./tsdb_outdir/dft_inserted_designs/
m8051_gate.dft_inserted_design/m8051.vg_no_instruments.
//  Writing interface netlist ./tsdb_outdir/dft_inserted_designs/
m8051_gate.dft_inserted_design/m8051.v_interface
```

Related Topics

report_edt_configurations

report_edt_pins

set_edt_options

set_edt_pins

write_design

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_failing_paths

Context: patterns -scan_diagnosis

Mode: analysis

Reports, for each diagnosed symptom from a failure log, the scan pattern number, suspect(s) and paths.

Usage

write_failing_paths [-DIAGnosis_report *diagnosis_report_filename*]
[-Pattern *number*] [-ENDpoints [*Cell_Ins_Name1 PO_Pin_Name1* ...]
[-OUTput *name* [-REPlace]]

Description

Reports, for each diagnosed symptom from a failure log, the scan pattern number, suspect(s) and paths.

You must specify performing at-speed diagnosis using the "set_diagnosis_options -at_speed on" command before using the write_failing_paths command. The write_failing_paths command issues a warning message if at-speed diagnosis is off. If you use the following commands:

read_patterns pat.file

set_diagnosis_options -at_speed on

write_failing_paths -diagnosis_report report.diag

Then you don't have to use the diagnose_failures command at all.

Arguments

- -DIAGnosis_report *diagnosis_report_filename*

  An optional switch and string pair specifying the diagnosis report filename from which failure diagnosis is processed. This option only provides results if the fail log is present in the path defined in the diagnosis report. If the tool cannot find the original fail log in the location stated in the diagnosis file, then no results are provided unless you re-run diagnosis.

- -Pattern *number*

  An optional switch and integer pair specifying the failing pattern number for which failing paths are reported.

- -ENDpoints *Cell_Ins_Name1 PO_Pin_Name1 ...*

  An optional switch and report failing paths that end at the given cell instance names or PO pin names. The end cell instances are the state elements where at-speed failures are captured. Default: report failing paths for all cells or POs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -OUTput *name*

  An optional switch and string pair specifying the filename (*name*) where failing paths are reported. The tool reports failing paths to the screen and log file.

Examples

Example Failing Path Report

```
//  Symptom = 1
//      Scan pattern: 42
//      Suspect = 1
        PATH "path1" =
        PIN /CORE/TP004/P1001/PC00W33_0/Q;
        PIN /CORE/TM00T/AL011/U117/B1  ;
        PIN /CORE/TM00T/AL011/U117/OEQ  ;
        PIN /CORE/TM00T/AL011/U39/I1  ;
        PIN /CORE/TM00T/AL011/U39/O  ;
        PIN /CORE/TM00T/AL011/AP00A00_0/D ;
        END ;
—— skipping paths "path2-4" in this example ——
        PATH "path5" =
        PIN /CORE/TP004/P1001/PC00W55_0/Q  ;
        PIN /CORE/TM00T/AL011/U116/B2  ;
        PIN /CORE/TM00T/AL011/U116/OEQ  ;
        PIN /CORE/TM00T/AL011/U54/B  ;
        PIN /CORE/TM00T/AL011/U54/OB ;
        PIN /CORE/TM00T/AL011/AP00A00_0/D  ;
        END ;
//      Suspect =  2
//         "path5"
//      Scan pattern: 121
//      Suspect =  1
//      "path1"
//      "path2"
//      Suspect =  2
//         "path5"
```

The above example shows 5 failing paths for symptom 1, suspect 1 at scan pattern 42. Detailed description of the PATH definition can be found in the FastScan manual. Also shown is that for same scan pattern, 42, there is only one failing path for symptom 1, suspect 2, which is equivalent to path5 that was defined earlier. For scan pattern 121, there are two patterns for suspect 1 of symptom 1, and they are path1 and path2 which are defined earlier; there is one path, path5 for suspect 2 of symptom 1.

Example 1

This example reports warning message because this command can only used for at-speed diagnosis.

set_diagnosis_options -at_speed off

diagnose_failures chip1.fail

write_failing_paths

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

This example reports the at-speed failing paths for the diagnosis result of *chip1.fail* on the screen.

```
set_diagnosis_options -at_speed on
diagnose_failures chip1.fail
write_failing_paths
```

Example 3

In this example at-speed failing paths of scan pattern 5 for *chip1.fail* and the failing paths of scan pattern 10 for *chip2.fail* are printed on the screen.

```
set_diagnosis_options -at_speed on
diagnose_failures chip1.fail -output chip1.diag
write_failing_paths -pattern 5
diagnose_failures chip2.fail -output chip2.diag
write_failing_paths -pattern 10
```

Example 4

The at-speed failing paths of diagnosis *chip1.diag* are printed on the screen. The -end_cell switch reports only failing paths ending at cell "/cpu/exu/pmi/ix926," where "/cpu/exu/pmi/ix926" is the name of the cell where at-speed failures are captured.

```
set_diagnosis_options -at_speed on
diagnose_failures chip1.fail -output results/chip1.diag
write_failing_paths  -end_cell /cpu/exu/pmi/ix926
```

Example 5

The critical paths of diagnosis results for *chip1.fail* are written to file named *chip1.path.*

```
set_diagnosis_options -at_speed on
diagnose_failures chip1.fail -output results/chip1.diag
write_failing_paths -output chip1.path
```

Related Topics

diagnose_failures

set_design_sources

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_failures

Context: dft -edt, patterns -scan, patterns -scan_diagnosis, patterns -failure_mapping

Mode: analysis

Injects faults and/or writes failing pattern results to a file.

Usage

write_failures ***failure_filename*** [-Replace]
[[{*pin_pathname* -Stuck_at {0 | 1}}…] |
[*pin_pathname* {-SLOW_TO_RISE | -SLOW_TO_FALL}]]
[-Pdet | -Exact] [-Max *max_failed_scan_pat_num*] [-CYCle]
[-Faulty_chain *faulty_chain_name faulty_chain_type* {-Cell *faulty_cell_id*…}…]

Description

Injects faults and/or writes failing pattern results to a file.

The resulting file can be used to perform a diagnosis of the failures with Tessent Diagnosis.

By default, the write_failures command performs a good simulation of the current design and test pattern source and writes the failing pattern information to the display. write_failures also does not produce the cell paths by default.

You can also use this command to inject faults into the design, simulate them, and write the resultant failures into the failure file for diagnosis. When faults are injected, a fault simulation is run with those values. The fault is injected with respect to the scan cell that is closest to the scan output.

For failure mapping, the tool creates core-level failure files by reverse mapping the top-level failures that were obtained by applying the top-level patterns generated with scan pattern retargeting. After reading a valid top-level failure file, the command extracts core-level failures for all failing cores and saves them into separate files that are named based on ***failure_filename*** and the naming convention as described below. The command strips the Verilog escaped characters too.

The tool names the core-level failure file for a failing core by appending its adjusted core instance name, its core module name, and its core test mode to the given failure file basename. A triple underscore ("___") is used to separate the core failure file basename from the other appended names, which are separated by double underscores ("__"). To convert the hierarchical instance name to a valid filename, the tool replaces "/" characters in the core instance name with a period character (".").

For example, the tool would save the core-level failure file for core instance core_10 of core module MyCore, tested in internal mode, to the name *./core_flogs/ fail1.top___core_10__MyCore__internal*. Likewise, the tool would save the core-level failure

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

file for the core instance A/B/core1 of core module “MyCore, tested in internal mode, to the name *./core_flogs/fail1.top___A.B.core1__MyCore__internal*.

Arguments

- ***failure_filename***

  A required string that specifies the name of a file to write the failing pattern information to.

- -Replace

  An optional switch that directs the tool to overwrite an existing ***failure_file.*** Use this if the specified ***failure_file*** already exists, and you want to update the contents.

- *pin_pathname* -Stuck_at {0 | 1}

  An optional, repeatable string, switch, and literal triplet that specifies the location and the value of a logic failure to inject into the design as follows:

  pin_pathname — A string that specifies the pin pathname of the location to inject the fault.

  -Stuck_at 0 | 1 — A switch and literal that specifies the stuck-at value for the fault. The stuck-at options include:

  0 — stuck-at-0 fault

  1 — stuck-at-1 fault

- *pin_pathname* {-SLOW_TO_RISE | -SLOW_TO_FALL}

  An optional repeatable string and switch pair that injects single transition fault to support at-speed diagnosis using the following criteria:

  pin_pathname — A string that specifies the pin pathname of the location to inject the transition fault.

  -SLOW_TO_RISE — A literal that specifies failure at the 0->1 transition.

  -SLOW_TO_FALL — A literal that specifies failure at the 1->0 transition.

  This argument supports injecting a single transition fault. Additionally, when the transition fault is injected, other faults (for example, stuck-at logic faults or chain faults) cannot be injected together, specifically this argument does not support a mixture of a transition fault with other faults.

- -Pdet

  An optional switch that directs the tool to output a failure when it simulates an X value where a binary value is expected for a specified fault. By default, only the binary detections are output to the file.

- -Exact

  An optional switch that directs the tool to output a failure when it simulates an X value where a binary value is expected or a binary value where an X value is expected for a specified fault. By default, only the binary detections are output to the file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Max *max_failed_scan_pat_num*

  An optional switch and integer pair that specifies a maximum number of failing patterns to write for the specified faults. When the maximum number is reached, the simulation stops. By default, all failing patterns are written.

- -CYCle

  An optional switch that writes the failure file in a cycle-based format. By default, a pattern-based format is output. For more format information, see the "Input File Requirements" chapter in the *Tessent Diagnosis User's Manual*.

- -Faulty_chain *faulty_chain_name faulty_chain_type*

  An optional repeatable switch and argument pair that specifies the names and fault types of the chain faults to inject into a design and simulate. Argument options include:

  faulty_chain_name — String that specifies the name of the faulty chain.

  faulty_chain_type — Literal that specifies the fault type associated with the chain. Argument options are the following:

  STUCK_AT_1

  STUCK_AT_0

  SLOW_TO_FALL

  SLOW_TO_RISE

  FAST_TO_RISE

  FAST_TO_FALL

  FAST

- -Cell *faulty_cell_id* ...

  An optional switch and repeatable integer that specifies the faulty cells within the specified faulty chain.

Examples

Example 1

The following example switches to analysis system mode, loads a test pattern source, injects a stuck-at 1 failure at the i_1006/o logic in the specified a pattern-based failure file.

```
set_system_mode analysis
read_patterns file1
write_failures fail1 i_1006/o -stuck_at 1

// failing_patterns=8 simulated_patterns=36 simulation_time=0.00 sec
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The output file looks like:

```
    4  chain1  3  L  H //uINTR/ix476
    6  /D_OUT(0)  L  H //uINTR/ix446
    7  chain1  3  L  H //uINTR/ix356
   ...
   29  /D_OUT(1)  L  H //uINTR/ix358
   29  chain1  3  L  H //uINTR/ix362
   31  /D_OUT(1)  L  H //uINTR/ix601
   31  /D_OUT(2)  L  H //uINTR/ix602
last_pattern_tested    35
```

Example 2

The following example shows a pattern-based output file written from Tessent TestKompress (4X compression).

```
0 edt_channel1 2  H L   // chain1---2---DFF_reg0
                        // chain2---2---DFF_reg1
                        // chain3---2---DFF_reg2
                        // chain4---2---DFF_reg3
1 edt_channel1 2  L H   // chain1---2---DFF_reg0
2 output[1]       H L   // PO
......
```

Example 3

The following example shows the cycle-based output file written from Tessent TestKompress (4X compression).

```
20 ch_out1 H L //pat 0 channel1 2 chain1---2---DFF_reg0
               //                chain2---2---DFF_reg1
               //                chain3---2---DFF_reg2
               //                chain4---2---DFF_reg3

......
```

Example 4

The following example reverse maps a top-level failure file into core-level failure files and saves them to the *./core_flogs* directory based on the naming convention described above.

write_failures ./core_flogs/top_flog1 -rep

```
//  command: write_failures ./core_flogs/top_flog1 -rep
//  Note: Core level faillog
//    (./core_flogs/top_flog1___logic_die_BOT_inst__logic_die__internal)
//    is created for core ( logic_die_BOT_inst__logic_die__internal ).
//  Note: Extract 1 core faillog for 1 failing core from top level
      faillog ( top_flogs/top_flog1 ).
```

Related Topics

diagnose_failures

read_failures

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_patterns

report_failures

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_fault_sites

Context: dft -edt, patterns -scan

Mode: analysis

Writes path definitions or bridge entries from the internal list to a file.

Usage

Bridge Faults Usage:

write_fault_sites {***filename***} [-Replace]
[-All | *net_pathname_pair* | *bridge_name…* | *net_pathname…* | *instance_name…*]
[-NAME | -SINGle] [-NET_pathname | -PIN_pathname] [-NOEQ]

Path Delay Faults Usage:

write_fault_sites {***filename***} [-Replace]
[*pathname*] [-All] [-Path *gate_id_begin gate_id_end*]

Description

Writes path definitions or bridge entries from the internal list to a file.

The write_fault_sites command writes the specified fault sites from the internal list to a file. Depending on the active fault model, fault sites may be defined with path definitions or bridge entries. Fault sites are loaded into the internal list with the read_fault_sites command.

For more information on bridge faults and the bridge fault definition file, refer to "The Static Bridge Fault Model" in the *Tessent Scan and ATPG User's Manual*.

For more information on path delay faults and the path definition file, refer to "Path Delay Test Set Creation" in the *Tessent Scan and ATPG User's Manual*.

By default, this command writes out the loaded fault attributes.

Arguments

- ***filename***

  A required string that specifies the name of the file to which the tool should write the path definition or bridge entry information.

- -Replace

  An optional switch that replaces the contents of the file, ***filename,*** if it already exists.

- -All

  An optional switch that displays all fault definitions for the currently loaded path definitions or bridge entries. This is the default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -Path *gate_id_begin gate_id_end*

  An optional switch and integer pair that specifies a particular path or portion of a path whose definition you want to write. Use this argument to write paths that were not defined in a path definition file and, therefore, were not loaded using the read_fault_sites command.

  The two integers specify two gate identification numbers that indicate the beginning and end of the path. The path begins at *gate_id_begin* and ends at *gate_id_end*.

  The value of *gate_id_begin* or *gate_id_end* is the unique gate identification number that the tool automatically assigns to every gate within the design during the model flattening process.

- *net_pathname_pair*

  A string pair that specifies the net or pin pathname to bridge fault site. The first string specifies NET1, and the second string specifies NET2.

- *bridge_name*

  A repeatable string that specifies the name of a specific bridge entry.

- *net_pathname*

  A repeatable string that specifies a single net or pin pathname contained in a bridge entry.

- *instance_name*

  A repeatable string that specifies an instance pathname contained in a bridge entry.

- -NAME

  An optional switch that specifies all entered strings are bridge names.

- -SINGle

  An optional switch that specifies all the entered strings are either a single net pathname or instance pathname. The tool searches for a net first. If a corresponding net cannot be found, the tool searches for a corresponding instance.

- -NET_pathname

  An optional switch that specifies to write net pathnames for bridge entries. This is the default.

- -PIN_pathname

  An optional switch that specifies to write pin names for bridge entries instead of net pathnames.

- -NOEQ

  An optional switch that writes only the representative fault information: net pair and fault categories.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following path delay example writes to the file my_path_definitions using the path definition file format, the pins in the specified path:

write_fault_sites /user/design/my_path_definitions -path 180 178

The following shows an example of the contents of a path definition file:

```
PATH "path0" =
    PIN /I$6/Q + ;
    PIN /I$35/B0 + ;
    PIN /I$35/C0 + ;
    PIN /I$1/I$650/IN + ;
    PIN /I$1/I$650/OUT - ;
    PIN /A_EQ_B + ;
END ;
```

Example 2

The following bridge example reports all bridge entries in the current fault site list, then writes one entry (including its attributes) to the file my_bridge_definitions using the bridge fault definition file format:

report_fault_sites

```
BRIDGE {
  NET1 = "/G5";
  NET2 = "/G10";
  FAULTS = {DS, DS, DS, DS};
  NAME = "U7.BRIDGE";
}
BRIDGE {
  NET1 = "/G4";
  NET2 = "/G10";
  FAULTS = {DS, DS, DS, DS};
  NAME = "U5.BRIDGE";
}
BRIDGE {
  NET1 = "/G11";
  NET2 = "/G10";
  FAULTS = {DS, DS, DS, DS};
  NAME = "U0.BRIDGE";
}
```

write_fault_sites /user/design/my_bridge_definitions U5.BRIDGE -name

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following shows the contents of the resultant *my_bridge_definitions* file:

```
VERSION    1.1

FAULT_TYPE="BRIDGE_STATIC_4WAY_DOM"

BRIDGE {
  NET1 = "/G4";
  NET2 = "/G10";
  FAULTS = {DS, DS, DS, DS};
  WEIGHT = 1.0000;
  DISTANCE = 4.96575000e-06 um;
  PARALLEL_run = 7.30000000e-05 um;
  X_COORDINATE = 174500;
  Y_COORDINATE = 570000;
  LAYER = "1";
  NAME = "U5.BRIDGE";
}
```

Related Topics

add_faults

delete_fault_sites

read_fault_sites

report_fault_sites

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_faults

Context: dft -edt, dft -test_points, patterns -scan

Mode: analysis

Writes fault information from the current fault list to a file.

Usage

Stuck/Transition Usage

write_faults ***filename*** [-Replace] [-NO_Subclass] [-FORmat {Basic | Mtfi}]
{[-Class *fault_class*[.*fault_subclass*]...] [-NOEQ] [-Stuck_at {01 | 0 | 1}]
[-CHANGE_classification_to *fault_class*[.*subclass*]]
[*pin_pathname* ...] [-Hierarchy *integer*] [-Min_count *integer*]
[-VErilog | -VHdl] [-CEll_name]
[-CLOCK_Domains {ALL | *clock_pathname* ...} [-NO_EQUivalent_clocks]
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]]
[-CAPture_procedures {ALL | *capture_procedure_name* ...}] [-DElay_data]
[-TIMING_CRITICAL]}
| {[-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls]}
| {-UNlisted}
[-FRom *pin* ...][-THrough *pin* ...][-TO *pin* ...][-FROM_Clock *pin* ...][-TO_Clock *pin* ...]
[-stop_at {sequential_elements | scan_cells | ports_only}]
[-fanout_off_path_stop {on | off }] [-fanin_off_path_stop { on | off }]

UDFM Usage

write_faults ***filename*** [-Replace] [-NO_Subclass] [-FORmat Mtfi]
{ -Class *fault_class*[.*fault_subclass*]... | *instance_pathname* |
[-CHANGE_classification_to *fault_class*[.*subclass*]] [-Hierarchy *integer*] [-NOEQ]
[-CLOCK_Domains {ALL | *clock_pathname* ...} [-NO_EQUivalent_clocks]
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]]
[-DElay_data] [-TIMING_CRITICAL] [-FAULT_TYPE {stuck | transition}]
|{[-Udfm_type *name*] [-INstance *name*] [-CEll *name*] [-MOdule *name*]}}
| {-UNlisted}
[-FRom *pin* ...][-THrough *pin* ...][-TO *pin* ...][-FROM_Clock *pin* ...][-TO_Clock *pin* ...]
[-stop_at {sequential_elements | scan_cells | ports_only}]
[-fanout_off_path_stop {on | off }] [-fanin_off_path_stop { on | off }]

Bridge Usage

write_faults ***filename*** [-Replace] [-NO_Subclass] [-FORmat {Basic | Mtfi}]
{ [*net_pathname_pair* | *bridge_name* ... | *net_pathname* ... | *instance_name* ...]
[-Class *fault_class*[.*fault_subclass*]]
[-NAME | -SINGle] [-NET_pathname | -PIN_pathname] [-NOEQ]
[-VErilog | -VHdl]
[-CLOCK_Domains {ALL | *clock_pathname* ...} [-NO_EQUivalent_clocks]
[-EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains]]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

[-CAPture_procedures {ALL | *capture_procedure_name* ...}]}
|{[-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls]}

Toggle/Iddq Usage

write_faults ***filename*** [-Replace] [-NO_Subclass] [-FORmat {Basic | Mtfi}]
{[-Class *fault_class*[.*fault_subclass*]...] [-NOEQ] [-Stuck_at {01 | 0 | 1}]
[-CHANGE_classification_to *fault_class*[.*subclass*]]
[*pin_pathname* ...] [-Hierarchy *integer*] [-Min_count *integer*]
[-VErilog | -VHdl] [-CEll_name]
[-CLOCK_Domains {ALL | *clock_pathname* ...} [-NO_EQUivalent_clocks]]
[-CAPture_procedures {ALL | *capture_procedure_name* ...}]}
| {[-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls]}
| {-UNlisted}
[-FRom *pin* ...][-THrough *pin* ...][-TO *pin* ...][-FROM_Clock *pin* ...][-TO_Clock *pin* ...]
[-stop_at {sequential_elements | scan_cells | ports_only}]
[-fanout_off_path_stop {on | off }] [-fanin_off_path_stop { on | off }]

Path Delay Usage

write_faults ***filename*** [-Replace] [-Both | -Rise | -Fall]
[-Class *fault_class*[.*fault_subclass*]] [*name_of_delay_path*] [-NO_Subclass]
[-CHANGE_classification_to *fault_class*[.*subclass*]]
[-VErilog | -VHdl] [-FORmat {Basic | Mtfi}]

Description

The write_faults command is similar to the report_faults command, except that the data is written into a file. You can review or modify the file and later load the information into the fault list with the read_faults command. You can use the optional arguments to narrow the focus of the report to only specific stuck-at or transition faults that occur on a specific object in a specific class. If you do not specify any of the optional arguments, write_faults writes information on all known faults to the file. The default output format for this command is MTFI. For more information about MTFI, refer to the "MTFI File Format" chapter in the *Tessent Scan and ATPG User's Manual*.

The file contains the following columns of information for each fault:

- **fault value** — The fault value may be either 0 (for stuck-at-0 or slow-to-rise transition faults) or 1 (for stuck-at-1 or slow-to-fall transition faults).
- **fault code** — A code name that indicates the lowest-level fault class assigned to the fault.
- **fault site** — The pin pathname of the fault site.
- **cell name** (optional) — The name of the cell corresponding to the fault (included only if you used the -Cell_name argument).

Use the -Hierarchy option to write a hierarchical summary of the selected faults. The summary identifies the number of faults in each level of hierarchy whose level does not exceed the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

specified level number. You can further specify the hierarchical summary by using the -Min_count option, which specifies the minimum number of faults that must be in a hierarchical level before writing.

You can choose to display either collapsed or uncollapsed faults by using the set_fault_mode command.

This command can write specific AU.TC and AU.PC faults when you previously issue the "report_statistics -detailed_analysis" command. If you want to report specific AU.TC faults, you must specify gate IDs instead of the full pin pathname. If you want to report specific AU.PC faults, you must specify the full pin pathname. See Example 5 for an example of writing specific AU.TC and AU.PC faults.

Arguments

- ***filename***

  A required string that specifies the name of the file where the tool is to write the fault information. The ***filename*** must be the first argument. If the filename ends with *.gz* or *.Z*, the tool automatically compresses the file with the GNU gzip utility. See set_gzip_options for more details.

- -Replace

  An optional switch that replaces the contents of ***filename*** if the file already exists.

- -Class *fault_class*[*.fault_subclass*]

  An optional switch and literal pair that specifies the fault class or fault class/sub-class to write. Use the code when specifying a fault class or sub-class. If you do not use this switch, the default is to write all fault classes.

  This command can write specific AU.TC and AU.PC faults after you issue the "report_statistics -detailed_analysis" command. For AU.TC faults, you must specify gate IDs rather than the full pin pathname. For AU.PC faults, you must specify the full pin pathname. For more information, refer to "Example 5" on page 3093.

- -change_classification_to *fault_class*[*.subclass*]

  An optional switch and fault class with optional subclass pair that enables you to force the tool to write faults as the specified class and optional subclass. By default, the tool writes the fault class (and subclass) of the faults learned by the tools (for example, AU.TI or DS). For a list of fault classes and subclasses refer to "Fault Sub-Classes" in the *Tessent Scan and ATPG User's Manual*.

- -FORmat [Basic | Mtfi]

  An optional switch and literal pair that specifies the format of the fault file. By default, the command writes out in MTFI format. For more information about MTFI, refer to the "MTFI File Format" chapter in the *Tessent Scan and ATPG User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -FRom *pin*…

  An optional switch and repeatable string pair that sets the start pin or pins of the cone of logic the tool uses to select faults to write. When specified, the tool performs forward cone tracing from the specified pin or pins until it encounters a stopping point. You can specify one or more instance names rather than pin names. When you specify an instance name with -from, the tool uses its output pins when defining the cone.

- -THrough *pin*…

  An optional switch and repeatable string pair that sets the through pin or pins of the cone of logic the tool uses to select faults to write. When specified, the tool performs forward and backward cone tracing from the specified pin or pins until it encounters a stopping point in both directions. You can specify one or more instance names rather than pin names. When you specify an instance name with -through, the tool uses its output pins when defining the cone.

- -TO *pin*…

  An optional switch and repeatable string pair that sets the end pin or pins of the cone of logic the tool uses to select faults to write. When specified, the tool performs backward cone tracing from the specified pin or pins until it encounters a stopping point. You can specify one or more instance names rather than pin names. When you specify an instance name with -to, the tool uses its input pins when defining the cone.

  Note

  When you use -from, -to, and -through with other switches, the selected faults are the intersection of the cones traced for all specified switches.

- -FROM_Clock *pin* ...

  An optional switch and repeatable string pair that sets the launch clock pin or pins of the cone of logic the tool uses to select faults to write. When specified, the tool performs forward cone tracing from the output ports of the state elements launched by the specified clock pin or pins until it encounters a stopping point.

- -TO_Clock *pin* ...

  An optional switch and repeatable string pair that sets the capture clock pin or pins of the cone of logic the tool uses to select faults to write. When specified, the tool performs backward cone tracing from the data ports of the state elements captured by the specified clock pin or pins until it encounters a stopping point.

- -stop_at {sequential_elements | scan_cells | ports_only}

  An optional switch and literal pair that sets the stop condition for tracing of the cone of logic used to select faults to write. By default, the traced logic cone includes only combinational logic, so cone tracing stops at any sequential elements. When you specify “scan_cells”, the cone includes non-scan cells and tracing stops only at scan cells. When you specify “ports_only”, the cone includes any sequential elements and tracing stops only at primary inputs and outputs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -fanout_off_path_stop {on | off}

  An optional switch and literal pair that specifies an additional stop condition when tracing the fan-in fault cone backward and reaching a multiple fanout gate G. When this switch is on, tracing stops at gate G unless the cone already includes all of the fanout gates of G. The default is off. You can use a separate command, set_fault_selection_options, to change the default of this stop condition.

- -fanin_off_path_stop {on | off}

  An optional switch and literal pair that specifies an additional stop condition when tracing the fanout fault cone forward and reaching a multiple fan-in gate G. When this switch is on, tracing stops at gate G unless the cone already includes all of the fan-in gates of G. The default is off. You can use a separate command, set_fault_selection_options, to change the default of this stop condition.

- -UNlisted

  An optional switch that writes only the unlisted fault data. If there are multiple grayboxed instances in the design, this switch writes accumulated fault numbers. This switch is not compatible with the -fault_type argument.

- -NO_Subclass

  An optional switch that prevents the display of fault sub-classes after the fault code. The default is to display fault sub-classes when available.

- -Stuck_at 01 | 0 | 1

  An optional switch and literal pair that specifies the stuck-at or transition faults you want to write, according to the following table:

	01 (default)	**0**	**1**
Stuck-At	stuck-at-0 stuck-at-1	stuck-at-0	stuck-at-1
Transition	slow-to-rise slow-to-fall	slow-to-rise	slow-to-fall

- *pin_pathname* OR *name_of_delay_path*

  An optional repeatable string that specifies the list of pins, instances, or delay paths whose faults you want to write.

- -Hierarchy *integer*

  An optional switch and integer pair that specifies the maximum fault class hierarchy level for which you want to write a hierarchical summary of the faults.

- -Min_count *integer*

  An optional switch and integer pair that you can use with the -Hierarchy option to specify the minimum number of faults that must be in a hierarchical level to write a hierarchical summary of the faults. The default is 1.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -NOEQ
  - For stuck-at, transition, toggle, iddq, and udfm faults only, an optional switch that turns off the display of "EQ" as the fault class for any equivalent faults; the fault class displayed is then that of the representative fault. When you do not specify this switch, the tool displays an "EQ" as the fault class for any equivalent faults. This switch is meaningful only when the set_fault_mode command is set to Uncollapsed. For more information about representative and equivalent faults, see "Fault Collapsing" in the *Tessent Scan and ATPG User's Manual*.
  - For bridge faults only, an optional switch that writes only the representative bridge fault information: the net pair and fault classes of each corresponding dominant net. The tool does not report physical data.
- -Both | -Rise | -Fall

  An optional switch that specifies which faults to write for each path already added with the read_fault_sites command. Use these switches only for path delay faults.

  -Both - An optional switch that writes both the slow-to-rise and slow-to-fall faults. This is the default.

  -Rise - An optional switch that writes only the slow-to-rise faults.

  -Fall - An optional switch that writes only the slow-to-fall faults.
- -VErilog | -VHdl

  An optional switch that outputs the fault paths in either Verilog or VHDL syntax, rather than using the existing netlist-independent format.
- -CEll_name

  An optional switch that specifies to list, for each reported fault, the corresponding circuit element (Tessent Cell library cell, Verilog primitive, primary input, or primary output) as summarized in Table 9-4. Cell names displayed for equivalent faults are based on the cells associated with the equivalent faults, not the cells associated with the representative faults. This switch is valid for the stuck-at, transition, toggle and IDDQ fault models only.

Table 9-4. Name Conventions Used by write_faults -Cell_name

Corresponding Circuit Element	Listed Cell Name
Tessent Cell library cell	Name of the Tessent Cell library cell
Supported Verilog primitive (for a list, see the *Tessent Cell Library Manual*)	Name of the Verilog primitive
Primary input or output	"primary_input" or "primary_output"

- *net_pathname_pair*

  A string pair that specifies the net or pin pathname to bridge fault site. The first string specifies NET1, and the second string specifies NET2.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- *bridge_name*

  A repeatable string that specifies the name of a specific bridge entry.

- *net_pathname*

  A repeatable string that specifies a single net or pin pathname contained in a bridge entry.

- *instance_name*

  A repeatable string that specifies an instance pathname contained in a bridge entry.

- -NAME

  An optional switch that specifies all entered strings are bridge names.

- -SINGle

  An optional switch that specifies all the entered strings are either a single net pathname or instance pathname. The tool searches for a net first. If it cannot find a corresponding net, the tool searches for a corresponding instance.

- -NET_pathname

  An optional switch that reports net pathnames for bridge entries. This is the default.

- -PIN_pathname

  An optional switch that reports pin pathnames for bridge entries instead of net pathnames.

- -CLOCK_Domains {ALL | *clock_pathname*…}

  An optional switch and literal or repeatable string pair that specifies a list of clocks and directs the tool to write only faults that are potentially detectable using any of the specified clocks (or equivalent clocks). For a transition fault to be potentially detectable, it must be detectable when the same clock (one of the specified clocks or an equivalent) is used for launch and capture. Any other type of fault must be detectable when one of the specified clocks or equivalents is used as the capture clock. The argument choices are as follows:

  ALL — A literal that specifies all the clocks in the design.

  clock_pathname — A repeatable string that specifies a particular clock.

  A fault written using this switch might also be detectable by an unspecified clock.

  The tool takes user-defined non-race clocks and faults between synchronous clock groups into account when adding or deleting faults by clock domain.

- -NO_EQUivalent_clocks

  An optional switch that prevents the -Clock_domains switch from writing faults in equivalent clock domains. When you specify this switch, the tool considers only faults within clock domains when adding or deleting faults by clock domain.

- -EXCLUDE_FAULTS_BETWEEN_SYNC_clock_domains

  An optional switch that takes user-defined non-race clocks and faults between synchronous clock groups into account when deleting faults in the clock domain.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -CAPture_procedures {ALL | *capture_procedure_name*…}

  An optional switch and literal or repeatable string pair that specifies a list of enabled named capture procedures and directs the tool to write faults that are potentially detectable by any of the specified procedures. The argument choices are as follows:

  ALL — A literal that specifies all enabled named capture procedures.

  capture_procedure_name — A repeatable string that specifies a particular enabled named capture procedure.

  See the set_capture_procedures command description for information about enabling or disabling named capture procedures.

- -DElay_data

  An optional switch that saves the delay faults, including the slacks of the faulty site.

- -TIMING_CRITICAL

  An optional switch that saves the timing critical faults identified by the tool. Before issuing the write_faults command with this switch, you must issue the set_atpg_timing command with the -TIMING_CRITICAL switch and *timing_margin%* value.

  Note

  In order to use the -DElay_data or -TIMING_CRITICAL switches, you must have previously loaded an SDF file containing timing information. See "Timing-Aware ATPG" in the *Tessent Scan and ATPG User's Manual* for complete information.

- -FAULT_TYPE {stuck | transition}

  An optional switch and literal pair that writes a stuck or transition fault file, as specified, containing the port-equivalent UDFM faults. This argument is not compatible with the -unlisted switch. The "stuck" value is only permitted with fault type "udfm -static_faults". The "transition" value is only permitted with fault type "udfm -delay_faults".

- -SCAN_Enable

  An optional switch that only writes faults that fan out to the select lines of multiplexers in the scan path. For this switch, a multiplexer is either a MUX simulation primitive or a nonprimitive type multiplexer composed of AND and OR gates. In essence, this switch writes all faults that are in the fan-in cone of local scan enable signals and are dominated by them.

- -CLOCK_Cones

  An optional switch that deletes all faults in the clock cone. The clock cone is the intersection of the fan-in of the sequential element clock ports and the fanout of the clock sources. This switch considers any sequential elements, such as flops, latches, RAMs, and ROMs, not just scan cells.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -IO

  An optional switch that specifies to write faults with the following conditions:

  - Only controlled by PIs — To be acted on by the -IO switch, a PI either must not be defined as a clock or, if defined as clock (or read/write control), must be constrained off during capture.
  - Only observed by POs

- -ASYnchronous_controls

  An optional switch that only writes faults that fan out to Set/Reset ports of state elements and RAMs. This switch applies to a subset of the faults that the -CLOCK_Cones switch writes.

- -Udfm_type *name*

  An optional switch and string pair that specifies a name of a UDFM fault type. Use this option to write the faults associated with a specified UDFM fault type to the file. The name string is case insensitive and may include any number of asterisk (*) and question mark (?) wildcard characters.

- -CEll *name*

  An optional switch and string pair that specifies the name of a library cell. Use this option to write all faults associated with a specified library cell to the file. The name string is case insensitive and may include any number of asterisk (*) and question mark (?) wildcard characters.

- -MOdule *name*

  An optional switch and string pair that specifies the name of a module. Use this option to write all faults associated with a specified module to the file. The name string is case insensitive and may include any number of asterisk (*) and question mark (?) wildcard characters.

- -INstance *name*

  An optional switch and string pair that specifies the name of an instance. Use this option to display all faults associated with a specified instance. The name string may include any number of asterisk (*) and question mark (?) wildcard characters.

Examples

Example 1

The following example performs an ATPG run and then writes all the untestable faults to a file for review:

```
set_fault_type stuck
set_system_mode analysis
create_patterns
write_faults faultlist -class ut -format basic
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The contents of the *faultlist* file look similar to the following:

```
0     UU    /ip_scan_clk
1     UU    /ip_scan_clk
0     UU    /ipt_scan_tf_en
1     UU    /ipt_scan_tf_en
0     UU    /ff0/qb
1     UU    /ff0/qb
...
```

Example 2

The following example rewrites all the untestable faults to the file but includes the name of the cell associated with each fault in the written information:

write_faults faultlist -class ut -cell_name -format basic -replace

The file now contains an additional column with the cell information:

```
0     UU    /ip_scan_clk  primary_input
1     UU    /ip_scan_clk  primary_input
0     UU    /ipt_scan_tf_en  primary_input
1     UU    /ipt_scan_tf_en  primary_input
0     UU    /ff0/qb  dsffd
1     UU    /ff0/qb  dsffd
```

Example 3

The following command writes out a fault file in MTFI format that contains all faults:

write_faults fault_file1 -format mtfi

This is an example of a small MTFI file that could result from the previous command:

```
FaultInformation {
   version : 1;
   FaultType Stuck  {
      FaultList {
         Format : Identifier, Class, Location;
         Instance ("") {
            1, UC, "in1";
            0, DS, "in1";
            0, DS, "i1/IN0";
            1, DS, "i1/OUT";
            1, DS, "i1/IN0";
            1, AU, "i1/IN1";
            0, UC, "i5/Z";
            1, DS, "out";
            1, DI , "i5/Z";
         }
      }
   }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

The following example displays the fault data for ds class faults associated with library cells that start with mux or an to the *my_fault_list* file:

write_faults my_fault_list -class ds -cell mux* -cell an*

```
// Command output
FaultInformation {
   version : 1;
   FaultType (UDFM) {
      FaultList {
         Format : Identifier, Class;
         UdfmType ("intra-cell-bridges_1") {
            Cell ("MUX2") {
               Instance ("/top/i1/i4") {
                  "F2", DS;
               }
            }
            Cell ("AND3") {
               Instance ("/top/i1/i456") {
                  "F2", DS;
                  "F4", DS;
                  "F5", DS;
                  "F7", DS;
               }
            }
         }
      }
   }
}
```

Example 5

The following example shows how to write specific AU.TC and AU.PC faults:

report_statistics -detailed_analysis
 # you must first run this command

write_faults fltlist_au.tc -class au.tc 1195
 # you must supply a gate ID rather than pin pathname

write_faults fltlist_au.pc -class au.pc mode[3]
 # you must supply a pin pathname

Example 6

The following example writes out bridge faults in the full format (the -noeq switch is unspecified):

write_faults exa.flt

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
FaultInformation {
version : 1;
FaultType (Bridge) {
  FaultList {
   Format : Identifier, Class, LocationA, LocationB, BridgeName;
   Instance ("") {
    A0, DS, "/G13", "/G12", "C506";
    A1, DS, -, -, -;
    B0, DS, -, -, -;
    B1, DS, -, -, -;
    A0, DS, "/G13", "/G12", "C505";
    A1, DS, -, -, -;
    B0, DS, -, -, -;
    B1, DS, -, -, -;
    A0, DS, "/G13", "/G12", "C504";
    A1, DS, -, -, -;
    B0, DS, -, -, -;
    B1, DS, -, -, -;
    A0, DS, "/G2", "/G4", "C478";
    A1, DS, -, -, -;
    B0, DS, -, -, -;
    B1, DS, -, -, -;
  }
 }
}
}
```

Example 7

The following example writes out bridge faults in an abbreviated format that lists only the net pair and fault classes of each corresponding dominant net:

write_faults exa.flt -noeq

```
FaultInformation {
version : 1;
FaultType (Bridge) {
  FaultList {
   Format : Identifier, Class, LocationA, LocationB, BridgeName;
   Instance ("") {
    A0, DS, "/G13", "/G12", "";
    A1, DS, -, -, -;
    B0, DS, -, -, -;
    B1, DS, -, -, -;
    A0, DS, "/G2", "/G4", "";
    A1, DS, -, -, -;
    B0, DS, -, -, -;
    B1, DS, -, -, -;
  }
 }
}
}
```

Related Topics

add_faults

delete_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_fault_sites

read_faults

set_checkpointing_options

report_faults

set_fault_mode

set_fault_sampling

set_fault_selection_options

set_fault_type

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_flat_model

Context: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: setup, analysis

Saves the flattened circuit model, the scan trace, and all DRC-related information to a specific binary file with options to encrypt the file, password protect it, and prohibit (or hide) commands from running on it.

Usage

write_flat_model ***filename***
[-Password *password*] [-Command_file *command_file*] [-Replace]
[-INCLUDE_Proc_simulation_data {Exclude_test_setup_and_test_end | All}]

Description

This command enables restoration of a new session to the previous session's state based on saved checkpoint data. The command enables you to enter into analysis mode without reading the netlist, flattening, and performing DRC.

To protect your IP, you can encrypt and password-protect the file. You can specify commands to disable or hide, which further protects your design data.

Note

After saving the flat model, you cannot add primary inputs to the model. This includes using the "add_scan_chains -internal" command.

The write_flat_model command does not save pattern or fault list information in the flat model file. However, you can use the set_checkpointing_options command to save the pattern set and the fault list periodically.

Note

If you are preparing a flat model for later use in Tessent Diagnosis, save the flat model immediately after performing ATPG (with the create_patterns command) and saving final patterns. For more information, refer to "Preparing the Design Netlist" in the *Tessent Diagnosis User's Manual*.

When reading in a previously-stored flat model, the tool defaults to the same system mode as when the model was saved. It is also possible to save only the flat model (no scan chain or DRC data) to be restored into setup mode from any system mode.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ***filename***

  A required string that specifies the name of the file to which you want to write the flattened circuit model. If the filename ends with *.gz* or *.Z*, the tool automatically compresses the file with the GNU gzip utility. See set_gzip_options for more details.

- -Password *password*

  An optional switch and string pair that directs the tool to encrypt the netlist and embed the specified *password* in the saved file. The *password* is case sensitive and there are no restrictions on its content. Before reading in a flat model that was saved using the -password switch, the tool requires you to enter the password that was provided with the switch when the model was saved.

  For batch runs, you can set the TESSENT_FLAT_MODEL_PASSWORD environment variable to the specified password. The tool then obtains the password from the environment variable instead of querying you for it. However, if the password setting of this variable is incorrect, the tool will query you for it.

  Caution

  Be sure to record the password in a secure location. This ensures that a forgotten password is only a minor inconvenience.

- -Command_file *command_file*

  An optional switch and string pair that specifies the pathname of an ASCII file you previously created that lists commands you want the tool to permanently disable (hide) for this flat model. You can create this file with any ASCII text editor.

  Choose one of the two keywords, **disable** or **enable**, to place by itself on the first line of the command file. On subsequent lines, list the commands to which the keyword applies, one command or command group per line. The keywords have the following meaning:

  - **disable** — Renders the commands listed after the keyword inoperative. If the design is one you read into the tool as a flat model, any commands hidden already as a result of the previous flattening remain inoperative; omitting them from the current “disable” list does not re-enable them. All other commands the tool normally supports are available.
  - **enable** — Hides commands except those listed after the keyword. If the design is one you read into the tool as a flat model, commands already hidden as a result of the previous flattening remain hidden; listing them after the “enable” keyword does not re-enable them.

  Note

  The dofile, exit, and help commands are immune to the command file and are always available.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You specify a command group by listing just the first word or words that all commands in the group have in common. For example, you could list "set" on one line in order to specify the group of all commands that start with the word, "set."

- -Replace

  An optional argument that lets you overwrite an existing circuit model file.

- -INCLUDE_Proc_simulation_data {Exclude_test_setup_and_test_end | All}

  An optional argument and keyword pair that lets you save all of the DRC simulation data with the flat model. By default, the tool saves DRC simulation data for everything except for the test_setup and test_end procedures with the flat model. Saving all DRC simulation data can save time if you need to debug your test_setup or test_end procedures. However, saving this data can result in a much larger flat model file, especially when the test_setup or test_end procedures contain a large number of cycles.

Examples

Example 1

The following example creates a flat model and saves the model along with all DRC simulation data:

create_flat_model
write_flat_model file1 -include_proc_simulation_data all

Example 2

The following example uses an external command file to ensure the saved flat model only works with the following commands:

- dofile
- exit
- help
- any command that starts with "set"
- any command that starts with "write_primary"

create_flat_model
write_flat_model file1 -command_file commands_for_file1
exit

```
shell> cat commands_for_file1
enable
set
write primary
```

Related Topics

diagnose_failures

create_flat_model

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

read_flat_model

set_checkpointing_options

write_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_gds

Context: patterns -scan_diagnosis

Mode: analysis

Writes out the all the physical information in the currently open LDB in the GDS format into the specified output file.

Usage

write_gds ***output_GDS_filename***

Description

You must open an LDB using the "open_layout ***layout_database_name***" command before you run the write_gds command. The write_gds command always creates a GDS file that has chip-level coordinates. The tool uses a hierarchical database to flatten the coordinate system, including when the layout database is for one core only.

Arguments

- ***output_GDS_filename***

  A required string that specifies the name of the GDS file created with the write_gds command.

  Note

  This command creates a flat GDS file which can be large if you run the command for a big design.

Related Topics

open_layout

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_icl

Context: all contexts

Mode: setup, analysis

Prerequisite: ICL modules must be available. You use the read_icl command to create ICL modules.

Writes out ICL modules created and/or read in with the read_icl command to the specified file.

Usage

write_icl ***filename*** [-modules *modules_list* [-hierarchical]]
[-exclude_modules *exclude_modules*] [-created]
[-modified] [-replace]

Description

Writes out ICL modules created and/or read in with the read_icl command to the specified file.

This command writes out read-in and/or created ICL modules. You can merge both read-in and created ICL modules into the same file. The recommended use model is to write out the ICL hierarchically into a single file which is the default behavior when neither the -modules or -created option are specified.

Use the -modules switch to write only specific ICL modules. The default is to write the top module with all of the modules in the hierarchy below including the read-in modules and the modules below them. Use the -created option if you only want to write out the ICL modules created by ICL extraction and omit those that were read-in.

Arguments

- ***filename***

  A required string that specifies the name of the file to which to write all specified modules.

  Note

  For backward compatibility with v2014.4 and earlier releases, you can use the -output_file switch to specify the name of the file. For example:

  write_icl -output_file icl_out.icl

- -modules *module_list*

  An optional switch and value pair that specifies the names of the modules to write out. The module_list value is a Tcl list of one or more ICL module names, or a collection of one or more ICL modules as returned by the get_icl_modules command.

- -hierarchical

  An optional switch that specifies to write out all child modules below the specified ICL modules. This switch can only be used with the -modules switch.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -exclude_modules *exclude_modules*

  An optional switch and value pair that excludes ICL modules from being written out. This argument is only useful when the -hierarchical option is inferred or specified. All ICL modules below these excluded ICL modules are also excluded if they are not specified in the -modules list or found below other non-excluded ICL modules.

- -created

  An optional switch that specifies to limit the written ICL modules to only those created. Specifying this option is equivalent to using the following: "-modules [get_icl_modules -filter is_created]."

- -modified

  An optional switch that verifies whether a module has the is_modified attribute set to true; if it does, the tool writes out the module.

- -replace

  An optional switch that specifies to overwrite an existing file. When you use this switch, the error message that is normally generated if the file being written out already exists is suppressed.

Examples

The following example writes out the current design and every other ICL module instantiated below it into a common file.

```
write_icl icl_out.icl
```

Related Topics

get_icl_modules

read_icl

set_current_design

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_loops

Context: dft -edt, dft -scan, patterns -scan

Mode: analysis

Writes a list of all the current feedback loops to a file.

Usage

write_loops ***filename*** [-Replace]

Description

The write_loops command writes all feedback loops in the circuit to a file. For each loop, the file contents show whether the loop is a coupling loop, single multiple fanout loop, or broken by a constant. The report also includes the pin pathname and gate type of each gate in each loop.

Arguments

- ***filename***

  A required string that specifies the name of the file to which you want to write the loops.

- -Replace

  An optional switch that replaces the contents of the file if the *filename* already exists.

Examples

The following example writes a list of all the loops in the circuit to a file:

```
set_system_mode analysis
write_loops loop.info -replace
```

Related Topics

report_loops

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_memory_repair_dictionary

Context: patterns -ijtag

Mode: analysis

Writes the CompressBisrChain configuration file.

Usage

write_memory_repair_dictionary [-file_name *filename*]
[-return_dictionary]
[-fuse_box_controller_icl_instance *fuse_box_controller_icl_instance*]
[-replace]

Description

Writes the configuration file to the current working directory, as a Tcl dictionary, that is required to run the CompressBisrChain script. The resulting file contains all the BISR controller parameters, as well as the ordered list of BISR register ICL instances listed per power domain group. The CompressBisrChain script is used to program the fuse box using the FuseBox Access mode, which uses the IJTAG interface instead of using the BISR controller's autonomous self-fusebox programming mode. For more information, refer to "Top-Level Verification and Pattern Generation" in *Tessent MemoryBIST User's Manual For Use With Tessent Shell*.

Arguments

- -file_name *filename*

  An optional switch and string pair that specifies the name of the file to which you want to write the CompressBisr configuration contents. If this property is not specified, the default output file written to the current working directory is named "*<design_name>_bisr_info.tcl*".

- -return_dictionary

  An optional switch that instructs the command to return the content as a Tcl dictionary rather than saving it to a file.

- -fuse_box_controller_icl_instance *fuse_box_controller_icl_instance*

  An optional switch and string pair that specifies the BISR controller ICL instance to use.

- -replace

  An optional switch that replaces the contents of *filename* if the file already exists.

Examples

Example 1

The following example shows the write_memory_repair_dictionary command executed after ICL extraction, which is the appropriate point in the memory BISR DFT flow to invoke the command and create the CompressBisrChain configuration file. The configuration file is

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

automatically created in the legacy LV design flow, but is generated at user discretion and specification in the Tessent Shell design flow.

```
process_dft_specification
extract_icl
set_context patterns -ijtag
write_memory_repair_dictionary -replace
```

Example 2

Figure 9-6 shows an example CompressBisrChain configuration file created by the write_memory_repair_dictionary command as shown in the previous example.

The design contains a sub-block named “core_a”, which contains BISR registers for memA and memB on power domain group “PDG_coreA2”, and BISR registers for memE on power domain group “PDG_coreA1”. The top-level design contains memC and memD on the default power domain group “PDG_Default”.

Figure 9-6. Example CompressBisrChain Configuration File

```
set memory_repair_dict {
  controller_instance top_design_tessent_mbisr_controller_inst
  buffer_size 7
  zero_counter_bits 5
  fuse_box_size 512
  fuse_box_address_bits 10
  max_fuse_box_programming_sessions 1
  power_domain_group_labels {PDG_Default PDG_coreA1 PDG_coreA2}
  fuse_box_init_duration 1000
  fuse_box_read_duration 200
  repair_method hard
  BisrChainDict {
    core_a.memA_bisr_inst.ShiftReg[0] {
      PowerDomainGroup PDG_coreA2
      GlobalChainIndex 0
    }
    .
   .
   .
    }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
 core_a.memA_bisr_inst.ShiftReg[14] {
   PowerDomainGroup PDG_coreA2
   GlobalChainIndex 14
 }
 core_a.memB_bisr_inst.ShiftReg[0] {
   PowerDomainGroup PDG_coreA2
   GlobalChainIndex 15
 }
 .
.
.
 core_a.memB_bisr_inst.ShiftReg[14] {
   PowerDomainGroup PDG_coreA2
   GlobalChainIndex 29
 }
 toBisr_PDG_coreA2_REG {
   PowerDomainGroup PDG_coreA2
   GlobalChainIndex 30
 }
 core_a.memE_bisr_inst.ShiftReg[0] {
   PowerDomainGroup PDG_coreA1
   GlobalChainIndex 31
 }
 .
.
.
 core_a.memE_bisr_inst.ShiftReg[14] {
   PowerDomainGroup PDG_coreA1
   GlobalChainIndex 45
 }
 toBisr_PDG_coreA1_REG {
   PowerDomainGroup PDG_coreA1
   GlobalChainIndex 46
 }
 memC_bisr_inst.ShiftReg[0] {
   PowerDomainGroup PDG_Default
   GlobalChainIndex 47
 }
 .
.
.
 memC_bisr_inst.ShiftReg[14] {
   PowerDomainGroup PDG_Default
   GlobalChainIndex 61
 }
 memD_bisr_inst.ShiftReg[0] {
   PowerDomainGroup PDG_Default
   GlobalChainIndex 62
 }
 .
.
.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
      memD_bisr_inst.ShiftReg[14] {
        PowerDomainGroup PDG_Default
        GlobalChainIndex 76
      }
      toBisr_PDG_Default_REG {
        PowerDomainGroup PDG_Default
        GlobalChainIndex 77
      }
    }
    pdg_length {
      PDG_coreA2 31
      PDG_coreA1 16
      PDG_Default 31
    }
    flow tessent_shell
  }
```

The sub-dictionary pdg_length contains a key for each power domain group with a corresponding value for the BISR chain length. The Tcl command shown below can be used to return a list of power domain and BISR chain length value pairs from the configuration file:

```
SETUP>dict get [write_memory_repair_dictionary -return_dictionary] pdg_length
```

For the configuration file provided in this example, the Tcl command returns the following list:

```
PDG_coreA2 31 PDG_coreA1 16 PDG_Default 31
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_modelfile

Context: dft -edt, patterns -scan

Mode: analysis

Writes all internal states for a RAM or ROM gate into the file that you specify.

Usage

write_modelfile ***filename RAM/ROM_instance_name*** [-Replace]

Description

Writes all internal states for a RAM or ROM gate into the file that you specify.

The write_modelfile command writes, in the Tessent modelfile format, all the internal states for a RAM or ROM gate into a file.

The RAM and ROM internal states are identical to the initial values. If no initial values exist, the command reports an error condition.

Arguments

- ***filename***

  A required string that specifies the name of the file to which you want to write the internal states for the RAM or ROM gate. The information is written in Tessent modelfile format.

- ***RAM/ROM_instance_name***

  A required string that specifies the instance name of the RAM or ROM gate whose internal states you want to write. The command reports an error condition if the instance contains multiple RAM/ROM gates.

- -Replace

  An optional switch that replaces the contents of the file if the *filename* already exists.

Examples

The following example writes all the internal states of a RAM gate into a file for review:

```
add_write_controls 0 w1
set_system_mode analysis
create_patterns
write_modelfile model.ram /p1.ram
```

Related Topics

read_modelfile

set_ram_initialization

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_patterns

Context: patterns -ijtag, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Mode: analysis

Saves the current test pattern set in memory to a file in a specified format.

Usage

For patterns -scan and EDT Off

write_patterns ***pattern_filename*** [*format_switch*]
[-pattern_sets [ssn_continuity] [ssh_on_chip_compare] [ssh_loopback] [chain] [scan] [lvx_loop] [lvx_verification] [burnin_loop]]
[-TESTER_Statistics_report]
[-Replace]
[-PROcfile *procedure_filename*]
[-PARAMeter_file *parameter_filename*]
[-PARAMETER_List {*keyword_value_pair*...}]
[-PARALlel | -Serial]
[-BEgin {*pattern_number* | *pattern_name*}]
[-END {*pattern_number* | *pattern_name*}]
[-TAg *tag_name*]
[-CEll_placement {Bottom | Top | None}]
[-ENVironment]
[-TEST_Setup {ONLy | ONE | OFf | ONE_Per_file}]
[-TEST_End {ONLy | ONE | OFf}]
[-TEST_Payload]
[-SSN_Setup {ONLy | OFf | ONE_Per_file}]
[-SSN_End {ONLy | OFf | ONE_Per_file}]
[-ALL_Setup_only]
[-ALL_End_only]
[-EXternal]
[-SAmple [*integer*]]
[-IDDQ_file]
[-MODE_Internal | -MODE_External]
[-NAme_match | -POSition_match]
[-NOPadding | -PAD0 | -PAD1]
[-MAXloads *load_number*]
[-ADD_pattern_index_range_to_filename | -ADD_Load_range_to_filename]]
[-PATtern_size *integer*]
[-MEMory_size *size_in_KB*]
[-SCAN_Memory_size *size_in_KB*]
[-VERBose]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For patterns -scan and EDT On

write_patterns ***pattern_filename*** [*format_switch*]
[-pattern_sets [ssn_continuity] [ssh_on_chip_compare] [ssh_loopback] [chain] [scan] [lvx_loop] [lvx_verification] [burnin_loop]]
[-TESTER_Statistics_report]
[-Replace]
[-PROcfile *procedure_filename*]
[-PARAMeter_file *parameter_filename*]
[-PARAMETER_List {*keyword_value_pair...*}]
[-PARALlel | -Serial]
[-BEgin {*pattern_number* | *pattern_name*}]
[-END {*pattern_number* | *pattern_name*}]
[-TAg *tag_name*]
[-CEll_placement {Bottom | Top | None}]
[-ENVironment]
[-TEST_Setup {ONLy | ONE | OFf | ONE_Per_file}]
[-TEST_End {ONLy | ONE | OFf}]
[-TEST_Payload]
[-SSN_Setup {ONLy | OFf | ONE_Per_file}]
[-SSN_End {ONLy | OFf | ONE_Per_file}]
[-ALL_Setup_only]
[-ALL_End_only]
[-EXternal]
[-SAmple [*integer*]]
[-IDDQ_file]
[-MODE_Internal | -MODE_External]
[-PAD0 | -PAD1]
[-MAXloads *load_number*]
[-ADD_Pattern_index_range_to_filename | -ADD_Load_range_to_filename]]
[-PATtern_size *integer*]
[-MEMory_size *size_in_KB*]
[-SCAN_Memory_size *size_in_KB*]
[{-EDT_Bypass | -EDT_Single_bypass_chain} | -EDT_Internal | -EDT_External]
[-CHANNEL_OUTput_data {ON | OFF}]
[-VERBose]

For patterns -ijtag

write_patterns ***pattern_filename*** [*format_switch*]
[-pattern_sets *pattern_set_list*]
[-Replace]
[-PROcfile *procedure_filename*]
[-PARAMeter_file *parameter_filename*]
[-PARAMETER_List {*keyword_value_pair...}*]
[-TEST_Setup {ONLy | ONE | OFf | ONE_Per_file}]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

[-ENVironment]
[-TEST_End {ONLy | ONE | OFf}]
[-TEST_Payload]
[-ALL_Setup_only]
[-ALL_End_only]
[-NOPadding | -PAD0 | -PAD1]
[-VERBose]

For patterns -scan_retargeting

write_patterns ***pattern_filename*** [*format_switch*]
[-pattern_sets [ssn_continuity] [ssh_on_chip_compare]
[ssh_loopback [-CHain_test | -SCan_test]]
[chain] [scan][lvx_loop] [lvx_verification] [burnin_loop]]
[-TESTER_Statistics_report]
[-Replace]
[-PROcfile *procedure_filename*]
[-PARAMeter_file *parameter_filename*]
[-PARAMETER_List {*keyword_value_pair*...}]
[-PARALlel | -Serial]
[-BEgin {*pattern_number* | *pattern_name*}]
[-END {*pattern_number* | *pattern_name*}]
[-TAg *tag_name*]
[-CEll_placement {Bottom | Top | None}]
[-ENVironment]
[-TEST_Setup {ONLy | ONE | OFf | ONE_Per_file}]
[-TEST_End {ONLy | ONE | OFf}]
[-TEST_Payload]
[-SSN_Setup {ONLy | OFf | ONE_Per_file}]
[-SSN_End {ONLy | OFf | ONE_Per_file}]
[-ALL_Setup_only]
[-ALL_End_only]
[-EXternal]
[-IDDQ_file]
[-MODE_Internal | -MODE_External]
[-NAme_match | -POSition_match]
[-NOPadding | -PAD0 | -PAD1]
[-MAXloads *load_number*]
[-ADD_Pattern_index_range_to_filename |
-ADD_Load_range_to_filename]]
[-PATtern_size *integer*]
[-MEMory_size *size_in_KB*]
[-SCAN_Memory_size *size_in_KB*]
[-VERBose]

For Hybrid TK/LBIST Flow Parallel Patterns

write_patterns ***pattern_filename*** -Verilog -PARALlel -MODE_Internal [-Replace]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For Hybrid TK/LBIST Flow PatDB Files

write_patterns ***pattern_filename*** -PATDB
[-max_scan_load_unload_size *integer*] [-Replace]

For User-Defined Broadside Vector Data

write_patterns **-vector_set *vector_set***

Description

The write_patterns command formats the patterns in memory to your specification and writes them out to a file. You can use write_patterns to prepare for simulation or to provide final patterns for the automated test equipment (ATE). You can use write_patterns to save patterns in PatDB format, however you must direct the write_patterns command where to store the files.

The write_tsdb_data command stores patterns (in PatDB format) along with your design in the Tessent Shell Data Base (TSDB) for later use by the Tessent tools. You can specify the location of the TSDB with the set_tsdb_output_directory command and the tool manages the individual file locations within the TSDB structure.

You can press Ctrl-C on the command line to cancel the write_patterns operation without creating any files.

By default, any signal that is classified as unused for the pattern being written is removed from the pattern file, reducing the size of the pattern file or of the Verilog testbench. To disable this behavior, set the ALL_EXCLUDE_UNUSED keyword to 0.

With few exceptions, write_patterns simply formats existing patterns. You must correctly specify pins and timing before generating or retargeting patterns. For example, techniques to handle slow scan enable transitions are covered in "Delaying Clock Pulses in Shift and Capture to Handle Slow Scan Enable Transitions" in the *Tessent Scan and ATPG User's Manual*.

Note

If you write patterns in the presence of SSN, the data stream splits into multiple patterns, depending on the configuration of SSN. The number of patterns stored in the pattern file can differ from those reported after ATPG or chain test generation.

Multiprocessing Support for Writing Patterns

You can use multiple threads to speed up writing scan patterns in most cases. To do so, you must use the add_processors command to specify the threads.

Hybrid TK/LBIST Flow

You can use the write_patterns command to write out parallel patterns and PatDB files for designs containing hybrid TK/LBIST hardware. Refer to the *Hybrid TK/LBIST Flow User's Manual* for flow descriptions.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

IDDQ Fault Type and External Capture Options

If you are using external capture options with IDDQ faults for use with multiple fault types, the tool returns a warning under the following circumstances:

1. You run the set_external_capture_options command.
2. You then set the fault type to IDDQ.

Under these circumstances, the tool returns the following warning:

```
// Warning: The set_external_capture_options <option> command is not
// compatible with IDDQ patterns.
// Set_external_capture_options <option> is being ignored while writing
// these patterns.
```

The tool writes the external_mode IDDQ patterns and ignores the external capture options.

Parallel Patterns and the "Only" Switches

In general, the tool writes parallel patterns by default. However, to reduce certain pattern file sizes, the STIL and WGL output default is the equivalent of -serial patterns if you use one of the following "only" switches that write only setup or end procedures into the pattern file:

- -test_setup only
- -test_end only
- -ssn_setup only
- -ssn_end only
- -all_setup_only
- -all_end_only

This excludes scan chain definitions and any timeplates or procedures needed for load_unload and shift procedures. If you want to include this information, specify the -parallel option explicitly in this case.

Arguments

- ***pattern_filename***

  A required string that specifies a name for the saved test pattern file. If the filename ends with *.gz* or *.Z*, the tool automatically compresses the file with the GNU gzip utility. See set_gzip_options for more details.

  Note

  If you write patterns in the pattern database (PatDB) format, do not end the pattern filename with *.gz* or *.Z*, because the PatDB format is already compressed to a size smaller than achievable with the GNU gzip utility. The read_patterns command does not support the *.gz* or *.Z* files for the PatDB format.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- *format_switch*

  An optional switch that specifies a format for the saved test patterns. Save simulation patterns in the same format as the design netlist.

  ASIC vendor test data formats usually support only a single timing file. You can specify the test timing that each ASIC vendor requires by using different timing definition files for each format.

  Any time you save the pattern set in an ASIC vendor data format, also save the patterns using the write_tsdb_data command or in PatDB, ASCII, binary, STIL, or WGL format. This is in case you want to read in the patterns from an external file using the read_patterns command. Only PatDB, ASCII, binary, STIL, or WGL formatted test patterns can be read by the read_patterns command.

  File format options:

  -Ascii — A switch that writes test patterns in the standard ASCII format. The ASCII format includes the statistics report, environment settings, scan structure definition, scan chain functional test, scan test patterns, and the scan cell information. This is the default.

  The tool pads excess load and unload values with X values. You can override this behavior with the -nopadding switch.

  If you use the -external or the -begin and -end switches, the tool omits test coverage and fault information from the ASCII pattern set.

  For information about ASCII formats for EDT test patterns, refer to the "{-EDT_Bypass | - EDT_Single_bypass_chain} | -EDT_Internal | -EDT_External" option description.

  If SSN is present in your design, the contents of the ASCII pattern file are not the same as the SSN bus view that the tool writes if using STIL or WGL formats. In this case, the ASCII pattern file represents the internal channel view.

  -PATDB — An optional switch that directs the tool to write out patterns in a pattern database (PatDB) format.

  The PatDB file includes the pattern information, such as PRPG and MISR values, which the tool uses to retarget the core-level PDL using IJTAG to generate top-level patterns.

  In ATPG mode, use the PatDB format in place of the ASCII format. For hierarchical designs, use core-level patterns in the PatDB format for retargeting scan patterns to the top level. You can also use them for core-level diagnosis performed on the fail logs generated by reverse-mapping chip-level failures.

  To write out PatDB format and the design database for archival purposes, use the write_tsdb_data command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

If SSN is present in the design, the PatDB format is the default and only format available for retargeting.

This format is not available for use in the patterns -ijtag context.

Tip

If using the Two-Pin Serial Port for IJTAG, refer to the topic "How to Avoid Simulation Issues When Using the Two-Pin Serial Port Controller" in the *Tessent Shell User's Manual* .

-BInary — A switch that writes the test patterns in binary format. This option is not supported in the patterns -scan_retargeting context.

-CTl2005 — A switch that writes the patterns in CTL format. CTL is an extension to the STIL language; the STIL language is used in a more structured manner, and with the addition of CTL information. You can customize the output format by creating a parameter file. For more information, refer to the description of the -parameter switch later in this section and to "Parameter File Format and Keywords" on page 4997.

The tool writes out detailed scan control information for uncompressed scan chains created in dft -scan context. Use the import_scan_mode command to set up the scan mode to include the data in the output CTL file. The tool returns a warning if the detailed information is missing.

The detailed scan control information for uncompressed scan chains is only available in the patterns -scan context. You cannot set the design level to "chip" because the CTL file is intended for hierarchical scan stitching in third-party tools. See Example 7 for details.

If you want the ScanStructures/ScanChain/ScanMasterClock CTL construct to refer only to clocks used in a particular scan chain, you must enable the E8 DRC in setup mode by changing its handling from "Ignored" to any other severity. If the E8 rule check is not performed, the tool writes out all the scan clocks in each ScanMasterClock.

-PDl — A switch that writes the patterns out in PDL format. This format is supported only for the patterns -ijtag context. The tool converts the original PDL to iRead and iWrite commands on ScanRegisters and primary input and output pins. The scan operations needed to perform the ICL network reconfiguration are also explicitly described with iWrite and iApply commands. All PDL events associated with each pattern set are grouped into individual iProc wrappers and each iProc is named with the name of the corresponding pattern set.

Tip

If using the Two-Pin Serial Port for IJTAG, refer to the topic "How to Avoid Simulation Issues When Using the Two-Pin Serial Port Controller" in the *Tessent Shell User's Manual* .

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

-STIL2005 — A switch that writes the patterns in STIL format that conforms to IEEE Std. 1450.0 and 1450.1. You can customize the output format with a parameter file. For more information, see the description of the -parameter switch later in this section and "Parameter File Format and Keywords" on page 4997. Minimum typing for this switch is -STIL2.

-STil1999 — A switch that writes the patterns in STIL format that conforms to IEEE Std. 1450.0. You can customize the output format with a parameter file. For more information, see the description of the -parameter switch later in this section and "Parameter File Format and Keywords" on page 4997.

-Verilog — A switch that writes the patterns in Verilog format. The Verilog format contains pattern information and timing information from the timing file as a sequence of events. Saving patterns in Verilog format also causes the tool to write out a configuration (*.cfg*) file along with the testbench and *.vec* files.

For example:

```
Patterns.v
Patterns.v.0.vec
Patterns.v.1.vec
Patterns.v.cfg
Patterns.v.po.name
Patterns.v.chain.name
```

Also, if you specify the -verilog switch, the tool automatically generates the *<testbench_file_name>.info.dict* file. The create_patterns_specification command uses the information in this file to populate the Patterns/SimulationOptions/LowerPhysicalBlockInstances for each Patterns wrapper in the created PatternsSpecification. This ensures that the simulator compiles and uses the correct design view for each sub-block of the design.

By default, the module name created for Verilog testbenches consists of the specified design name followed by an underscore, followed by the name of the specified test procedure file followed by _ctl. The tool automatically converts all periods to underscores. For example:

*<design_name>*_*<test_proc_file_name>*_ctl

You can use the "TESSENT_PRAGMA simulation_only {begin | end}" directive in a proc file for test cycles that are only for simulation. Declare these directives using an annotate statement, such as in the following:

```
procedure test_setup =
  timeplate tp1 ;
  annotate TESSENT_PRAGMA simulation_only begin;
  cycle =
     …
  end;
  iCall …;
  …
  annotate TESSENT_PRAGMA simulation_only end;
  …
end;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Any test cycles between simulation_only begin and end directives, or any test cycles after a begin directive with no end directive, will be excluded from any non-simulation format (that is, non-Verilog) output, along with the annotations. The annotations are also not included in any Verilog output.

For more information about the annotate statement, refer to "Procedure Definition" in the *Tessent Shell User's Manual*.

If these statements appear in an incorrect order, such as having an "end" annotation before a "begin" annotation, or multiple "begin" annotations with no "end" annotations between them, the tool reports a W5 DRC error.

Caution

You must carefully manage these simulation_only directives to ensure that they do not cause miscompares in tester patterns.

For information specific to the chain name file see the description of the SIM_NO_CHAINNAME_FILE parameter keyword.

The configuration file contains the following lines:

```
Patterns.v.0.vec  740
Patterns.v.1.vec  820
```

(The number after the filename is the number of data lines to read from the *.vec* file.)

Even though the testbench is created with each write_patterns command, you only have to compile the design and testbench *.v* files once. You can use additional patterns just by copying the *.vec* and *.cfg* file to the location of the compiled simulation files.

The tool saves all timeplates currently loaded into the tool, not just the timeplates used by the current set of patterns. If you want to process a new procedure file with different timeplates, or if the netlist information is modified in any way, you must compile the new testbench before simulating.

The default Verilog testbench supports the following Verilog plusargs: STARTPAT, ENDPAT, CHAINTEST, END_AFTER_SETUP, SKIP_SETUP, and CONFIG. For more information, refer to "The Verilog Testbench" in the *Tessent Scan and ATPG User's Manual*. For information about support of STARTPAT and ENDPAT in SSN serial testbenches, refer to the section "STARTPAT and ENDPAT in SSN Serial Testbench" in "SSN Limitations" in the *Tessent Shell User's Manual*.

You can use the Verilog patterns to interface to a Verilog simulator such as Questa SIM. You can customize the output format with a parameter file. For more information, see the description of the -parameter switch later in this section and "Parameter File Format and Keywords" on page 4997.

-Wgl — A switch that writes the patterns in WGL format. The WGL format contains the waveform pattern information and any timing information from the timing file. You can use the WGL format patterns to generate test patterns in a variety of tester and simulator formats.

-Fjtdl — A switch that writes the test patterns in Fujitsu ASIC format: FTDL-E.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

-MItdl — A switch that writes the test patterns in Mitsubishi ASIC format: MITDL.

For MITDL format, multiple pulse timing must be a cyclical repetition of the first pulse. Consequently, multi-pulse and double-pulse timing in the procedure file only works in the MITDL output without an error if the timing fits the restrictions of the MITDL syntax. See "Timeplate Definition" in the *Tessent Shell User's Manual*.

-TItdl — A switch that writes the test patterns in Texas Instruments ASIC format: TDL91. You can customize the output format with the parameter file. For more information on the format of the parameter file, refer to "Parameter File Format and Keywords" on page 4997.

Note

If the design uses a single timeplate, the tool writes out patterns in TDL version 4.3 format. This removes the TIMING, END_TIMING, and SET_TIMING statements from the test pattern file. Also, the VERSION statement changes from 6.0 to 4.3. However, if the design uses multiple timeplates, the tool writes TDL version 6.0.

-TSTl2 — A switch that writes the patterns in Toshiba ASIC format: TSTL2.

-SVf — A switch that writes out the IJTAG patterns in Serial Vector Format. This switch is supported only for the patterns -ijtag context and you can only use it to write IJTAG pattern sets to files.

If the IJTAG patterns use more than one TAP interface or use non-TAP scan interfaces, the tool automatically generates the SVF file with a series of PIO commands instead of SDR, SIR, STATE, or TRST commands.

The TCK ratio of the pattern sets must be 1. (The TCK ratio is set by means of the -tck_ratio switch of the open_pattern_set command.)

If you use the -test_setup and -test_end switches of the write_patterns command, the tool converts the test setup procedure and the test end procedure to SVF and places them at the beginning and at the end of the pattern file, respectively.

Tip

If using the Two-Pin Serial Port for IJTAG, refer to the topic "How to Avoid Simulation Issues When Using the Two-Pin Serial Port Controller" in the *Tessent Shell User's Manual*.

- -TESTER_Statistics_report

  An optional switch that generates a report of pattern tester statistics, taking into account the other write_patterns options you supply. This report is identical to the report that the report_tester_statistics command produces. This switch is only valid for designs with SSN, and only for STIL1999, STIL2005, and serial Verilog patterns. If you use it with other pattern types, the tool reports an error.

- -Replace

  An optional switch that replaces the contents of an existing *pattern_filename*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -PROcfile *procedure_filename*

  An optional switch and string that specifies the name of the test procedure file that contains the non-scan event timing information. This option saves test patterns using the enhanced pattern output that includes timing information from the test procedure file.

  Note

  In the patterns -ijtag context, the tool returns an error if the specified procedure file contains a timeplate with the same name as an existing timeplate in an IJTAG pattern set.

  You can specify the procedure filename with this argument or specify it before using this command by running the read_procfile command. You cannot use this argument with the -ascii or -binary formats.

  For more details about the test procedure file, including which output format switches are valid in conjunction with the -procfile switch, refer to "Test Procedure File" in the *Tessent Shell User's Manual*.

- -pattern_sets [chain] [scan]

  An optional switch and keyword pair that specifies which pattern type (scan or chain) to write from the current session (in patterns -scan and patterns -scan_retargeting contexts). By default, the tool writes both scan and chain patterns. If you specify both chain and scan patterns, the tool writes chain patterns first regardless of the order of specification.

- -pattern_sets [ssh_loopback [-CHain_test | -SCan_test]]

  An optional switch and keyword pair that writes out SSH loopback patterns for the active SSH instruments in the current design (in patterns -scan and patterns -scan_retargeting contexts).

  Note

  Using the -chain_test or -scan_test option is unnecessary in the patterns -scan context because neither of them changes the generated ssh_loopback patterns. Their use is only meaningful in the patterns -scan_retargeting context.

  SSH loopback patterns are similar to chain patterns, with a random sequence replacing the scan data. This sequence is loaded and immediately unloaded from SSHs, bypassing any scan logic connected to those SSHs.

  The number of scan patterns may influence the programming of the SSH if you are retargeting patterns for more than one core type. For example, writing only chain patterns can result in different SSH programming than for scan patterns, or a range of patterns.

  To ensure the SSH loopback patterns use the same SSH configuration as the chain patterns, use the following command:

```
write_patterns -verilog -pattern_sets ssh_loopback -chain_test
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can replicate previous pattern range SSH configurations:

```
write_patterns -verilog -pattern_sets ssh_loopback -scan_test -begin
25 -end 29
```

- -pattern_sets [ssn_continuity]

  An optional switch and keyword pair that writes out SSN continuity patterns. SSN continuity patterns verify that the SSN datapath has been correctly integrated into the design. This option requires at least one SSH with a bus interface. If you use this option with the IJTAG streaming interface, the tool reports an error.

  Note

  To build a continuity pattern if you do not have any SSH on the datapath (for example, in a situation with SSN bus_in to bus_out through a multiplexer), you must create the pattern in the patterns -ijtag context, using the "create_ssn_continuity_patterns -ssn_datapath_ids" command with open_pattern_set and close_pattern_set.

- -pattern_sets [ssh_on_chip_compare]

  An optional switch and keyword pair that writes out SSH on-chip compare test patterns. This option requires at least one SSH with enabled on-chip compare hardware. If you have on-chip compare hardware, you must apply on-chip compare test patterns before you apply scan and chain test patterns to ensure proper functionality. Refer to the section "On-Chip Compare With SSN" in the *Tessent Shell User's Manual* for more information about this functionality. If you use this option with the IJTAG streaming interface, the tool reports an error.

- -pattern_sets [lvx_loop]

  An optional switch and keyword pair that creates LVX patterns. Refer to LVX Support in the *Tessent Shell User's Manual* for more information.

  The tool supports the lvx_loop pattern set only for STIL, WGL, CTL, and serial Verilog testbench formats, and reports an error if you specify a different format.

- -pattern_sets [lvx_verification]

  An optional switch and keyword pair that creates LVX verification patterns. LVX verification patterns are functionally identical to LVX loop patterns, but the output observed on the SSN bus is compared against an expected value to provide a pass/fail status. Refer to LVX Support in the *Tessent Shell User's Manual* for more information.

  The tool supports the lvx_verification pattern set only for STIL, WGL, CTL, and serial Verilog testbench formats, and reports an error if you specify a different format.

- -pattern_sets [burnin_loop]

  An optional switch and keyword pair that creates burn-in patterns. Refer to the set_burnin_options command examples to see a sample portion of a dofile using the "write_patterns -pattern_sets burnin_loop" command and set_burnin_options.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The tool supports the burnin_loop pattern set only for STIL, WGL, CTL, and serial Verilog testbench formats, and reports an error if you specify a different format.

For more information about burn-in patterns, refer to the section Burn-In Pattern Support in the *Tessent Shell User's Manual*.

- -pattern_sets *pattern_set_list*

  An optional switch and string for the patterns -ijtag context that specifies a Tcl list of pattern set names that were created using the open_pattern_set command. The tool writes patterns in the order specified in the *pattern_set_list*. If you do not use the -pattern_sets switch, the command writes all pattern sets in the order specified by the open_pattern_set command.

- -PARAMeter_file *parameter_filename*

  An optional switch and string pair that specifies an external parameter file to retrieve field names for use in certain pattern output formats. For more information about the format of the parameter file, refer to "Parameter File Format and Keywords" on page 4997.

- -PARAMETER_List {*keyword_value_pair...*}

  An optional switch and keyword/value pair that specifies one or more parameter file keywords and their values for use in certain pattern output formats. The keywords and values are delimited by spaces, and you must enclose the list in braces { }. For more information about parameter file keywords, refer to "Parameter File Format and Keywords" on page 4997.

  Note

  The tool does not support the -parameter_list switch for ASCII and binary pattern formats.

- -PARALlel

  An optional switch that saves all scan cells in parallel. This is the default.

  In designs with scan cells, only scan pins are active during the scan shift cycles. If the tool includes the state of each pin during each shift cycle, it may produce very large pattern files. Simulating the shift operations of these patterns might require a considerable amount of time if you use a different simulator. You can avoid these problems by saving all scan cells in parallel. The -parallel switch is compatible with all formats.

  Although -parallel is the default format, if you are writing STIL or WGL output using "only" options that write only setup or end procedures, the output is the equivalent of serial patterns. In these cases, to include scan chain definitions, timeplates, and procedures needed for load_unload and shift, you must specify -parallel explicitly.

  If the design contains EDT logic and you save parallel Verilog patterns, the tool saves the data with respect to the internal scan chains. These patterns are for simulation and cannot be saved as serial. But if you write patterns in parallel tester formats (for example, WGL), the tool saves the data with respect to the external scan channels.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The tool returns an AG11 DRC error if you attempt to write patterns in a parallel format other than Verilog or STIL using a multiple cycle shift procedure. Use a serial format instead.

- -Serial

  An optional switch that saves all scan cells in series. You can only use this switch with the -stil1999, -stil2005, -titdl (enhanced pattern output only), -wgl, and -verilog format type switches.

  Note

  You cannot use the -serial switch and the -scan_memory_size switch in the same command.

  If you use this switch together with the -verilog switch and the -parameter switch in the same command, the tool checks the specified parameter file for the SIM_SHIFT_DEBUG keyword. If this keyword exists and is set to 1, every scan chain in the Verilog pattern set contains a parallel observe of the entire scan chain for every shift, while serially shifting the test patterns. If this keyword is set to two or more, the parallel observe occurs for that many shifts, not all. This format is particularly useful for debugging scan chain problems because it enables you to locate the exact scan element causing a problem in a scan chain.

  Tip

  Use SIM_SHIFT_DEBUG serial Verilog patterns only for debugging scan chain problems. They typically take more time to create than other kinds of patterns and contain large amounts of data (so file sizes tend to be large). Also, because they simulate serially, the patterns take more time to simulate. Consider using the -pattern_sets chain, -begin, and -end switches to limit the size of the pattern file.

  For more information about the parameter file and the SIM_SHIFT_DEBUG keyword, refer to "Parameter File Format and Keywords" on page 4997.

- -BEgin {*pattern_number* | *pattern_name*}

  An optional switch and integer (*pattern_number*) or string (*pattern_name*) that specify the pattern to begin storing test patterns. You can use this switch to specify the first pattern in a range to include in the test patterns. The default pattern_number is 0. There is no default pattern_name.

  For Tessent FastScan and Tessent TestKompress, the tool assigns a pattern_number integer to each pattern during creation. Use the *pattern_name* string with -begin if the patterns have been previously written using the "write_patterns *pattern_filename* -tag *tag_name*" command in the current session or if you read external patterns with tag names.

  After ATPG or pattern retargeting, tag-based pattern names are not yet available and you must use -begin *pattern_number* to write patterns without tag names or together with the -tag *tag_name* to create the *pattern_name*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If used together with "-pattern_sets chain" in the same command, the -begin switch applies to chain test patterns only. If used together with "-pattern_sets scan" in the same command, the -begin switch applies to scan patterns only.

Only one -begin argument can be used per write_patterns command. To save multiple ranges of patterns, for example patterns 11 through 20 and 81 through 100, use the set_pattern_filtering command with the -range or -norange switches, or both, before you run the write_patterns command.

If you save only a portion of the internal patterns in the -ascii format, test coverage and fault information is not included.

In this example, the write_pattens command, using -begin *pattern_number*, creates an ASCII format pattern file beginning at pattern 25:

```
create_patterns
write_patterns pat1.ascii -ascii -begin 25
```

The following example demonstrates -begin *pattern_name*. First, "write_patterns -tag *tag_name*" creates the tag name for the patterns. Then, another write_patterns command uses the -begin and -end switches with the tag name to write a subset of the patterns:

```
create_patterns
write_patterns tagged_patterns.ascii -tag my_tag -ascii
write_patterns tagged_pat1.ascii -begin my_tag_25 -end my_tag_49
```

- -END {*pattern_number* | *pattern_name*}

  An optional switch and integer (*pattern_number*) or string (*pattern_name*) that specify the last pattern of a range to include in the test patterns. This argument is inclusive; the tool stores the pattern identified by the *pattern_number* or *pattern_name* you specify. The default pattern is the last pattern of the pattern set.

  Note

  Using the -begin and -end options together in the presence of SSN ensures that the specified range of patterns for each active core is stored. Using only the -end option has the same effect but with the starting pattern as pattern 0.

  For Tessent FastScan and Tessent TestKompress, the tool assigns a pattern_number integer to each pattern during creation. Use the *pattern_name* string with -begin if the patterns have been previously written using the "write_patterns *pattern_filename* -tag *tag_name*" command in the current session or if you read external patterns with tag names.

  After ATPG or pattern retargeting, tag-based pattern names are not yet available and you must use -end *pattern_number* to write patterns without tag names or together with the -tag *tag_name* to create the *pattern_name*.

  If used together with "-pattern_sets chain" in the same command, the -end switch applies to chain test patterns only. If used together with "-pattern_sets scan" in the same command, the -end switch applies to scan patterns only.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If you save only a portion of the internal patterns in the -ascii format, the tool does not include test coverage and fault information.

In this example, the write_patterns command, using -end *pattern_number*, creates an ASCII format pattern file ending at pattern 25:

```
create_patterns
write_patterns pat1.ascii -ascii -end 25
```

This example demonstrates using -end *pattern_name*. First, write_patterns -tag *tag_name* creates the tag name for the patterns. Then another write_patterns command uses the -begin and -end switches with the *tag name* to write a subset of the patterns:

```
create_patterns
write_patterns tagged_patterns.ascii -tag my_tag -ascii
write_patterns tagged_pat1.ascii -begin my_tag_25 -end my_tag_49
```

- -TAg *tag_name*

  An optional switch that adds a unique user-specified label, *tag_name*, to each pattern. All chain tests automatically have "chain" as the *tag_name* regardless of the *tag_name* given in the -tag switch. Because all tag names must be unique, "chain" is not a valid *tag_name*. If the *tag_name* supplied is not unique, the tool returns an error message and the command aborts.

  If the tool reads in patterns using the "set_pattern_source -external" command, the patterns must also be unique. The command aborts if the tool attempts to make two identically named patterns co-resident by any method. This uniqueness extends across both the internal and external pattern sets. You cannot have the same *pattern_name* in the internal and external sets.

- -CEll_placement Bottom | Top | None

  An optional switch and literal pair that controls the placement of the scan cell data in the ASCII pattern file. The literal choices:

  Bottom — A literal that places the scan data after the patterns, at the end of the file. This is the default.

  Top — A literal that places the scan data before the patterns, at the top of the file.

  None — A literal that excludes the scan data from the file.

- -ENVironment

  An optional switch that adds information about the current Tessent FastScan or Tessent TestKompress environment to the pattern file as comments. The information includes the identification stamp number, the environment settings, and the DRC rules. The information is the same as the report_environment and report_id_stamp commands display.

- -TEST_Setup {ONLy | ONE | OFf | ONE_Per_file}

  An optional switch and literal pair that specifies how the command saves the test_setup procedure. If you do not use this switch, the command writes one test_setup procedure at the start of each pattern file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The literal choices that control the test_setup procedure:

ONLy — A literal that saves the test_setup procedure as a separate file. This option does not work if you specify any of the following switches:

-maxloads	-test_end only	-pattern_sets
-begin	-end	-test_payload
-ssn_setup only	-ssn_end only	

ONE — A literal that applies only one test_setup procedure if both chain and scan test patterns are saved in one pattern file. If you save patterns in formats other than ASCII and binary, the tool saves a single test_setup procedure in the first file generated if you used the -maxloads switch with this "one" option.

OFf — A literal that saves patterns without including the test_setup vectors. Using this option may produce vectors that simulate incorrectly or produce failures on a tester.

ONE_Per_file — A literal that places one test_setup procedure at the start of each pattern file, even if you save multiple pattern files with one command by using the -maxloads, -memory_size, or -scan_memory_size switches. This is the default option.

- -TEST_End {ONLy | ONE | OFf}

  An optional switch and literal pair that specifies how the command saves the test_end procedure. If you do not use this switch, the tool applies one test_end procedure at the end of the pattern file. The literal choices that control the test_end procedure:

  ONLy — A literal that saves the test_end procedure as a separate file. This option does not work if you specify any of the following switches:

-maxloads	-test_setup only	-pattern_sets
-begin	-end	-test_payload
-ssn_setup only	-ssn_end only	

  ONE — A literal that applies one test_end procedure at the end of the pattern file after all pattern sets are done. This is the default option. If you also use -maxloads, the tool saves a test_end procedure at the end of the last generated pattern file.

  OFf — A literal that saves patterns without including the test_end vectors. For ASCII and binary patterns, this switch has no effect. Using this switch might produce vectors that do not simulate correctly or might produce failures if used on a tester.

- -TEST_Payload

  An optional switch that suppresses writing all setup and end procs. This is equivalent to specifying "-test_setup off -ssn_setup off -test_end off -ssn_end off" in contexts where the SSN proc switches are valid, or "-test_setup off -test_end off" in contexts where the SSN proc switches are not available.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -SSN_Setup {ONLy | OFf | ONE_Per_file}

  An optional switch and literal pair that specifies how the command writes the ssn_setup procedure. If you do not use this switch, the tool adds a single ssn_setup procedure per pattern file as needed. The literal choices that control the ssn_setup procedure:

  ONLy — A literal that specifies to write a file containing only the ssn_setup procedure. This option does not work if you specify any of the following switches:

-maxloads	-test_setup only	-test_end only
-test_payload	-ssn_end only	

  Caution

  If you write the ssn_setup and ssn_end procedures to separate files, you must ensure that the pattern parameters (-begin, - end, and -pattern_sets) are identical for the write_patterns command you use to write the actual payload patterns. If they are not, a failure results on the SSN bus output.

  For a complete description of the SSN pattern structure and how to apply SSN patterns on the tester if you use the -ssn_setup and -ssn_end options, refer to the topic "SSN Pattern Structure" in the *Tessent Shell User's Manual*.

  OFf — A literal that specifies to write files without an ssn_setup procedure.

  ONE_Per_file — A literal that specifies to write a single ssn_setup procedure per pattern file, if the call is required.

  Refer to Example 8, Example 9, and Example 10 in this topic for examples of the order to apply patterns on the tester if you use this argument.

- -SSN_End {ONLy | OFf | ONE_Per_file}

  An optional switch and literal pair that specifies how the command writes the ssn_end procedure. If you do not use this switch, the tool adds a single ssn_end procedure per pattern file as needed. The literal choices that control the ssn_end procedure:

  ONLy — A literal that specifies to write a file containing only the ssn_end procedure. This option does not work if you specify any of the following switches:

-maxloads	-test_setup only	-test_end only
-test_payload	-ssn_setup only	

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Caution

If you write the ssn_setup and ssn_end procedures to separate files, you must ensure that the pattern parameters (-begin, - end, and -pattern_sets) are identical for the write_patterns command you use to write the actual payload patterns. If they are not, a failure results on the SSN bus output.

For a complete description of the SSN pattern structure and how to apply SSN patterns on the tester if you use the -ssn_setup and -ssn_end options, refer to SSN Pattern Structure in the *Tessent Shell User's Manual*.

OFf — A literal that specifies to write files without an ssn_end procedure.

ONE_Per_file — A literal that specifies to write a single ssn_end procedure per pattern file, if the call is required.

Refer to Example 8, Example 9, and Example 10 in this topic for examples of the order to apply patterns on the tester if you use this argument.

- -ALL_Setup_only

  An optional switch that writes the test_setup and ssn_setup procedures into a single file. If SSN is not present in the design, the tool writes only the test_setup procedure. This option does not work if you specify any of the following switches:

-all_end_only	-test_setup	-test_end
-test_payload	-ssn_setup	-ssn_end

- -ALL_End_only

  An optional switch that writes the test_end and ssn_end procedures into a single file. If SSN is not present in the design, the tool writes only the test_end procedure. This option does not work if you specify any of the following switches:

-all_setup_only	-test_setup	-test_end
-test_payload	-ssn_setup	-ssn_end

  Note

  Do not use the -all_end_only switch with the -maxloads switch. If you use the -maxloads switch, the ssn_end procedure must be part of each payload file.

- -EXternal

  An optional switch that saves the current external pattern set to the specified *pattern_filename*. By default, the tool saves internal test patterns.

  External patterns saved in ASCII format do not include test coverage and fault information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -SAMple [*integer*]

  Tip

  This switch has been superseded by the set_pattern_filtering command, which has multiple sampling options.

  An optional switch and integer pair that specifies to save a sample of patterns for each pattern type instead of saving all the patterns. The optional integer specifies the number of patterns contained in each sample. If you do not include an integer, the default is to sample one pattern per type. This switch is not supported in the patterns -scan_retargeting context.

- -IDDQ_file

  An optional switch that creates an ASCII file that lists the cycle number, vector number, and measure time for each saved pattern that has an IDDQ measurement point. This IDDQ information file has the same name as the pattern file, with ".IDDQ" appended to the end. You can create it for all enhanced pattern output formats. This switch does not support ASCII or binary output formats.

- -MODE_Internal

  An optional switch that saves test patterns using the internal mode of the named capture procedures. Test patterns saved with this switch treat all internal clocks and internal primary inputs as directly controllable pins. This is the default if you write patterns in ASCII or binary format, or if you are in LBIST mode in the TK/LBIST hybrid flow.

  You can use the exclude_from_mode_internal_patterns pseudo_port attribute to exclude specific pseudo_ports from the pattern file or testbench. Once you set this attribute to true, the "write_patterns *pattern_filename* -mode_internal" command and switch excludes that specific set of pseudo_ports from the pattern file and testbench.

- -MODE_External

  An optional switch that specifies to write patterns using the external mode of the named capture procedures and with no user-defined internal pins. If you used the add_clocks or add_primary_inputs command to define one or more internal pins while setting up for ATPG, you need to use this switch to save patterns correctly. This is the default if you write patterns in tester formats (for example, WGL) or Verilog format.

  LBIST mode in the TK/LBIST hybrid flow does not support external mode patterns.

  Note

  If a capture procedure does not contain an external mode, the -mode_external switch uses the internal mode cycle information, but ignores internal primary input (PI) events. The switch also ignores internal pin events in scan procedures such as shift and load_unload.

  If you have internal cut points as pseudo_ports, you use the include_in_mode_external_patterns pseudo_port attribute to specify the internal cut points. Once you set this attribute to true, the "write_patterns *pattern_filename* -mode_external"

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

command writes a specific set of pseudo_ports and writes this mixed mode testbench with only a subset of the cut points.

- -NAme_match

  An optional switch that instantiates the device under test in the testbench using the name matching method. This switch is only available for designs that do not contain EDT. If you start with a Verilog netlist and do not restart with a flat model, the Verilog testbench uses the position matching method.

- -POSition_match

  An optional switch that places the device under test in the testbench using the position matching method. This switch is only available for designs that do not contain EDT.

- -NOPadding

  An optional switch that prevents the tool from padding shorter scan chains with X values for the input data in saved ASCII test patterns. This switch is only available for designs that do not contain EDT.

  If the design contains scan chains of differing lengths, the tool pads the patterns for the shorter chains with X values to represent unknown or don't care bits in the input data. Use this switch to eliminate X values in the input data due to short scan chains. This switch can only be used for ASCII test patterns.

  To load unpadded test patterns into a tool, use the read_patterns command.

- -PAD0 | -PAD1

  Optional switches that control the pad values in patterns for designs with no EDT.

  For designs that contain EDT logic, both -PAD0 and -PAD1 operate the same: the tools pads all EDT patterns with 0.

  For designs that do not contain EDT logic, these switches replace X values, representing unknown or don't care bits, with either a 1 or a 0 value in the saved test patterns. Options to pad patterns:

  -PAD0 — Replaces X values with 0 values.

  -PAD1 — Replaces X values with 1 values.

  This behavior is the same for STIL, WGL, and Verilog. If you do not specify -pad0 or -pad1, an X (which is "don't care") is used on shorter chains. For parallel load scan loading, it is also left as X. For serial loads in a Verilog testbench, the tool shifts in each bit one vector at a time so it has access to the value of the previous bit and updates the X with the force value from the previous vector. Because padding with X means "don't care", it is correct to use the previous force value in the place of that X.

  This switch is only valid for test patterns that support scan chain padding.

- -MAXloads *load_number*

  An optional switch and positive integer pair that specifies the maximum number of scan loads, *load_number*, to include in a pattern file. Much like the -begin and -end switches, the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

-maxloads switch automatically breaks up the pattern file into a series of pattern files with the specified number of scan loads in each.

Note

A single core may trigger a file split if you use the "-maxloads *load_number*" option in the presence of SSN. All other active cores may not have reached the *load_number* limit.

For example, if pattern creation results in 730 basic scan patterns, and the following command uses -maxloads 100:

```
write_patterns my_patterns -maxloads 100
```

This command saves the same sequence of files with the same pattern content as the following series of commands:

```
write_patterns my_patterns_1 -begin 0 -end 99
write_patterns my_patterns_2 -begin 100 -end 199
...
write_patterns my_patterns_7 -begin 600 -end 699
write_patterns my_patterns_8 -begin 700
```

In either case, the first file contains 100 chain and scan patterns; every other file contains 100 scan patterns; the last file contains the remaining patterns.

If the specified number of maxloads is greater than the number of chain test patterns, the first file may contain fewer scan patterns than the specified number.

You can also use -maxloads to break up a pattern subset you specify with the -begin and -end arguments in the same command. For example, here is the command to write patterns 50 through 99 to a sequence of files that each contain no more than 15 patterns:

```
write_patterns my_patterns -maxloads 15 -begin 50 -end 99
```

Caution

If you use the -maxloads option in the presence of SSN, you must apply the pattern sequence on the tester in the correct order. If you do not, mismatches occur on the SSN bus output.

For a complete description of the SSN pattern structure and how to apply SSN patterns on the tester if you use the -maxloads option, refer to SSN Pattern Structure in the *Tessent Shell User's Manual*.

If you use -maxloads, the tool automatically appends an underscore and sequence number to the name of each resultant pattern file, starting with number one (_1). If you have more chain test patterns than maxloads, the tool returns this warning:

```
//  Warning: The number of chain test patterns, 39, exceeds the
//           number of scan loads specified for a single file,
//           the first file will have extra scan loads.
//           Use a larger value for -maxloads, -memory_size, or
//           -scan_memory_size to correct this.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The -maxloads switch does not support the PatDB format and cannot be used with the -pattern_size switch.

- -ADD_Pattern_index_range_to_filename

  A boolean option that adds the pattern index range to the split pattern filename for the files generated with the -maxloads option. If you specify this option, the tool appends an underscore and the pattern index range (_0_999, _1000_1999) to the filename. By default, the tool appends an underscore and sequence number.

  The filenames reflect the range of patterns in the file. The filename has this pattern: temp_0_8.stil. Each file should have a maxloads number of patterns except the last.

  The first file contains a chain test and the scan test.

- -ADD_Load_range_to_filename

  A boolean option that adds the range of loads to the split pattern filename for the files generated with the -maxloads option. If you specify this option, the tool appends an underscore and the load range (_0_999, _1000_1999) to the filename. By default, the tool appends an underscore and sequence number.

  The filenames reflect the range of loads in the file. The filename has this pattern: temp_0_8.stil.

  The first file contains a chain test and the scan test.

- -PATtern_size *integer*

  An optional switch and integer pair that specifies the size of the memory array in the Verilog testbench and controls the size of the *.vec* data files that store the Verilog testbench vector data. You specify the *integer* in kilobytes.

  Note

  The -pattern_size switch is valid only for the -verilog format output.

  The default pattern size is 128 MB. However, the tool adjusts any size specified to hold a multiple of the largest pattern. For example, if the largest pattern requires 3 MB for one pattern, the file size is a multiple of 3 MB, resulting in a 126 MB pattern size. The maximum pattern size is 2147483646 KB (approximately 268 GB), and the tool returns an error if this size is exceeded.

- -MEMory_size *size_in_KB*

  An optional switch and integer pair that specifies the maximum size, *size_in_KB*, of available main tester memory, and sets this number as a limit on the size of the pattern data in a pattern file. Much like the -maxloads switch, the -memory_size switch automatically breaks up a large pattern file into a series of smaller pattern files that each contain no more than the specified number of kilobytes of pattern data. The tool assumes that one pattern value (0, 1, X, or Z) requires one byte on the tester.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If you specify this switch and the -scan_memory_size switch, the -memory_size switch specifies the kilobyte limit for only the non-scan pattern data. In this case, the tool ends each pattern file when the size of either the scan data or the non-scan data reaches its limit.

The number of vectors in each pattern file varies; this difference occurs because each file contains the vectors from the test_setup procedure and the vectors for each Tessent FastScan pattern it contains. Vectors for a single pattern are not split between pattern files.

Note

The tool does not support the -memory_size switch if you use SSN.

- -SCAN_Memory_size *size_in_KB*

  An optional switch and integer pair that specifies the maximum size, *size_in_KB*, of tester memory available for only scan pattern data, and sets this number as a limit on the size of this data in the pattern file. Much like the -maxloads switch, the -scan_memory_size switch automatically breaks up a large pattern file into a series of smaller pattern files that each contain no more than the specified number of kilobytes of scan pattern data. The tool assumes one scan pattern value (0, 1, X, or Z) requires one byte on the tester.

  If you specify this switch and the -memory_size switch, the latter specifies the kilobyte limit for only the non-scan pattern data. In this case, the tool ends each pattern file when the size of either the scan data or the non-scan data reaches its limit.

  Note

  You cannot use the -scan_memory_size switch and the -serial switch in the same command.

  The tool does not support the -scan_memory_size switch if you use SSN.

  The number of vectors in each pattern file varies; this difference occurs because each file contains the vectors from the test_setup procedure and the vectors for each Tessent FastScan pattern it contains. Vectors for a single pattern are not split between pattern files.

- -VERBose

  An optional switch that writes extra information messages about the files being saved to stdout or the tool's log file. The messages use this syntax:

  ```
  // Writing file_type filename
  ```

  where *file_type* is a phrase describing the file, and *filename* is the specified or generated filename. For example, *file_type* could be "STIL patterns," "WGL patterns," "Verilog testbench," or "Verilog vectors."

- {-EDT_Bypass | - EDT_Single_bypass_chain} | -EDT_Internal | -EDT_External

  Optional switches that specify how to save EDT test patterns. By default, Tessent TestKompress saves ASCII test patterns that define load data with respect to the input

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

channels of the EDT logic and unload data with respect to the core scan chains. The options for controlling the EDT patterns:

-EDT_Bypass — An optional switch used with the -binary format switch to save EDT bypass test patterns in binary format. EDT bypass test patterns are the decompressed version of the EDT test patterns. This option is only valid for EDT test pattern generation. For more information, see "Generation of Identical EDT and Bypass Test Patterns" in the *Tessent TestKompress User's Manual*.

Note

This option cannot be used if bypass chains are inserted in the design netlist before the EDT logic is generated.

-EDT_Single_bypass_chain — An optional switch used with the -binary format switch to save EDT bypass test patterns for the single bypass chain configuration. The test patterns are saved in binary format. This option is only valid for EDT test pattern generation. For more information, see "Dual Bypass Configurations" in the *Tessent TestKompress User's Manual*.

-EDT_Internal — An optional switch that saves EDT test pattern data for only the internal core scan chains to an ASCII file. This option is only valid for EDT test pattern generation. The test patterns contain core scan chain force and observe data except for X values observed on the output of the compactor due to X blocking or scan chain masking. For more information, see "Understanding Scan Chain Masking in the Compactor" in the *Tessent TestKompress User's Manual*.

Note

If you save the test patterns with the -edt_internal switch, they cannot be read back into the tool.

-EDT_External — An optional switch that saves EDT test pattern data with respect to the (compressed) input and output channels. This option is only valid for EDT test pattern generation. These test patterns are similar to WGL and STIL tester formats and must be resimulated before they can be saved out to an external test pattern file.

Note

If you use SSN, the EDT input and output channels connect directly to the SSH, and the tool saves the pattern data with respect to those internal locations.

- -CHANNEL_OUTput_data {ON | OFF}

  An optional switch and literal pair that enables or disables the channel output data computation. By default, the tool makes a determination about whether calculating the channel output data is beneficial. When this computation is enabled, the run time for this command may increase slightly, but the run time when writing patterns for retargeting decreases significantly for this database. You can safely disable this computation if you do not plan to use the pattern for retargeting.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -max_scan_load_unload_size *integer*

  An optional switch and value pair that specifies the number of LBIST patterns that have their scan chain data stored in the pattern database for diagnostic purposes. This option does not affect the PRPG/MISR/low-power seeds/signatures; the pattern database continues to store them for the entire pattern range. The default is 1024.

- **-vector_set *vector_set***

  A required switch and value pair that specifies a Tcl vector_set definition specified as the pattern set to write.

  Note

  If you use the default timing for IJTAG operations to obtain the vector sets, the timeplate for the written vector sets must match the IJTAG timeplate. Otherwise, simulation mismatches may occur.

  See "open_pattern_set" on page 1427 for timing diagrams.

  This switch is mutually exclusive with any other write_patterns command switch or option, except for the *format_switch*. The pattern source is only compatible with STI pattern formats, such as STIL, WGL, Verilog, and TITDL. See "Vector Data Callbacks" in the *Tessent Scan and ATPG User's Manual* for information on the contents of a Tcl vector_set.

  Tessent tools cannot read STIL and WGL patterns written with the -vector_set argument.

Examples

Example 1

The following example performs an ATPG run and saves only the first 15 test patterns in the Verilog format. It then writes the complete PatDB file with the TSDB database.

```
SETUP>set_system_mode analysis
ANALYSIS>create_patterns
ANALYSIS>write_patterns file1 -verilog timefile -end 14
ANALYSIS>write_tsdb_data -replace
```

Example 2

The following example shows the module name created in the enhanced Verilog output with the -procfile switch. If the design name is MAIN, and you run the following write_patterns command:

```
ANALYSIS>write_patterns pattern1.pat -procfile myproc -verilog -replace
```

the module name in the testbench is MAIN_myproc_ctl.

Example 3

The following example (in patterns -scan context) saves Verilog chain test patterns that contain a parallel observe of the entire scan chain for every shift, while serially shifting the chain test patterns. This is useful for debugging scan chain problems. The example assumes the parameter file, *test.param*, contains a "SIM_SHIFT_DEBUG 1;" line.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

**ANALYSIS>write_patterns chain_shift_debug_patt.v -verilog -pattern_sets chain -serial \
-parameter test.param**

Example 4

The following examples show the output of the -verbose switch:

ANALYSIS>write_patterns test_pat.v -verilog -replace -verbose

```
// Writing Verilog testbench test_pat.v
// Writing Verilog PO names test_pat.v.po.name
// Writing Verilog chain names test_pat.v.chain.name
// Writing Verilog vectors test_pat.v.0.vec
// Writing Verilog config file test_pat.v.cfg
// Writing Info Dictionary test_pat.v.info.dict
```

ANALYSIS>write_patterns test_pat.stil -stil -replace -verbose -maxloads 10

```
// Writing STIL patterns test_pat_0.stil
// Writing STIL patterns test_pat_1.stil
```

Example 5

The following example writes the test_setup procedure as a separate STIL file:

ANALYSIS>write_patterns test_setup.stil -stil -replace -test_setup only

Example 6

This example uses the -add_pattern_index_range_to_filename switch to add the pattern range to the split pattern filename. The maxloads is 10 and the pattern count is 29. The tool creates three files: temp_0_8.stil, temp_9_18.stil, and temp_19_28.stil. The first file only has nine scan patterns, instead of ten, because it contains one chain test pattern.

**ANALYSIS>write_patterns temp.stil -stil -verbose -maxloads 10 \
-add_pattern_index_range -replace**

```
//  command: write_patterns temp.stil -stil -verbose -maxloads 10
//               -add_pattern_index_range_to_filename -replace
//  Writing STIL pattern             temp_0_8.stil
//  Writing STIL pattern             temp_9_18.stil
//  Writing STIL pattern             temp_19_28.stil
```

Example 7

This example presents a dofile to stitch uncompressed chains and write a relevant CTL file, which you can use to convey scan control information to third-party tools. The E8 DRC is set to “note” instead of its default “ignore” so that it can report potential clock skew problems on the tester.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_context dft -scan
read_cell_library ../data/a1.lib
read_verilog ../data/d.v
set_current_design
add_clocks 0 clk
set_system_mode analysis
analyze_wrapper_cells
add_scan_mode int_mode
add_scan_mode ext_mode
analyze_scan_chains
insert_test_logic

set_system_mode setup
set_context patterns -scan
set_drc_handling e8 note
import_scan_mode ext_mode
set_system_mode analysis
create_patterns
write_patterns design_ext_mode.ctl -ctl2005
```

Example 8

The following example writes out an SSN pattern with the test_setup, ssn_setup, and payload patterns in separate STIL files. It includes an ssn_end procedure in each payload pattern file. The last payload pattern file written contains the test_end procedure.

Caution

If you write the ssn_setup and ssn_end procedures to separate files, you must ensure that the pattern parameters (-begin, - end, and -pattern_sets) are identical for the write_patterns command you use to write the actual payload patterns. If they are not, a failure results on the SSN bus output.

For a complete description of the SSN pattern structure and how to apply SSN patterns on the tester if you use the -ssn_setup and -ssn_end options, refer to "SSN Pattern Structure" in the *Tessent Shell User's Manual*.

```
ANALYSIS>write_patterns test_setup.stil -STIL2005 -test_setup only
ANALYSIS>write_patterns ssn_setup.stil -STIL2005 -ssn_setup only
ANALYSIS>write_patterns payload.stil -STIL2005 -test_setup off -ssn_setup off \
            -replace -maxloads 100
```

On the tester, you must apply the patterns in exactly the following order:

1. *test_setup.stil*
2. *ssn_setup.stil*
3. *payload_1.stil*
4. *payload_2.stil*

   …

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

5. *payload_<N>.stil*

Example 9

The following example writes out an SSN pattern with the test_setup, ssn_setup, ssn_end, test_end, and payload patterns in separate STIL files.

Caution

If you write the ssn_setup and ssn_end procedures to separate files, you must ensure that the pattern parameters (-begin, - end, and -pattern_sets) are identical for the write_patterns command you use to write the actual payload patterns. If they are not, a failure results on the SSN bus output.

For a complete description of the SSN pattern structure and how to apply SSN patterns on the tester if you use the -ssn_setup and -ssn_end options, refer to SSN Pattern Structure in the *Tessent Shell User's Manual*.

```
ANALYSIS>write_patterns test_setup.stil -STIL2005 -test_setup only
ANALYSIS>write_patterns ssn_setup.stil -STIL2005 -ssn_setup only
ANALYSIS>write_patterns ssn_end.stil -STIL2005 -ssn_end only
ANALYSIS>write_patterns test_end.stil -STIL2005 -test_end only
ANALYSIS>write_patterns payload.stil -STIL2005 -test_setup off \
            -ssn_setup off -ssn_end off -test_end off -replace -maxloads 100
```

On the tester, you must apply the patterns in exactly the following order:

1. *test_setup.stil*
2. *ssn_setup.stil*
3. *payload_1.stil*
4. *ssn_end.stil*
5. *payload_2.stil*
6. *ssn_end.stil*

   …

7. *payload_<N>.stil*
8. *ssn_end.stil*
9. *test_end.stil*

Example 10

The following example sets up Streaming-Through-IJTAG with the set_ssn_options command and writes out an SSN pattern with the test_setup, ssn_setup, ssn_end, test_end, and payload patterns in separate STIL files.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Caution

If you write the ssn_setup and ssn_end procedures to separate files, you must ensure that the pattern parameters (-begin, - end, and -pattern_sets) are identical for the write_patterns command you use to write the actual payload patterns. If they are not, a failure results on the SSN bus output.

For a complete description of the SSN pattern structure and how to apply SSN patterns on the tester if you use the -ssn_setup and -ssn_end options, refer to "SSN Pattern Structure" in the *Tessent Shell User's Manual*.

Tip

If using the Two-Pin Serial Port for IJTAG, refer to the topic "How to Avoid Simulation Issues When Using the Two-Pin Serial Port Controller" in the *Tessent Shell User's Manual* .

```
ANALYSIS>set_ssn_options -streaming_interface ijtag
…
ANALYSIS>write_patterns test_setup.stil -STIL2005 -test_setup only
ANALYSIS>write_patterns streaming_through_ijtag_ssn_setup.stil -STIL2005 -ssn_setup
only
ANALYSIS>write_patterns streaming_through_ijtag_ssn_end.stil -STIL2005 -ssn_end only
ANALYSIS>write_patterns streaming_through_ijtag_test_end.stil -STIL2005 -test_end only
ANALYSIS>write_patterns streaming_through_ijtag_payload.stil -STIL2005 \
            -test_setup off -ssn_setup off -ssn_end off -test_end off -replace -maxloads 100
```

On the tester, you must apply the patterns in exactly the following order:

1. *test_setup.stil*
2. *streaming_through_ijtag_ssn_setup.stil*
3. *streaming_through_ijtag_payload_1.stil*
4. *streaming_through_ijtag_ssn_end.stil*
5. *streaming_through_ijtag_ssn_setup.stil*
6. *streaming_through_ijtag_payload_2.stil*
7. *streaming_through_ijtag_ssn_end.stil*

   …

8. *streaming_through_ijtag_ssn_setup.stil*
9. *streaming_through_ijtag_payload_<N>.stil*
10. *streaming_through_ijtag_ssn_end.stil*
11. *streaming_through_ijtag_test_end.stil*

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

add_scan_groups

analyze_simulation_mismatches

get_write_patterns_options

read_patterns

read_procfile

report_measure_cycles

report_procedures

report_timeplates

report_write_patterns_options

set_chain_test

set_external_simulator

set_pattern_classification

set_write_patterns_options

write_procfile

Vector Creation and Modification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_pdl

Context: all contexts

Mode: setup, analysis

Writes out PDL iProcs associated with ICL modules to a file.

Usage

write_pdl ***output_file***
[-of_icl_modules *icl_modules*]
[-replace] [-silent]

Description

Use this command to write PDL iProcs currently in memory to a file. This can be useful when using flat models because the ICL and PDL are not stored in the flat model.

Tip

If the test_setup proc stored within your flat model is based on IJTAG, you need to export both the ICL (using the write_icl command) and the PDL (using write_pdl) along with the flat model (using write_flat_model) to be able to go back to setup mode and back to analysis with that flat model.

If you do not specify the -of_icl_module switch, the tool writes out the iProcs for all ICL modules with PDL loaded in memory. If you use the -of_icl_modules switch, the tool writes out the PDL associated with the specified ICL modules.

Arguments

- ***output_file***

  A required string to specify the name of the output file.

- -of_icl_modules *icl_modules*

  An optional switch and value pair to specify a collection of ICL modules or a Tcl list of ICL module name patterns. If you use this switch, the tool exports the PDL associated with the specified ICL modules to the output file. If you do not use this switch, the tool exports the PDL of all ICL modules having PDL in memory to the file.

- -replace

  An optional switch to overwrite an existing file. When you use this switch, the tool suppresses the error message normally generated if the file already exists.

- -silent

  An optional switch to suppress the following:

  - Errors generated if you call the command without the -of_icl_modules switch, and there is no ICL module having PDL in memory.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Errors if you specify the -of_icl_module switch for an object that is not an ICL module, is an empty collection, or the Tcl list of name patterns matches no ICL modules.
- Warnings if a specified ICL module has no PDL in memory.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_primary_inputs

Context: dft -scan, dft -test_points

Mode: setup, analysis

Writes primary inputs to the specified file.

Usage

write_primary_inputs ***filename*** [-Replace] [-All | *primary_input_pin*…]

Description

Writes primary inputs to the specified file.

The write_primary_inputs command writes a list of either all the primary inputs of a circuit or a specific list of primary inputs that you specify into a file where it can be reviewed.

This command is identical to the report_primary_inputs command except the data is written into a file.

Arguments

- ***filename***

  A required string that specifies the name of the file to which the tool writes the primary inputs.

- -Replace

  An optional switch that specifies for the tool to replace the contents of the file, if the file already exists.

- -All

  An optional switch that specifies to write all the primary inputs to the file. This is the default.

- *primary_input_pin*

  An optional repeatable string that specifies a list of primary input pins that you want to write to the file.

Examples

The following example writes all primary inputs to a file:

```
write_primary_inputs inputfile
```

Related Topics

add_primary_inputs

delete_primary_inputs

report_primary_inputs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_primary_outputs

Context: dft -scan, dft -test_points

Mode: setup, analysis

Writes primary outputs to the specified file.

Usage

write_primary_outputs ***filename*** [-Replace] [-All | *primary_output_pin*…]

Description

Writes primary outputs to the specified file.

The write_primary_outputs command writes a list of either all the primary outputs of a circuit or a specific list of primary outputs that you specify into a file where it can be reviewed.

This command is identical to the report_primary_outputs command except the data is written into a file.

Arguments

- ***filename***

  A required string that specifies the name of the file to which the tool writes the primary outputs.

- -Replace

  An optional switch that specifies for the tool to replace the contents of the file, if the file already exists.

- -All

  An optional switch that specifies to write all the primary outputs to the file. This is the default.

- *primary_input_pin*

  An optional repeatable string that specifies a list of primary output pins that you want to write to the file.

Examples

The following example writes all primary outputs to a file:

```
write_primary_outputs outputfile
```

Related Topics

add_primary_outputs

delete_primary_outputs

report_primary_outputs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_procedure_testbench

Context: dft -edt, patterns -scan

Mode: analysis

Writes a testbench for verifying correct operation of internal signals used in procedures.

Usage

write_procedure_testbench ***testbench_filename*** [-Replace]
[-PARAMeter *parameter_filename*] [-ALl | -CApture *capture_name…* | -LOad_unload]

Description

Writes a testbench for verifying correct operation of internal signals used in procedures.

The write_procedure_testbench command writes a Verilog testbench for verifying correct operation of internal signals used in procedures, including named capture procedures. The testbench verifies that if a procedure forces or pulses an internal signal, the value forced or pulsed is actually produced by the circuit when simulated in a Verilog simulator. (The expect values in the testbench are based on the statements in the procedure file that force or pulse internal pins.)

Note

The testbench written by this command is unrelated to the testbench written by the write_patterns command, which enables you to simulate and verify the ATPG patterns. Saving patterns is the last step of the ATPG process, whereas you should write a testbench to verify named capture procedures before you create patterns.

Determining from incorrect patterns where the mistakes are in the procedures can be difficult. This command helps you find the mistakes in the procedures, so you can correct them before they result in incorrect patterns.

Arguments

- ***testbench_filename***

  A required string that specifies the name of the file to which you want to write the testbench.

- -Replace

  An optional switch that replaces the contents of *testbench_filename* if the file already exists. By default, the command does not overwrite an existing file.

- -PARAMeter *parameter_filename*

  An optional switch and string pair that specifies an external parameter file to retrieve field names for use in certain output formats. For more information on the format of the parameter file, refer to the "Parameter File Format and Keywords" on page 4997.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -All

  An optional switch that specifies to write a testbench that simulates the test_setup procedure, followed by all named capture procedures simulated as load_unload/capture pairs. Values forced on internal signals in all procedures are used as expected responses. Named capture procedures that do not force values on internal signals are skipped. An error message is issued if none of the procedures force values on internal signals. This is the default.

- -CAPture *capture_name*

  An optional switch and repeatable string that specifies a list of named capture procedures to verify. In this case, the command writes a testbench that simulates the test_setup procedure, followed by the load_unload procedure with shift procedure, and then the specified named capture procedure(s). If you specify more than one named capture procedure, the testbench simulates the additional ones as additional load_unload/capture pairs. Each load_unload/capture pair is simulated twice. Values forced on internal signals in the named capture procedures are used as expected responses. An error message is issued if a specified named capture procedure does not force any values on internal signals.

- -LOAd_unload

  An optional switch that specifies to write a testbench that simulates the load_unload and shift procedures. This testbench simulates the test_setup procedure (if present), followed by the load_unload procedure with one shift. Values forced on internal signals in the load_unload and shift procedures are used as expected responses.

Examples

Example 1

The following example writes a Verilog testbench for verifying the functionality of the named capture procedure “cap1.”

write_procedure_testbench verilog_tb_cap1.v -capture cap1 -replace

Example 2

The following example assumes the pathname of the Questa SIM bin directory has previously been added to the PATH variable. From the shell, the example first creates a Verilog library named “proc_test_work,” then compiles the following three elements into the library:

- Source for the top level design
- Testbench written out in the preceding example
- Parts library

With these prerequisites met, the example simulates the testbench.

```
% vlib proc_test_work
% vlog -work proc_test_work counter_syn_scan.v verilog_tb_cap1.v \
      -v my_verilog_source/verilog_lib.v
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
** Note: (vsim-3812) Design is being optimized...
//                              ... Tool Banner ...
//
-- Compiling module mux21_macro
-- Compiling module sffr_macro
-- Compiling module lmmux
...
-- Scanning library file 'my_verilog_source/verilog_lib.v
-- Compiling UDP dff_r_err
-- Compiling UDP dff_r

Top level modules:
        counter_verilog_tb_cap1_v_ctl

vsim counter_verilog_tb_cap1_v_ctl -lib proc_test_work \
      -t 100ps -c -voptargs=+acc=npr -do "run -all"

...
# //
# Loading proc_test_work.counter_verilog_tb_cap1_v_ctl
# Loading proc_test_work.counter
# Loading proc_test_work.count4
...
# Loading proc_test_work.dff_r
# Loading proc_test_work.dff_err
# Loading proc_test_work.dff_p
# run -all
#         1850: inst_0/QI_nx7 simulated 0  expected 1
#         1850: Procedure: CAPTURE1 Event: force inst_0/QI_nx7 to 1 at 0
```

The last two lines indicate the procedure named “capture1” forces a 1 on the internal pin /inst_0/QI_nx7, but the value produced on the pin when simulated in the Verilog simulator is 0. This suggests the force statement is applying an incorrect value. The next example shows what the corresponding force statement (highlighted in bold) might look like in the test procedure file:

```
procedure capture capture1 =
   scan_group groupcnt ;
   mode internal =
      timeplate gen_tp1 ;
      cycle =
         force clear 0 ;
         force clock[0] 0 ;
         ...
         force clock[9] 0 ;
         force test_clk 0 ;
         force /inst_0/QI_nx7 1;
         force_pi ;
         measure_po ;
         pulse clock[0] ;
         ...
         pulse clock[4] ;
      end;
   end;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
      mode external =
         timeplate gen_tp1_long ;
         cycle =
            force clear 0 ;
            force clock[0] 0 ;
            ...
            force clock[9] 0 ;
            force test_clk 0 ;
            force_pi ;
            measure_po ;
            pulse clock[0] ;
            ...
            pulse clock[4] ;
         end;
      end;
   end; // capture1
```

Example 3

The following example shows what the message at the end of the Verilog simulation run looks like if there are no mismatches:

```
# run -all
# No error between simulated and expected patterns.
```

Related Topics

report_capture_procedures

report_procedures

report_timeplates

set_capture_procedures

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_procfile

Context: dft -edt, dft -scan, dft -test_points, patterns -ijtag, patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Writes existing procedure and timing data to the named test procedure file.

Usage

write_procfile ***proc_filename*** [-Replace] [-Full [-EXternal]] [-EXpand_iprocs]

Description

Writes existing procedure and timing data to the named test procedure file.

For complete information on test procedure file creation, syntax, and structure, refer to "Test Procedure File" in the *Tessent Shell User's Manual*.

The write_procfile command writes out existing procedure and timing data to the named test procedure file.

Arguments

- ***proc_filename***

  A required string that specifies the name of the file to which you want to write existing procedure and timing data.

- -Replace

  An optional switch that replaces the contents of the file if the ***proc_filename*** already exists.

- -Full

  An optional switch that causes the tool to parse the ATPG pattern list (if any) and create all needed non-scan procedures before writing the procedure file data. In Tessent FastScan and Tessent TestKompress, this option always creates a capture procedure.

  Note

  The -Full option can cause the write_procfile command to take more time if there are a large number of ATPG-generated patterns in the internal pattern list.

- -EXternal

  An optional switch that causes the -Full switch to use the external patterns specified by the read_patterns command instead of internally generated patterns. This switch behavior is similar to the -External switch behavior for the write_patterns command. The -External switch is only valid when using the -Full switch.

- -EXpand_iprocs

  An optional switch that replaces all iCall and iMerge statements within procedures with the cycles that represent the solution to these iCall statement. For iMerges, the tool replaces the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

entire iMerge with the cycles representing the solution to that iMerge. The start and end of the expanded cycles is marked by comments in the reported procedure or procedure file. Use this switch to when debugging E14 DRC violations.

Examples

The following example writes the existing procedure and timing data to the specified file:

```
write_procfile myprocfile.proc
```

Related Topics

add_scan_groups

read_procfile

report_procedures

report_timeplates

write_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_scan_order

Context: dft -scan

Mode: insertion

Writes out a ScanDEF file that describes the newly created scan chains by defining a number of segments. Each segment defines some part of the chain and describes one or more scan cells (instance path, scan in pin, and scan out pin) within that segment. The layout tool uses this information to determine which cells are interchangeable when optimizing the scan chain structure to meet timing constraints and minimize layout congestion.

Usage

write_scan_order ***filename*** [-Replace] [-use_escaping_rule {Verilog | Lefdef}]

Description

The write_scan_order command writes the scan segments into the SCANCHAINS section of the ScanDEF file, where each statement describes a single scan segment. ScanDEF is a subset of the DEF language. By default, when the tool successfully traces through a pre-existing scan segment in a closed-design entity (such as a library cell, celldefine, or physical block), it writes the cells that comprise that segment to the ORDERED list of the corresponding scan segment's statement.

The tool considers modules that have the "exclude_from_concatenated_netlist" attribute specified as hard macros and does not include them in the concatenated netlist. When writing the ScanDEF file, any traced scan elements from existing segments on these modules are omitted between the scanIn and scanOut sections of these segments and stop at the SI/SO boundary, as is done for a cell library.

Arguments

- ***filename***

  A required string that specifies the name of the DEF file.

- -Replace

  An optional switch that replaces the specified DEF file, if it exists.

- -use_escaping_rule {Verilog | Lefdef}

  An optional switch and literal pair that specifies whether to use the Verilog escaping mechanism or the LEF/DEF escaping mechanism for specifying special characters in strings/identifiers. The default is Verilog.

  In the Verilog escaping mechanism, you start the escaped identifier with a backslash character, "\", and terminate the escaped identifier with a white space. Neither the leading backslash character nor the terminating white space is considered to be part of the identifier.

  In the LEF/DEF escaping mechanism, you use the backslash character, "\", in front of any character that has a special meaning in LEF/DEF, in which case the special meaning of the character is ignored.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The escape character cannot be used with the pound sign, '#', in LEF/DEF.

Examples

Example 1

This example shows the information the tool reports with the report_scan_cells command and what it writes to the ScanDEF file.

report_scan_cells

```
Scan mode 'unwrapped' scan cells:
-----------------------------------------------------------------------------------------
        Chain   Group                          ShiftReg   Library            Clock    Clock
CellNo  Name    Name   Pathname               ID/CellNo  ModelName  ScanOut  Pinname  Polarity
-----------------------------------------------------------------------------------------
-       chain0  dummy  /ts_lockup_latchn_clkc0_intno15_i  -/-  latchN   clk1   (-)
0       chain0  dummy  /ud1                               -/-  sff  Q   clk1   (+)
1       chain0  dummy  /ud15                              -/-  sff  Q   clk1   (+)
2       chain0  dummy  /ud14                              -/-  sff  Q   clk1   (+)
3       chain0  dummy  /ud13                              -/-  sff  Q   clk1   (+)
4       chain0  dummy  /ud12                              -/-  sff  Q   clk1   (+)
5       chain0  dummy  /ud11                              -/-  sff  Q   clk1   (+)
6       chain0  dummy  /ud6                               -/-  sff  Q   clk1   (+)
7       chain0  dummy  /ud5                               -/-  sff  Q   clk1   (+)
-       chain1  dummy  /ts_lockup_latchn_clkc1_intno17_i  -/-  latchN   clk1   (+)
0       chain1  dummy  /ud10                              3/2  dff  Q   clk1   (-)
1       chain1  dummy  /ud9                               3/1  sff  Q   clk1   (+)
2       chain1  dummy  /ud8                               2/2  dff  QB  clk1   (+)
3       chain1  dummy  /ud7                               2/1  sff  Q   clk1   (+)
4       chain1  dummy  /ud4                               1/3  dff  QB  clk1   (+)
5       chain1  dummy  /ud3                               1/2  dff  Q   clk1   (+)
6       chain1  dummy  /ud2                               1/1  sff  Q   clk1   (+)

-----------------------------------------------------------------------------
```

The tool identifies three shift registers, as shown in the following report_shift_registers command output.

report_shift_registers -verbose

```
-----------------------------------------------------
             SequentialCell    Clock        Library
Id   Length  InstancePathName  Edge & Name  ModelName
-----------------------------------------------------
[1]  3       '/ud2'            + clk1       'dff'
             '/ud3'            + clk1       'dff'
             '/ud4'            + clk1       'dff'
[2]  2       '/ud7'            + clk1       'sff'
             '/ud8'            + clk1       'dff'
[3]  2       '/ud9'            + clk1       'sff'
             '/ud10'           + clk1       'dff'
```

Given this input, the following command generates the ScanDEF file below.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_scan_order results/scandef -replace

```
#
#    DESC: ScanDEF written by Tessent Shell on <>
#

VERSION 5.7 ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN A ;
UNITS DISTANCE MICRONS 1000 ;

SCANCHAINS 2 ;

- scan_segment_0
  + START tessent_persistent_cell_buf_extsi18_i Y
  + FLOATING
      ud5 ( IN SI ) ( OUT Q )
      ud6 ( IN SI ) ( OUT Q )
      ud11 ( IN SI ) ( OUT Q )
      ud12 ( IN SI ) ( OUT Q )
      ud13 ( IN SI ) ( OUT Q )
      ud14 ( IN SI ) ( OUT Q )
      ud15 ( IN SI ) ( OUT Q )
      ud1 ( IN SI ) ( OUT Q )
  + STOP ts_lockup_latchn_clkc0_intno15_i d
  # Partition constraints - clock domain: clk1(+);
  #                         chain type: core; scan mode(s): unwrapped
  + PARTITION partition_1  MAXBITS 8 ;


- scan_segment_1
  + START tessent_persistent_cell_buf_extsi19_i Y
  + ORDERED
      ud2 ( IN SI ) ( OUT Q )
      ud3 ( IN D ) ( OUT Q )
      ud4 ( IN D ) ( OUT QB )
  + ORDERED
      ud7 ( IN SI ) ( OUT Q )
      ud8 ( IN D ) ( OUT QB )
  + ORDERED
      ud9 ( IN SI ) ( OUT Q )
      ud10 ( IN D ) ( OUT Q )
  + STOP ts_lockup_latchn_clkc1_intno17_i d
  # Partition constraints - clock domain: clk1(+);
  #                         chain type: core; scan mode(s): unwrapped
  + PARTITION partition_1  MAXBITS 7 ;


END SCANCHAINS

END DESIGN
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

In this ScanDEF file, FLOATING and ORDERED lists separate regular and shift register flip-flops within the same domain, respectively. The tool writes each shift register in a separate ORDERED list.

Example 2

The following example shows an excerpt from a ScanDEF file written when power domains are present in the design.

write_scan_order

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
...

- scan_segment_2
  + START tessent_persistent_cell_default_PD11_buf_extsi34_i Y
  + FLOATING
      dffrs_p1 ( IN SI ) ( OUT Q )
      dffrs_p0 ( IN SI ) ( OUT Q )
      non_scan1 ( IN SI ) ( OUT Q )
  + STOP ts_default_PD11_lockup_latchn_clkc2_intno27_i d
  # Partition constraints - clock domain: clk2(+);
  #                         power domain island: PD1.1;
  #                         chain type: core;
  #                         scan mode(s): default
  + PARTITION partition_1  MAXBITS 3 ;


- scan_segment_3
  + START ts_default_PD41_lockup_latchp_clkc3_extsi35_i Q
  + FLOATING
      dff3_p7 ( IN SI ) ( OUT Q )
      dff3_p6 ( IN SI ) ( OUT Q )
      dff3_p5 ( IN SI ) ( OUT Q )
  + STOP tessent_persistent_cell_default_PD41_buf_extso12_i A
  # Partition constraints - clock domain: clk2(-);
  #                         power domain island: PD4.1;
  #                         chain type: core;
  #                         scan mode(s): default
  + PARTITION partition_2  MAXBITS 3 ;

...

- scan_segment_6
  + START tessent_persistent_cell_default_PD11_buf_intno17_i Y
  + FLOATING
      non_scan3 ( IN SI ) ( OUT Q )
  + STOP tessent_persistent_cell_default_PD11_buf_extso29_i A
  # Partition constraints - clock domain: clk2(-);
  #                         power domain island: PD1.1;
  #                         chain type: core;
  #                         scan mode(s): default
  + PARTITION partition_3  MAXBITS 1 ;


- scan_segment_7
  + START tessent_persistent_cell_default_PD11_buf_extsi38_i Y
  + FLOATING
      non_scan0 ( IN SI ) ( OUT Q )
  + STOP ts_default_PD11_lockup_latchn_clkc6_intno26_i d
  # Partition constraints - clock domain: clk(+);
  #                         power domain island: PD1.1;
  #                         chain type: core;
  #                         scan mode(s): default
  + PARTITION partition_4  MAXBITS 1 ;


- scan_segment_8
  + START ts_default_PD11_lockup_latchn_clkc6_intno26_i q
  + FLOATING
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
    non_scan2 ( IN SI ) ( OUT Q )
+ STOP tessent_persistent_cell_default_PD11_buf_extso28_i A
# Partition constraints - clock domain: clk(-);
#                         power domain island: PD1.1;
#                         chain type: core;
#                         scan mode(s): default
+ PARTITION partition_5  MAXBITS 1 ;
```

Note

The tool creates different PARTITION labels and assigns them based on clock and power domains.

Example 3

The following example shows the Tessent OCC scan segment excerpt from a ScanDEF file and includes both combinational and sequential logic.

It also shows a representation of three OCC scan segments, for which the -no_trace switch was defined. The scan segment container is represented as a hierarchical instance in the ScanDEF file, with no details about cells inside. This handling can apply to any scan segment.

Note

The tool writes the sequential length of an untraced segment using the BITS keyword.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
...

- scan_segment_0
  + START tessent_persistent_cell_0_buf_extsi59_i Y
  + FLOATING
      top_rtl5_tessent_occ_c1_c1_inst ( IN scan_in ) ( OUT scan_out ) ( BITS 3 )
  + STOP tessent_persistent_cell_0_buf_intno0_i A
  # Partition constraints -
  # clock domain: tessent_persistent_cell_shift_capture_clock/ \
  #   GCK_pport(+);
  # chain type: core;
  # scan mode(s): unwrapped
  + PARTITION partition_1  MAXBITS 3 ;


- scan_segment_1
  + START tessent_persistent_cell_0_buf_intno0_i Y
  + FLOATING
     top_rtl5_tessent_occ_c1_c2_inst ( IN scan_in ) ( OUT scan_out ) \
        ( BITS 3 )
     top_rtl5_tessent_occ_c1_c3_inst ( IN scan_in ) ( OUT scan_out ) \
        ( BITS 3 )
  + ORDERED
      top_rtl5_tessent_occ_c4_c4_inst/tessent_persistent_cell_scan_in_buf \
        ( IN A ) ( OUT Y ) ( BITS 0 )
      top_rtl5_tessent_occ_c4_c4_inst/occ_control/ShiftReg/U10 \
        ( IN A0 ) ( OUT Y ) ( BITS 0 )
      top_rtl5_tessent_occ_c4_c4_inst/occ_control/ShiftReg/U4 \
        ( IN A0 ) ( OUT Y ) ( BITS 0 )
      top_rtl5_tessent_occ_c4_c4_inst/occ_control/ShiftReg/FF_reg_2_ \
        ( IN D ) ( OUT Q )
      top_rtl5_tessent_occ_c4_c4_inst/occ_control/ShiftReg/U5 \
        ( IN A1 ) ( OUT Y ) ( BITS 0 )
      top_rtl5_tessent_occ_c4_c4_inst/occ_control/ShiftReg/FF_reg_1_ \
        ( IN D ) ( OUT QB )
      top_rtl5_tessent_occ_c4_c4_inst/occ_control/ShiftReg/U8 \
        ( IN A0 ) ( OUT Y ) ( BITS 0 )
      top_rtl5_tessent_occ_c4_c4_inst/occ_control/ShiftReg/U7 \
        ( IN A ) ( OUT Y ) ( BITS 0 )
      top_rtl5_tessent_occ_c4_c4_inst/occ_control/ShiftReg/FF_reg_0_ \
        ( IN D ) ( OUT Q )
      top_rtl5_tessent_occ_c4_c4_inst/occ_control/U39 \
        ( IN A0 ) ( OUT Y ) ( BITS 0 )
      top_rtl5_tessent_occ_c4_c4_inst/occ_control/U37 \
        ( IN A1 ) ( OUT Y ) ( BITS 0 )
      top_rtl5_tessent_occ_c4_c4_inst/occ_control/scan_out_reg \
        ( IN D ) ( OUT Q ) ( BITS 0 )
      top_rtl5_tessent_occ_c4_c4_inst/tessent_persistent_cell_scan_out_buf \
        ( IN A ) ( OUT Y ) ( BITS 0 )
  + STOP tessent_persistent_cell_0_buf_intno3_i A
  # Partition constraints - clock domain: sc(+) sc(-);
  #                         chain type: core;
  #                         scan mode(s): unwrapped
  + PARTITION partition_2  MAXBITS 9 ;

...
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_scan_setup

Context: dft -test_points

Mode: insertion

Writes out a dofile containing all necessary steps for scan insertion.

Usage

write_scan_setup **-prefix *filename_prefix*** [-replace] [-all_internal_clocks]

Description

Writes out a dofile containing all necessary steps for scan insertion.

The existing chain and sub-chain information is carried over from previous dofiles.

When you use this command for test point insertion in a post-scan design, the tool writes out the dofile with the test point flops. The name of the new dofile is *filename_prefix*_tp_flops.

Arguments

- **-prefix *filename_prefix***

  A required switch and string that specifies a name prefix for the scan setup files.

  The filenames are composed of this prefix followed by an underscore (_) and a descriptive file-specific suffix. You can include the pathname to an existing directory as a component of the *filename_prefix*, and the files are written to that location.

- -replace

  An optional switch that replaces the contents of each file that already exists.

- -all_internal_clocks

  An optional switch that specifies to write all internally defined clocks to the timeplate, to the ATPG dofile, and to the shift procedure.

Examples

The following example writes out a dofile and test procedure file that can be used from the dft -scan context to perform X-bounding and scan insertion on the design:

```
analyze_test_points
insert_test_logic
write_design -output netlist_with_test_points.v -replace
write_scan_setup -prefix scan_setup -replace
```

Related Topics

analyze_test_points

insert_test_logic

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_test_points

set_test_point_types

set_test_point_analysis_options

set_test_point_insertion_options

write_scan_setup

write_test_point_dofile

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_schematic_dofile

Context: all contexts

Modes: all modes

Writes out a dofile to recreate the current state of a schematic window from the command line.

Usage

write_schematic_dofile ***filename*** [-display *window_name*] [-replace]

Description

Writes a dofile containing the commands needed to recreate the Flat or Hierarchical Schematic window. When you run the dofile with the source command, if there is a mismatch between the schematic stored in the dofile and the loaded design, an error message is issued.

Arguments

- ***filename***

  A required string that specifies the name of the dofile you are writing.

- -display *window_name*

  An optional switch and literal that specifies the Tessent Visualizer schematic window to save to the dofile. The valid options for the *window_name* argument are as follows:

  flat_schematic — A literal that specifies the Flat Schematic window.

  hierarchical_schematic — A literal that specifies the Hierarchical Schematic window.

  ijtag_network — A literal that specifies the IJTAG Network window.

  icl_schematic — A literal that specifies the ICL Schematic window.

  By default, the Flat Schematic window is saved to the dofile.

- -replace

  An optional switch that specifies that an existing file is overwritten.

Examples

The following example writes a dofile that restores the objects currently displayed in the Hierarchical Schematic window of Tessent Visualizer:

```
write_schematic_dofile schemFlat.do -display hierarchical_schematic -replace
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_test_point_dofile

Context: dft -test_points

Mode: all modes

Writes out a dofile containing the test points.

Usage

write_test_point_dofile ***filename*** [-all | -control | -observe] [-tool_analyzed_only] [-replace]

Description

Writes out a dofile containing the test points.

Writes out a dofile consisting of add_control_points and/or add_observe_points commands that describe the currently identified test point locations. This dofile can be used to manually edit the list of test points and override the default selections prior to inserting the hardware. It can also be used to completely bypass the analysis step during a subsequent tool invocation.

Arguments

- ***filename***

  A required string that specifies the name of the file into which you want to write the test points information.

- -all | -control | -observe

  An optional switch that specifies the type of test point to write to the dofile. Choose one from the following:

  -all — Writes out both control and observe points.

  -control — Writes out only the control points.

  -observe — Writes out only the observe points.

- -tool_analyzed_only

  An optional switch that writes out only the tool generated only test points to the dofile. If you do not specify this switch, then the tool writes out both the tool-generated and the manually-added test points.

- -replace

  An optional switch that directs the tool to overwrite an existing *filename*.

Examples

The following example performs test point analysis and then writes out the dofile:

```
analyze_test_points
write_test_point_dofile my_test_points.dofile -replace
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

add_control_points

add_observe_points

analyze_test_points

insert_test_logic

report_test_points

set_test_point_analysis_options

set_test_point_insertion_options

set_test_point_types

write_scan_setup

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

write_tsdb_data

Context: patterns -scan, patterns -scan_retargeting

Mode: analysis

Populates the contents of the Tessent Shell Data Base (TSDB).

Usage

write_tsdb_data [-design_id *string*] [-pattern_id *string*]
[-max_scan_load_unload_size *integer*] [-replace] [-channel_output_data {on | off}]

Description

Writes the contents of the Tessent Shell Data Base (TSDB), which is a structured directory containing subdirectories and files. The subdirectory naming is designed to group related information for each specific view of each design block. The files in the TSDB include the flat model, the Tessent Core Description file, the PatDB pattern file, and the fault list.

Note

For different configurations, you must use different modes, that is each fault type and each configuration requires a different mode. For example, if you have high compression and low compression patterns, you must generate these patterns using different modes. Refer to the set_current_mode command.

When using this command, the fault list is not written out if pattern filtering is enabled with the set_pattern_filtering command.

LogicBIST Usage

Writes the <design_name>.lbist_mode_<mode_name> directories that contain all of the files needed to enable logicBIST test mode during PatternsSpecification processing. This enables diagnosis of the failures in the production flow. The files generated after LBIST fault simulation are used when you use IJTAG to generate the top-level patterns.

The command detects that it must create the *<design_name>.lbist_mode_<mode_name>* directory when the context is “patterns -scan” and at least one add_lfsrs command was specified in the setup mode. You can check whether the logicBIST mode is enabled using the “get_context -logic_bist” command.

ATPG and Pattern Retargeting Usage

Writes the <design_name>.atpg_mode_<mode_name> or <design_name>.atpg_retargeting_mode_<mode_name> directories. When performing ATPG and pattern retargeting in Tessent Shell, the following usage applies:

- If you invoke write_tsdb_data after core-level ATPG, the tool writes the files you specified. These files are needed for retargeting core-level patterns as well as for the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

patterns and the flat model needed for core-level diagnosis after reverse-mapping of chip-level failures.

- The TCD for the core is auto-loaded (if not already in memory) at the next level when performing pattern retargeting and add_core_instances is used to add the design core.
- For "patterns -scan_retargeting" context and performing core extraction, using write_tsdb_data writes out the TCD. No flat model/patterns/fault list are written out in this case. If it is chip level, this TCD is used for reverse-mapping of fails back to the cores, or for subsequently performing pattern retargeting without rerunning core extraction. See "Scan Pattern Retargeting" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- -design_id *string*

  An optional switch and value pair that specifies the *design_id* value to use when creating the logic_test_cores TSDB sub-directory.

  If you specify the *design_id* with the -design_id switch, then the tool uses this *design_id*. A *design_id* must only use a combination of the following characters:

  - Alphabetic, both uppercase and lowercase. For example: "abc", "ABC", "AbC", and so on.
  - Numeric. For example: "123".
  - Underscore. For example: "A_1".

  If you do not specify the *design_id*, then the tool uses the value of the design_id module attribute that is set on the current design module when this attribute has a non-null value. When you load a design with the read_design command, then the tool sets the design_id attribute when reading the design. Additionally, the tool also sets this attribute after elaboration of the current design to the value returned by get_context -design_id when the value of the *design_id* is null (the design was not loaded with read_design, but, rather, with read_verilog). If the design_id module attribute on the current design is still null by the time you use the write_tsdb_data command, then the tool uses the value returned by get_context -design_id.

  Do not confuse *design_id* with *mode_name* and *pattern_id* when creating the logic_test_core containers. The mode name is set using the set_current_mode command while in the setup mode of the -patterns scan context. The *pattern_id* is specified using the -pattern_id switch and is used to differentiate different pattern sets for a given mode name of a given design view. The *design_id* defines the design view used; this is typically used to differentiate the pre-layout versus post-layout design views so that you can create your post-layout patterns without losing the pre-layout copies.

- -pattern_id *pattern_id*

  An optional switch and value pair that specifies the *pattern_id* when saving the <design_name>_<mode_name>_<fault_type>[_<pattern_id>].faults.gz file. For a specific

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

fault type, you may have performed different simulations such as a different value of n-detect. For a given LogicBIST mode of operation, you can save multiple patDBs to the same *logic_test_cores* directory. For UDFM, you may have a different set of faults during different fault simulation runs. The optional pattern ID is a way to save the fault classification under different filenames.

Specifying a pattern ID changes the filename for the patDB; the tool adds the *pattern_id* string to the default patDB name for a given mode. See Example 2, which shows an example for LogicBIST but the filename changes apply to all modes/fault types.

- -max_scan_load_unload_size *integer*

  An optional switch and value pair that specifies the number of LBIST patterns that have their scan chain data stored in the pattern database for diagnostic purposes. This option does not affect the PRPG/MISR/low-power seeds/signatures; the pattern database continues to store them for the entire pattern range. The default is 1024.

- -replace

  An optional switch that suppresses the error that the tool normally reports when you rerun the command and the <*design_name*>[_<*design_id*>].*logic_test_core* (see "logic_test_cores" on page 4829) directory already exist. Instead of producing an error, the files inside the <*design_name*>[_<*design_id*>].*logic_test_core* directory are silently overwritten. If you run the command multiple times with different pattern_id values, the files that are independent of the pattern id exist because they were saved by the last invocation. These files include the following: <*design_name*>_<*mode_name*>.*flat.gz*, <*design_name*>_<*mode_name*>.*patdb*, and <*design_name*>_<*mode_name*>.*tcd.gz*.

  Multiple <*design_name*>_<*mode_name*>_<*fault_type*>[_<*pattern_id*>].*faults.gz* files can coexist in the directory.

- -channel_output_data {on | off}

  An optional switch and literal pair that turns on or off the channel output data computation. By default, the tool makes a determination about whether calculating the channel output data is beneficial. When this computation is enabled, the run time for this command may increase slightly, but the run time when writing patterns for retargeting decreases significantly for this database. You can safely disable this computation if you do not plan to use the TSDB for retargeting.

Examples

Example 1

In this example, the write_tsdb_data command is invoked after fault simulation is performed to store all of the files that are needed to generate signoff and manufacturing patterns and perform diagnosis once the chip is in production. The symbol <...> represents the path "./tsdb_outdir/logic_test_cores/mychip_pre_layout.logic_test_core". The "_pre_layout" string is appended to the design name in the "mychip_pre_layout.logic_test_core" name because the -design_id option was specified when the set_context command was issued.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_context patterns -scan -design_id pre_layout
<read_verilog and setup commands here>
set_current_mode normal

set_system_mode analysis
set_fault_type stuck
add_faults -all
set_random_patterns 100

simulate_patterns -source bist -store_patterns all
write_tsdb_data -replace
```

```
//  Writing <...>/mychip.lbist_mode_normal/mychip1_lbist.tcd.gz
//  Writing <...>/mychip.lbist_mode_normal/mychip1_lbist.flat.gz
//  Writing <...>/mychip.lbist_mode_normal/mychip1_lbist.patdb
//  Writing <...>/mychip.lbist_mode_normal/m8051_lbist_stuck.faults.gz
```

Example 2

The following example shows what the logic_test_core directory structure for the core "mychip" looks like when you first run write_tsdb_data.

```
tsdb_outdir/logic_test_cores
└── mychip.logic_test_core
    └── mychip.lbist_mode_lbist
        ├── mychip_lbist.flat.gz
        ├── mychip_lbist.patdb
        ├── mychip_lbist_stuck.faults.gz
        └── mychip_lbist.tcd.gz
```

Suppose you rerun fault simulation with a new LogicBIST switching threshold and declare the following pattern ID:

```
write_tsdb_data -pattern_id pwr_cfg_8
```

The logic_test_cores directory appears as follows:

```
tsdb_outdir/logic_test_cores
└── mychip.logic_test_core
    └── mychip.lbist_mode_lbist
        ├── mychip_lbist.flat.gz
        ├── mychip_lbist.patdb
        ├── mychip_lbist_pwr_cfg_8.patdb
        ├── mychip_lbist_stuck.faults.gz
        ├── mychip_lbist_stuck_pwr_cfg_8.faults.gz
        └── mychip_lbist.tcd.gz
```

Related Topics

write_core_description

write_patterns

write_flat_model

write_faults

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 10
Design Rule Checking

Design rule checks (DRC) are separated into categories based on the type of verification they perform.

Caution

The purpose of DRCs is to protect against patterns failing on silicon, and to preserve diagnosis accuracy and resolution. The Tessent tools enable you to downgrade some DRCs. You should only downgrade the tool handling of a DRC as a last resort, with a complete understanding of the hazards involved.

Design Rule Overview 3169
Scan Insertion Checking 3169
How to Troubleshoot Rules Violations 3170
Design Rules 3177
RAM Rules (A Rules). 3191
Assertion Rules (ASSERT Rules) 3205
BIST Rules (B Rules) 3211
BSCAN Rules 3218
Clock Rules (C Rules) 3222
Scan Cell Data Rules (D Rules) 3267
Pre-DFT Clock Rules (DFT_C Rules) 3280
Pre-DFT Scan Rules (DFT_S Rules) 3310
Extra Rules (E Rules) 3312
EDT Finder Rules (F Rules) 3336
Flattening Rules (FN, FP, and FG Rules). 3361
General Rules (G Rules). 3368
ICL Extraction Rules (I Rules) 3373
ICL Semantic Rules (ICL Rules) 3383
EDT Rules (K Rules) 3445
Procedure Rules (P Rules) 3466
Core Mapping for ATPG and Scan Pattern Retargeting Rules (R Rules) 3514
Scannability Rules (S Rules) 3539
Scan-Wrapper Rules (SW Rules) 3549
SSN ICL Rules (SSN_ICL Rules) 3554
SSN Retargeting Rules (SSN_R Rules) 3557
Scan Chain Trace Rules (T Rules) 3587
Power-Aware Rules (V Rules) 3610
Timing Rules (W Rules). 3623
X-bounding Rules (XB Rules) 3645

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Other DRC Messages 3647

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Design Rule Overview

The design rules are organized into functional categories.

Scan Insertion Checking 3169

How to Troubleshoot Rules Violations 3170

Rules Settings 3170
How to Turn on ATPG Analysis 3170
How to Debug DRC Violations in Tessent Visualizer 3170
How to Set the Level of Gate Data 3171
How to Set the Gate Information Type 3172
How to Report Gate Data 3173
Flattening Rule Violations 3176
Design Rule Output 3176

Scan Insertion Checking

Prior to scan insertion, the tool performs limited rule checks on the design as you switch from setup to analysis mode. Part of the checking it does is scannability checking.

For more information, refer to Scannability Rules (S Rules).

For primary clock inputs gated by other logic, a test procedure file describes the logic conditions that permit propagation of the clock signal through these gates. For uncontrollable clock circuitry, the tool can assist you in modifying your circuit by inserting test logic circuitry at these clock nodes whenever necessary. Refer to "Test Logic Insertion" in the *Tessent Scan and ATPG User's Manual* for details.

If you specify existing scan circuitry, or if you have a test procedure file that sets up conditions to allow some state elements to be scan candidates, the tool performs more extensive checking. After you add scan circuitry to your design and generate or write a test procedure file, you should go back to setup mode and specify this information. Then you can return to analysis mode and perform extensive rules checking before entering patterns -scan context.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

How to Troubleshoot Rules Violations

You need to troubleshoot your design to identify the cause of design rule violations.

Rules Settings 3170
How to Turn on ATPG Analysis 3170
How to Debug DRC Violations in Tessent Visualizer 3170
How to Set the Level of Gate Data 3171
How to Set the Gate Information Type 3172
How to Report Gate Data 3173
Flattening Rule Violations. 3176
Design Rule Output 3176

Rules Settings

Some rules enable you to specify how a violation is handled either error, warning, note, or ignore.

How to Turn on ATPG Analysis

The Atpg_analysis option to the set_drc_handling command provides full test generation analysis during rules checking for clock rules C1, C3, C4, C5, C6, some D rules and some E rules (such as E4, E5, E8, E10, E11 and E12). For the C1 rule, this analysis is enabled by default; for other rules, the tool does not perform the analysis unless you direct it to.

For example, assume that you select Atpg_analysis for clock rule C1 and the tool simulates a clock input to be X. The rule violation occurs when the analysis determines that it is, or may be, possible to turn the clock input on when all other defined clocks are off, and constrained pins are at their constrained values.

Note

The ATPG analysis rules checking process may require additional CPU time and memory.

How to Debug DRC Violations in Tessent Visualizer

When you run the analyze_drc_violation command, the tool automatically opens the relevant Tessent Visualizer schematic window or Text/HDL Viewer (if it is not already open). It sets gate reporting to match the type of rule violation that has occurred.

For more information about using Tessent Visualizer to debug DRC violations, see "DRC Browser" in the *Tessent Shell User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

analyze_drc_violation

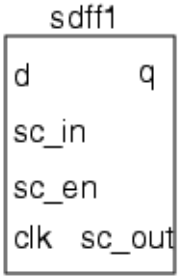
DRC Browser [Tessent Shell User's Manual]

How to Set the Level of Gate Data

The tools can report data at different levels; therefore, you should specify the level of information before you issue the report_gates command.

You do this with the set_gate_level command. Setting the gate level to design (the default) reports information at the design cell (library model) level. The following figure depicts a scannable-equivalent DFF cell library model at the design level.

Figure 10-1. Example of Design Level



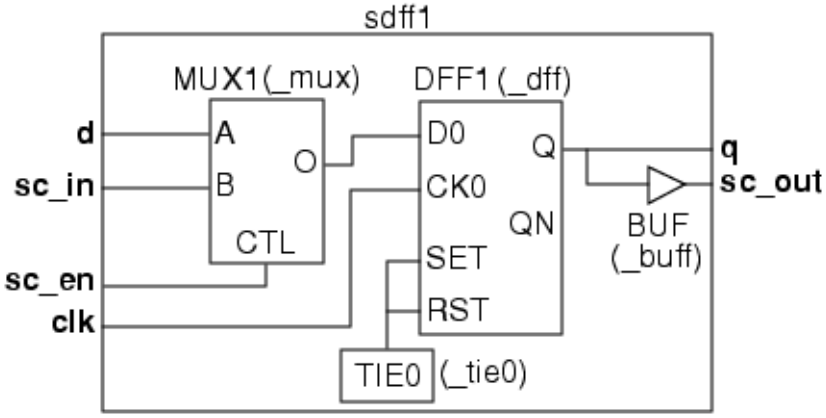
If the gate level is set to design, the report_gates command displays the following information:

```
instance_name        cell_type
    input_pin_name   I   (data)    pin_pathname ...
    input_pin_name   I   (data)    pin_pathname ...
                   . . .
    input_pin_name   O   (data)    pin_pathname ...
```

Setting the gate level to primitive reports information at the simulation gate level. The following figure depicts the sdff1 library model at the primitive level.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-2. Example of Primitive Level



If you set the gate level to primitive, the report_gates command displays the following information:

```
instance_name        (gate_id#)    gate_type
    input_pin_name   I   (data)    gate_id#-pin_pathname ...
    input_pin_name   I   (data)    gate_id#-pin_pathname ...
    . . .
```

input_pin_name 0 (data) gate_id#-pin_pathname...

How to Set the Gate Information Type

The set_gate_report command specifies the type of information that you want to appear when you report_gates data with the report_gates command. The multitude of options this command supports varies somewhat depending on which tool you are using.

The common usage of the set_gate_report command is as follows:

set_gate_report {**Normal** | **Trace** | **Error_pattern** | **TIe_value** | [**Drc_pattern** ***procedure_name*** {*time* | -All}] {**-COnstrain_value** [ON |OFf]}}

- The Trace option displays the values of the gates obtained during scan chain tracing. That is, this option displays data obtained on an error condition (not warning) during simulation of the **shift** procedure. You can use this option to help determine why a scan chain was not properly sensitized.
- The Error_pattern option displays the simulated values of the gate and its inputs, for the pattern (event) that had an error. This option displays such information as cell disturbances during the **load_unload** procedure or bus contention problems.
- The Normal option is the default. It displays only standard connectivity data.
- The Drc_pattern option displays an identified procedure's simulated gate values during the designated time. This option is similar to the Trace option, but is more versatile

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

because it allows access to the data obtained from simulation of any of the test procedures.

- The Parallel_pattern option displays simulated values for a selected pattern in the last simulation pass. A “pattern” is any time event that occurs during the test procedure. When the ATPG tool encounters problems in generating patterns, you can access the simulation data with this option.
- The -COnstrain_value {ON | OFf} option, when on, displays the simulated values that result from all natural tied gates, learned constant value non-scan cells, constrained pins, and constrained cells, the blocked flag (B), and the unused flag (UU).

How to Report Gate Data

If you encounter rules violations when you attempt to exit the Setup mode, you may need more information about specific gates in the design for troubleshooting purposes. While the violation message may give some information as to the location of the problem, you may need to track down the source of the problem by reporting on a sequence of gates in the design. The report_gates command is a very powerful command you can use to report on netlist data.

The following subsections show how to use report_gates to display various types of information for troubleshooting purposes. For more information on this command, refer to the report_gates reference page.

In Tessent Visualizer, you can report data on a selected gate by pressing Ctrl + R or right-clicking the gate or gates and selecting Report gates.

How to Report on a Specific Gate

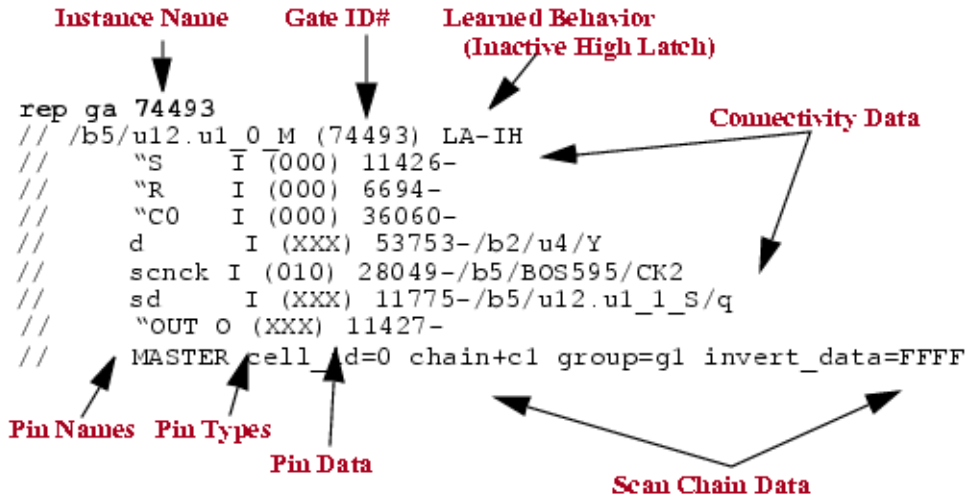
You can use the report_gates command to display information for selected gates, which you identify by either a gate index number or a pin pathname of a pin connected to the gate. This command reports the gate name, its gate type, and its connectivity to other gates. For example, to use report_gates in this manner, you could specify:

report_gates 74493

Figure 10-3 shows a report with primitive-level information for a gate with an ID number of 74493.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-3. Data Reported for a Specific Gate



How to Report on All Gates of a Specified Type

You can use report_gates to report on all gates of a specified type. The report_gates usage for this case is:

report_gates {-Type *gate_type*}...

The supported gate types are those listed as simulation primitives in the "Simulation Primitives of the Flattened Model" section of the *Tessent Scan and ATPG User's Manual*.

The following example shows how to report on all TIE0 gates.

report_gates -type tie0

```
//  ------------------------------------------------------
//  List of TIE0 gates
//  ------------------------------------------------------
//  /u1/inst__565_ff_d_0__dff (13)  TIE0
//      "OUT"  O  267-  266-
//  /u1/inst__565_ff_d_1__13 (14)  TIE0
//      "OUT"  O  269-  268-
//  /u1/inst__565_ff_d_2__13 (15)  TIE0
//      "OUT"  O  271-  270-
//  Total number of tie0 gates = 3
```

How to Report a Histogram of All Gate Types

You can use report_gates to show a distribution (histogram) of all gates in the design. To use report_gates in this manner, specify:

report_gates -type Histogram

The following example shows the type of data this command displays.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  -------------------------------------------------------
//  List of histogram of gates
//  -------------------------------------------------------
//  BUF=175  INV=30  AND=3  NAND=17  OR=7  NOR=5  XOR=2  LA=14
//  PI=12  PO=7  TIE0=7  MUX=7
```

How to Report on a Path Between Two Gates

You can also use report_gates to display information on the circuitry between two specified gates. To use report_gates in this manner, specify:

report_gates -path <gate1_ID#> <gate2_ID#>

How to Report on the First Input of a Gate

The report_gates command can display data on the gate connected to the first input of the previously reported gate. This lets you quickly and easily trace backward through circuitry. To use report_gates in this manner, first report on a specific gate and then enter:

back

The following example shows how to use report_gates and back to trace backward through the first input of the previously reported gate.

report_gates 26

```
//  /u1/inst__565_ff_d_1__13 (26)  BUF
//      "I0"   I  269-
//      "OUT"  O  268-  75-
```

back

```
//  /u1/inst__565_ff_d_1__13 (269)  LA
//      "S"    I  14-
//      "R"    I  145-
//      SCLK   I  4-/clk
//      D      I  265-/u1/_g32/X
//      ACLK   I  2-/scan_mclk
//      SDI    I  20-/u1/inst__565_ff_d_0__dff/Q2
//      "OUT"  O  26-  27-
```

back

```
//  /u1/inst__565_ff_d_1__13 (14)  TIE0
//      "OUT"  O  269-  268-
```

How to Report on the First Fanout of a Gate

Similar to tracing backward through circuitry, you can also trace forward through the first fanout of the previously reported gate. To use report_gates in this manner, first report on a specific gate and then enter:

forward

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following example shows how to use report_gates and forward to trace forward through the first fanout of the previously reported gate.

report_gates 269

```
//  /u1/inst__565_ff_d_1__13 (269)  LA
//      "S"    I  14-
//      "R"    I  145-
//      SCLK   I  4-/clk
//      D      I  265-/u1/_g32/X
//      ACLK   I  2-/scan_mclk
//      SDI    I  20-/u1/inst__565_ff_d_0__dff/Q2
//      "OUT"  O  26-  27-
```

forward

```
//  /u1/inst__565_ff_d_1__13 (26)  BUF
//      "I0"   I  269-
//      "OUT"  O  268-  75-
```

forward

```
//  /u1/inst__565_ff_d_1__13 (268)  LA
//      "S"    I  14-
//      "R"    I  145-
//      BCLK   I  1-/scan_sclk
//      "D0"   I  26-
//      "OUT"  O  24-  25-
```

Flattening Rule Violations

If you encounter flattening rule violations, you can use the report_flattener_rules command to display either a summary of the flattening rule violations or the data for a specific violation, for troubleshooting purposes.

In addition, you can use the set_flattener_rule_handling command to change the handling of the net, pin, and gate rules.

Design Rule Output

Many error messages provide a line number and filename to help you resolve the error condition. However, some error messages may relate to problems with internally generated data and do not provide a line number and filename.

For example, a syntax error (P1) in a test procedure file would produce the following message:

```
The following occurred at line 10 in file testproc
Syntax error in line number 10. (P1-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Design Rules

Each DRC class has multiple DRCs.

RAM Rules (A Rules) 3191
A1 3191
A2 3192
A3 3193
A4 3193
A5 3194
A6 3195
A7 3195
A9 3196
A10 3197
A11 3197
A12 3198
A13 3199
A14 3200
A15 3200
A16 3201
A17 3201
A18 3202

Assertion Rules (ASSERT Rules) 3205
Assertion Values and the Enforcement of Assertions 3205
ASSERT1 3205
ASSERT2 3207
ASSERT3 3209

BIST Rules (B Rules) 3211
B2 3211
B4 3211
B5 3212
B6 3212
B7 3214
B17 3216

BSCAN Rules 3218
BSCAN1 3218
BSCAN2 3219
BSCAN3 3219
BSCAN4 3219
BSCAN5 3220
BSCAN6 3220
BSCAN7 3220
BSCAN8 3220
BSCAN9 3221

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Clock Rules (C Rules) 3222
Clock Terminology 3222
The ATPG Analysis Option 3227
C1 3227
C2 3232
C3 3233
C4 3239
C5 3244
C6 3246
C7 3248
C8 3249
C9 3250
C10 3252
C16 3253
C17 3254
C19 3255
C21 3257
C22 3257
C23 3260
C24 3261
C25 3262
C26 3263
C27 3265
Scan Cell Data Rules (D Rules) 3267
D1 3267
D2 3271
D3 3272
D4 3273
D5 3274
D6 3275
D7 3276
D8 3277
D12 3278
Pre-DFT Clock Rules (DFT_C Rules) 3280
Simulation Contexts for Pre-DFT DRCs 3280
DFT_C1 3284
DFT_C2 3287
DFT_C3 3289
DFT_C4 3290
DFT_C5 3290
DFT_C6 3291
DFT_C7 3293
DFT_C8 3294
DFT_C9 3295
DFT_C10 3299

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DFT_C11 3306
DFT_C12 3307
DFT_C13 3308

Pre-DFT Scan Rules (DFT_S Rules) 3310
DFT_S1 3310

Extra Rules (E Rules) 3312
E1 3312
E2 3313
E3 3313
E4 3314
E5 3317
E6 3319
E7 3320
E8 3321
E9 3322
E10 3322
E11 3327
E12 3328
E13 3328
E14 3329
E15 3333
E16 3333

EDT Finder Rules (F Rules) 3336
F3 3336
F4 3337
F5 3337
F6 3338
F7 3340
F8 3340
F9 3346
F10 3347
F11 3349
F12 3350
F13 3350
F14 3351
F15 3352
F16 3352
F17 3353
F18 3353
F19 3354
F20 3355
F21 3357
F22 3357

Flattening Rules (FN, FP, and FG Rules) 3361

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

FN1 3362
FN2 3362
FN3 3362
FN4 3362
FN5 3363
FN6 3363
FN7 3363
FN8 3363
FN9 3364
FP1 3364
FP2 3364
FP3 3364
FP4 3364
FP5 3365
FP6 3365
FP7 3365
FP8 3365
FP9 3366
FP10 3366
FP11 3366
FP12 3366
FP13 3366
FG3 3367
FG7 3367
FG8 3367

General Rules (G Rules) 3368
G1 3368
G2 3368
G3 3369
G4 3369
G5 3369
G6 3370
G7 3370
G8 3370
G9 3371
G10 3371
G11 3372
G12 3372

ICL Extraction Rules (I Rules) 3373
I1 3373
I2 3375
I3 3376
I4 3377
I5 3378
I6 3379

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

I7 3381
ICL Semantic Rules (ICL Rules) 3383
ICL1 3387
ICL2 3388
ICL3 3388
ICL4 3389
ICL5 3389
ICL6 3389
ICL7 3389
ICL8 3390
ICL9 3390
ICL10 3390
ICL11 3391
ICL12 3391
ICL13 3391
ICL14 3391
ICL15 3392
ICL16 3392
ICL17 3392
ICL18 3392
ICL19 3393
ICL20 3393
ICL21 3393
ICL22 3394
ICL23 3394
ICL24 3394
ICL25 3394
ICL26 3395
ICL27 3395
ICL28 3395
ICL29 3395
ICL30 3396
ICL31 3396
ICL32 3396
ICL33 3396
ICL34 3397
ICL35 3397
ICL36 3397
ICL37 3398
ICL38 3398
ICL39 3398
ICL40 3399
ICL41 3399
ICL42 3399
ICL44 3399

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL45 3400
ICL48 3400
ICL49 3400
ICL50 3401
ICL51 3401
ICL52 3402
ICL53 3402
ICL54 3402
ICL55 3402
ICL56 3403
ICL57 3403
ICL58 3404
ICL59 3404
ICL60 3405
ICL61 3405
ICL62 3405
ICL63 3405
ICL64 3406
ICL65 3406
ICL66 3407
ICL67 3407
ICL68 3407
ICL70 3408
ICL71 3408
ICL75 3411
ICL76 3411
ICL77 3412
ICL78 3412
ICL79 3413
ICL80 3413
ICL81 3417
ICL82 3418
ICL83 3418
ICL84 3418
ICL85 3419
ICL86 3419
ICL87 3419
ICL88 3419
ICL89 3420
ICL90 3421
ICL91 3421
ICL92 3422
ICL93 3422
ICL94 3422
ICL95 3422

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL96 3423
ICL97 3423
ICL98 3423
ICL99 3424
ICL100 3424
ICL101 3424
ICL102 3425
ICL103 3425
ICL104 3425
ICL105 3426
ICL106 3427
ICL107 3427
ICL108 3427
ICL109 3428
ICL110 3428
ICL111 3428
ICL112 3428
ICL113 3429
ICL114 3429
ICL115 3429
ICL116 3430
ICL117 3430
ICL118 3430
ICL119 3431
ICL120 3431
ICL121 3431
ICL122 3432
ICL123 3432
ICL124 3432
ICL125 3433
ICL126 3433
ICL127 3434
ICL128 3434
ICL129 3434
ICL130 3435
ICL132 3435
ICL133 3436
ICL134 3436
ICL135 3437
ICL136 3438
ICL137 3439
ICL138 3439
ICL139 3440
ICL140 3440
ICL141 3442

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL142 3443
ICL143 3443
ICL144 3443

EDT Rules (K Rules) 3445
K1 3446
K2 3446
K3 3446
K4 3447
K5 3447
K6 3447
K7 3449
K8 3449
K9 3449
K10 3451
K11 3451
K12 3452
K13 3452
K14 3452
K15 3453
K16 3458
K17 3458
K18 3458
K19 3459
K20 3460
K21 3461
K22 3461
K23 3463
K24 3463
K25 3464

Procedure Rules (P Rules) 3466
P1 3468
P2 3469
P3 3469
P4 3470
P5 3470
P6 3470
P7 3471
P8 3471
P9 3472
P10 3472
P11 3472
P12 3473
P13 3474
P14 3474
P15 3475

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

P16 3475
P17 3475
P18 3476
P19 3476
P20 3476
P21 3477
P22 3477
P23 3478
P24 3478
P25 3478
P26 3479
P27 3479
P28 3480
P29 3480
P30 3481
P31 3481
P32 3482
P33 3482
P34 3483
P35 3483
P36 3483
P37 3484
P38 3484
P39 3484
P40 3485
P41 3485
P42 3485
P43 3486
P44 3486
P45 3487
P46 3487
P47 3488
P48 3488
P49 3488
P50 3489
P51 3489
P52 3489
P53 3490
P54 3491
P55 3491
P56 3492
P58 3492
P59 3492
P60 3493
P61 3494

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

P62 3494
P63 3495
P64 3495
P65 3495
P66 3496
P70 3496
P71 3497
P72 3497
P73 3497
P74 3498
P75 3498
P76 3499
P77 3500
P78 3500
P79 3500
P80 3501
P81 3502
P82 3502
P83 3503
P84 3504
P85 3504
P86 3504
P87 3505
P88 3505
P89 3506
P90 3506
P91 3507
P92 3507
P93 3508
P94 3508
P96 3511
P97 3513

Core Mapping for ATPG and Scan Pattern Retargeting Rules (R Rules). 3514

R1 3515
R2 3515
R3 3515
R4 3516
R5 3516
R6 3517
R7 3518
R8 3518
R9 3521
R10 3522
R11 3523
R12 3524

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

R13 3525
R14 3525
R15 3527
R16 3530
R17 3532
R18 3532
R19 3533
R20 3534
R21 3534
R22 3535
R23 3536
R27 3536
R28 3537

Scannability Rules (S Rules) 3539
S1 3540
S2 3542
S3 3542
S4 3543
S5 3544
S6 3545
S7 3546
S8 3547

Scan-Wrapper Rules (SW Rules) 3549
SW1 3549
SW2 3550
SW3 3550
SW4 3550
SW5 3551
SW6 3552
SW7 3552

SSN ICL Rules (SSN_ICL Rules) 3554
SSN_ICL2 3554
SSN_ICL3 3555

SSN Retargeting Rules (SSN_R Rules) 3557
SSN_R1 3557
SSN_R2 3558
SSN_R3 3558
SSN_R4 3559
SSN_R5 3562
SSN_R6 3564
SSN_R7 3565
SSN_R8 3567
SSN_R9 3570
SSN_R10 3571

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SSN_R11 3573
SSN_R12 3576
SSN_R13 3578
SSN_R15 3579
SSN_R16 3580
SSN_R17 3581
SSN_R18 3582
SSN_R19 3584
SSN_R20 3585
SSN_R21 3586
SSN_R23 3586

Scan Chain Trace Rules (T Rules) 3587
T1 3588
T2 3588
T3 3589
T4 3593
T5 3597
T6 3599
T8 3600
T9 3600
T10 3601
T11 3601
T12 3601
T13 3602
T14 3602
T15 3602
T16 3603
T17 3603
T18 3604
T19 3604
T20 3604
T21 3605
T22 3605
T23 3606
T24 3606
T25 3607
T26 3608
T27 3609

Power-Aware Rules (V Rules) 3610
V1 3611
V2 3612
V3 3612
V4 3612
V5 3613
V6 3613

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

V7 3613
V8 3614
V9 3614
V10 3615
V11 3616
V12 3617
V13 3618
V14 3619
V15 3619
V16 3620
V17 3620
V18 3620
V19 3621
V20 3621
V21 3622

Timing Rules (W Rules) 3623
W1 3624
W2 3624
W3 3625
W4 3625
W5 3625
W6 3626
W7 3626
W8 3627
W9 3627
W10 3627
W11 3628
W12 3628
W13 3629
W14 3629
W15 3629
W18 3630
W19 3630
W20 3631
W21 3631
W22 3632
W23 3632
W24 3632
W25 3633
W26 3633
W27 3634
W28 3634
W29 3635
W30 3635
W31 3636

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

W32 3636
W33 3637
W34 3638
W35 3638
W36 3639
W37 3639
W38 3640
W39 3641
W41 3642
W42 3642
W43 3643

X-bounding Rules (XB Rules) 3645
XB1 3645
XB2 3646

Other DRC Messages 3647
Oscillation Limitation 3647
RAM Summary Results and Test Capability 3647

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

RAM Rules (A Rules)

The tool checks RAM gates to identify proper test methods. You can select the handling of any RAM rule to be error, warning, note, or ignore.

Tip

In Tessent Visualizer, you can press Ctrl + R to execute a report_gates command on selected gates.

A1 3191
A2 3192
A3 3193
A4 3193
A5 3194
A6 3195
A7 3195
A9 3196
A10 3197
A11 3197
A12 3198
A13 3199
A14 3200
A15 3200
A16 3201
A17 3201
A18 3202

A1

Category: RAM

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

When all write control lines are at their off-state, all write, set, and reset inputs of RAMs must be at their inactive state. The tool performs this check using the simulated values that result when all defined write control lines are at their off-state, the constrained pins are set to their constrained values, and the initialized non-scan cells are set to their stable states. The rule violation occurs if any write, set, or reset input of any RAM gate is not off.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The default handling for this rule violation is a warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed on the error message. This identifies the input that is not held off, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not defining all write control lines (including those that are RAM set and reset lines) or defining the wrong off-state.

The occurrence message is:

```
Write controls off failed to force off RAM T line of N (G).  (A1-1)
```

T is the type of input (write, set, or reset), N is the instance name of the RAM gate, G is the gate ID number, and A1-1 is the rule and violation ID number.

The summary message is:

```
N RAM write/set/reset lines not forced off when write controls are off.
(A1)
```

N is the number of occurrences of rules violation A1.

A2

Category: RAM

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

A defined scan clock must not propagate to a RAM gate, except for its read lines. The tool performs this check by determining the forward cone of influence for all clock pins (clock cone). The bounds for the cone of influence are scan cells and circuitry set to a fixed value when constrained pins are set to their constrained values and the initialized non-scan cells are set to their stable states. The rule violation occurs when a RAM gate is in a clock cone.

The default setting for this DRC is “Ignore.” You can enable this check with the set_drc_handling command. When the DRC is enabled and an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, W indicates write control line cone, B indicates both, and “-” indicates no cone.

The occurrence message is:

```
Scan clock is connected to RAM N (G). (A2-1)
```

N is the instance name of the RAM gate and G is the gate ID number.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The summary message is:

```
N RAMs are connected to a scan clock. (A2)
```

N is the number of occurrences of rules violation A2.

A3

Category: RAM

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

A write or read control line must not propagate to an address line of a RAM gate. The tool performs this check by determining the forward cone of influence for all write and read control lines (write/read cone). The bounds for the cone of influence are scan cells, RAM gates, and circuitry set to a fixed value when constrained pins are set to their constrained values and the initialized non-scan cells are set to their stable states. The rule violation occurs when an address line of a RAM gate is in the write/read cone.

The default handling for this rule violation is warning. When an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, W indicates write/read control line cone, B indicates both, and "-" indicates no cone.

The occurrence message is:

```
RAM control line connected to address input of RAM N (G).  (A3-1)
```

N is the instance name of the RAM gate and G is the gate ID number.

The summary message is:

```
N RAM address inputs are connected to a RAM control line.  (A3)
```

N is the number of occurrences of rules violation A3.

A4

Category: RAM

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

A write or read control line must not propagate to a data line of a RAM gate. The tool performs this check by determining the forward cone of influence for all write and read control lines (write/read cone). The bounds for the cone of influence are scan cells, RAM gates, and circuitry set to a fixed value when constrained pins are set to their constrained value, and the initialized non-scan cells are set to their stable stated. The rule violation occurs when a data-in line of a RAM gate is in the write/read cone.

The default handling for this rule violation is warning. When an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, W indicates write/read control line cone, B indicates both, and "-" indicates no cone.

The occurrence message is:

```
RAM control line connected to data input of RAM N (G). (A4-1)
```

N is the instance name of the RAM gate and G is the gate ID number.

The summary message is:

```
N RAM data inputs are connected to a RAM control line. (A4)
```

N is the number of occurrences of rules violation A4.

A5

Category: RAM

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

A RAM gate must not propagate to another RAM gate. The tool performs this check by determining the forward cone of influence for all RAM gates (RAM cone). The bounds for the cone of influence are scan cells, RAM gates, and circuitry set to a fixed value when constrained pins are set to their constrained values and the initialized non-scan cells are set to their stable states. The rule violation occurs when an input of a RAM gate is in the RAM cone.

The default handling for this rule violation is warning. The occurrence message is:

```
RAM N1 (G1) connected to RAM N2 (G2). (A5-1)
```

N1 is the instance name of one RAM gate, G1 is its gate ID number, N2 is the instance name of the other RAM gate, and G2 is its gate ID number.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The summary message is:

```
N RAMs are connected to RAMs. (A5)
```

N is the number of occurrences of rules violation A5.

A6

Category: RAM

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

All the write inputs of all RAMs and all the read inputs of all data_hold RAMs must be at their off-state during all test procedures, except test_setup. The tool performs this check using the simulated values that result when it simulates the test procedures. The rule violation occurs if any write, set, or reset input of any RAM, or a read input of a data_hold RAM is not off.

The default handling for this rules violation is warning. Failure to satisfy this rule for write inputs results in the RAM being unavailable to hold its contents during scan operation, which may cause a loss in test coverage. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the input that is not held off, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not defining all write or read control lines, or defining the wrong off-state.

The occurrence message is:

```
L line of RAM N (G) not off during time T of P procedure.  (A6-1)
```

L is the type of input (write, read, set, or reset), N is the instance name of the RAM gate, G is the gate ID number, T is the time, and P is the procedure name.

The summary message is:

```
There were N occurrences of uncontrolled RAMs during test  procedures.
(A6)
```

N is the number of occurrences of rules violation A6.

A7

Category: RAM

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When all read control lines are at their off-state, all read inputs of RAMs with the read_off attribute set to hold must be at their inactive state. The tool performs this check using the simulated values that result when all defined read control lines are at their off-state, the constrained pins are set to their constrained values, and the initialized non-scan cells are set to their stable states. The rule violation occurs if any read input of a data_hold RAM gate is not off.

The default handling for this rules violation is warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the input that is not held off, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not defining all read control lines or defining the wrong off-state.

The occurrence message is:

```
Read controls off failed to force off RAM read line of N (G).  (A7-1)
```

N is the instance name of the RAM gate and G is the gate ID number.

The summary message is:

```
N RAM read lines not forced off when read controls are off.  (A7)
```

N is the number of occurrences of rules violation A7.

A9

Category: RAM

Contexts Supported: dft -scan, dft -test_points, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

A read-only RAM model should define an initialization file. If an initialization file is not defined, the tool treats instances of the RAM as TIEX, reducing test coverage.

The default handling for this rule violation is warning. To correct the problem, be sure the Tessent Cell library model of the RAM includes an init_file attribute that correctly specifies an initialization file.

The occurrence message is:

```
The initialization file is not defined for read-only RAM N (G).(A9-1)
```

N is the instance name of the RAM gate and G is the gate ID number.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The summary message is:

```
N read-only RAMs do not have initialization file. (A9)
```

N is the number of occurrences of rules violation A9.

A10

Category: RAM

Contexts Supported: dft -scan, dft -test_points, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

The content of a read-only RAM that does not have a write port must not be disturbed by any procedures except test_setup.

The default handling for this rule violation is warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the set or reset port that is not held off during simulation of a procedure, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not forcing the primary inputs, or not initializing the state elements, that control the RAM's set or reset port to an appropriate value.

When the A10 or A11 rule is violated, the tool treats the RAM as TIEX, which may reduce fault coverage.

The occurrence message is:

```
The content of read-only RAM N (G) can be disturbed through L line during
time T of P procedure. (A10-1)
```

N is the instance name of the RAM gate, G is the gate ID number, L is the type of input (set or reset), T is the time, and P is the procedure name.

The summary message is:

```
There were N occurrences of disturbed read-only RAMs during test
procedures.  (A10)
```

N is the number of occurrences of rules violation A10.

A11

Category: RAM

Contexts Supported: dft -scan, dft -test_points, patterns -scan_diagnosis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Default Handling: Warning

report_drc_rules: Supported

The content of a read-only RAM that does not have a write port must not be disturbed through the set or reset lines during capture.

The default handling for this rule violation is warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the set or reset port that is not held off during capture, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not constraining the primary inputs or the state elements that control the RAM's set or reset port.

When the A10 or A11 rule is violated, the tool treats the RAM as TIEX, which may reduce fault coverage.

The occurrence message is:

```
The content of read-only RAM N (G) can be disturbed through L line in
capture window.(A11-1)
```

N is the instance name of the RAM gate, G is the gate ID number, and L is the type of input (set or reset).

The summary message is:

```
There were N occurrences of disturbed read-only RAMs in capture window.
(A11)
```

N is the number of occurrences of rules violation A11.

A12

Category: RAM

Contexts Supported: dft -scan, dft -test_points, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

The content of a read-only RAM that has a write port must not be disturbed by any procedure except test_setup (in order for the initialization file to be usable for test generation).

The default handling for this rule violation is warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the set or reset port that is not held off during a procedure, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not defining the write, set or reset operation properly.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When rule A12 or A13 is violated, the tool ignores the RAM's contents initialized through the initialization file.

The occurrence message is:

```
The initialization file of RAM N (G) cannot be used during test pattern
generation since RAM content can be disturbed through L line during time T
of P procedure.(A12-1)
```

N is the instance name of the RAM gate, G is the gate ID number, L is the type of input (write, set or reset), T is the time, and P is the procedure name.

The summary message is:

```
There were N occurrences of disturbed RAM initialization file during test
procedures.  (A12)
```

N is the number of occurrences of rules violation A12.

A13

Category: RAM

Contexts Supported: dft -scan, dft -test_points, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

The content of a read-only RAM that has a write port must not be disturbed through the write, set or reset lines during capture (in order for the initialization file to be usable for test generation).

The default handling for this rule violation is warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the set or reset port that is not held off during capture, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not defining the write, set or reset operation properly.

When rule A12 and/or A13 is violated, the tool ignores the RAM's contents initialized through the initialization file.

The occurrence message is:

```
The initialization file of RAM N (G) cannot be used during test pattern
generation since RAM content can be disturbed through L line in capture
window. (A13-1)
```

N is the instance name of the RAM gate, G is the gate ID number, and L is the type of input (write, set or reset).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The summary message is:

```
There were N occurrences of disturbed RAM initialization file in capture
window. (A13)
```

N is the number of occurrences of rules violation A13.

A14

Category: RAM

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

RAM should be observable. The tool performs this check by forcing all pin constraints and tied cells and check if there is any sensitizable path from the outputs of a RAM to any observation point. The rule violation occurs if there is none.

The default handling for this rule violation is a warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and use the report_gates command for the gate ID number displayed on the error message.

```
// Warning: RAM N (G) is unobservable. (A14-1)
```

N is the instance name of the RAM gate and G is its gate ID number.

This identifies the RAM gate that cannot be observed. The usual cause of this error condition is constraints blocking the output of the RAM.

A15

Category: RAM

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

RAM should be able to read. The tool performs this check by forcing all pin constraints and cell constraints and check the simulation values at the read enable and read clock lines of each read control port of a RAM. The rule violation occurs if the simulation values indicate that all read control ports are off.

The default handling for this rule violation is a warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and use the report_gates command for the gate ID number displayed on the error message.

```
// Warning: RAM N (G) cannot read because its read control ports are
// constrained off. (A15-1).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

N is the instance name of the RAM gate and G is its gate ID number.

This identifies the RAM gate that cannot be observed. The usual cause of this error condition is constraints that prevent the read enable or read clock lines of a RAM to be turned ON.

A16

Category: RAM

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

RAM should be able to write. The tool performs this check by forcing all pin constraints and cell constraints and check the simulation values at the write enable and write clock lines of each write control port of a RAM. The rule violation occurs if the simulation values indicate that all write control ports are off. This rule checking is not performed for read-only ROMs.

The default handling for this rule violation is a warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and use the report_gates command for the gate ID number displayed on the error message.

```
// Warning: RAM N (G) cannot write because its write control ports are
// constrained off. (A16-1).
```

N is the instance name of the RAM gate and G is its gate ID number.

A17

Category: RAM

Contexts Supported: dft -edt, dft -logic_bist, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

This rule checks if a RAM cannot capture fault effects because its read port cannot be set to an active state.

The information provided by A17 helps you debug low fault coverage caused by read port issues. To check for A17, the tool forces and propagates constant values such as pin constraints, tied non-scan elements, and constant cell constraints. It also calculates forbidden values. The tool then processes each clock port. If the constrained value of the RAM's read_enable or read_clock is X, and if either of two signals cannot be justified to 1, the tool reports an A17 violation.

To minimize the number of violations reported, the tool traces from rule violation read ports to a common source point, which can be a primary input or internal pin. The violating sources are

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reported in order by number of affected elements with the first violation having the most, and so on. This is an example of the reported message when the read enable or read clock is X:

report_drc_rule A17

```
//  Warning: Internal gate N1 (G1) has a value of X and cannot be justified
//  to 1.
//  This prevents read by C memory elements, one of which is N2 (G2).
//  (A17-1)
```

N1 is the name of the gate with X value, G1 is the gate ID of that gate. C is the number of memory elements, N2 is the name of the gate whose read is prevented, G2 is the gate ID of that gate.

When a single primary input (PI) drives both the read_enable and the read_clock which are inverted from each other, this is the warning message:

```
//  Warning: Primary input P (S) cannot be justified to both 0 and 1
//  simultaneously to control read enable and clock ports that are
//  inverted from each other.
//  This prevents read by C memory elements, one of which is N (G).
//  (A17-1)
```

P is the primary input name, S is the net ID number, C is the count of memory elements, N is the gate name, and G is the gate ID number.

When the read clock and read enable pins cannot be justified to the on state simultaneously, this is the warning message:

```
//  Warning: Internal gate N1 (G1) and N2 (G2) that control read enable
//  and clock respectively cannot be justified to 1 and 1 simultaneously.
//  This prevents read by C memory elements, one of which is N3 (G3).
//  (A17-1)
```

N1 is the name of the read_enable gate, G1 is the gate ID of that gate. N2 is the name of the read_clock gate, G2 is the gate ID number of that gate, C is the number of memory elements, N3 is the name of the gate whose read is prevented, G3 is the gate ID of that gate.

The default handling of A17 is Warning. It is performed during system mode transition from Setup to Analysis. The summary message is:

```
A17:  #fails=1 handling=warning (memory elements cannot read because clock
ports cannot be fully controlled).
```

To debug A17 violations, use the "set_gate_report -constrain_value on" command to look at the constrained values on the read ports of affected RAMs.

A18

Category: RAM

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Contexts Supported: dft -edt, dft -logic_bist, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

This rule checks if a RAM cannot capture fault effects because its write port cannot be set to an active state.

The information provided by A18 helps you debug low fault coverage caused by write port issues. To check for A18, the tool forces and propagates constant values such as pin constraints, tied non-scan elements, and constant cell constraints. It also calculates forbidden values. The tool then processes each clock port. If the constrained value of the RAM's write_enable or the write_clock is X, and if either of the two signals cannot be justified to 1, the tool reports an A18 violation.

To minimize the number of violations reported, the tool traces from rule violation write ports to a common source point, which can be a primary input or internal pin. The violating sources are reported in order by number of affected elements with the first violation having the most, and so on. This is an example of the reported message:

report_drc_rule A18

```
//  Warning: Internal gate N1 (G1) has a value of X and cannot be justified
//  to 1.
//  This prevents write by C memory elements, one of which is N2 (G2).
//  (A18-1)
```

N1 is the name of the gate with X value, G1 is the gate ID of that gate. C is the number of memory elements, N2 is the name of the gate whose write is prevented, G2 is the gate ID of that gate.

When a single primary input (PI) drives both the write_enable and the write_clock which are inverted from each other, this is the warning message:

```
//  Warning: Primary input P (S) cannot be justified to both 0 and 1
//  simultaneously to control write enable and clock ports that are
//  inverted from each other.
//  This prevents write by C memory elements, one of which is N (G).
//  (A18-1)
```

P is the primary input name, S is the net ID number, C is the count of memory elements, N is the name of the gate whose write is prevented, and G is the gate ID number of that gate.

When the write clock and write enable pins cannot be justified to the on state simultaneously, this is the warning message:

```
//  Warning: Internal gate N1 (G1) and N2 (G2) that control write enable
//  and clock respectively cannot be justified to 1 and 1 simultaneously.
//  This prevents write by C memory elements, one of which is N3 (G3).
//  (A18-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

N1 is the name of the write_enable gate, G1 is the gate ID of that gate. N2 is the name of the write_clock gate, G2 is the gate ID number of that gate, C is the number of memory elements, N3 is the name of the gate whose write is prevented, G3 is the gate ID of that gate.

The default handling of A18 is Warning. It is performed during system mode transition from Setup to Analysis. The summary message is:

```
//  Warning: There were 264 A18 violations (memory elements cannot write
//  because clock ports cannot be fully controlled).
```

To debug A18 violations, use the "set_gate_report -constrain_value on" command to look at the constrained values on the write ports of affected RAMs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Assertion Rules (ASSERT Rules)

DRC checks that the assertion values on the pins of models from read_cell_library occur during DRC simulation over the specified period.

Assertion Values and the Enforcement of Assertions 3205
ASSERT1 3205
ASSERT2 3207
ASSERT3 3209

Assertion Values and the Enforcement of Assertions

This section describes assertion values and which phases of an ATPG simulation the tool checks the enforcement of assertions.

Assertion Values

Assertion values can be 0, 1, Z, or stable. A stable assertion is satisfied if the value is fixed for all the phases of the assertion: always 0, always 1, or always Z.

Enforcement of Assertions

DRC checks that instances of read_cell_library models with an assert_shift, assert_capture, or assert_pattern have the value specified at the time the tool enforces the assertion.

The five phases of a typical ATPG simulation are as follows:

1. test_setup before the first load_unload
2. load_unload before shift
3. shift procedures (and the changes between them)
4. Going from shift to capture at the end of load_unload when pin constraints are forced
5. Capture

The tool enforces assert_shift in phase 3 only (no assertion checks in phases 1, 2, 4, or 5).

The tool enforces assert_capture in phase 5 only (no assertion checks in phases 1, 2, 3, or 4).

The tool enforces assert_pattern in phases 2, 3, 4, and 5 (no assertion checks in phase 1).

ASSERT1

Category: DRC Simulation Value Checks

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Default Handling: Error

report_drc_rules: Supported

Values defined on Tessent cell library model pins via assert_shift should be as defined.

Values which should occur on all instances of a model throughout shift should be specified by an assert_shift on the input of the model. These values can be specified in Tessent cell library syntax or as Verilog assertions using Verilog syntax. Libcomp can be used to translate the Verilog assertions into Tessent syntax. For example:

```
module mux_scan (Q, CK, D, SI, SE);
   output Q;
   reg Q;
   input CK;
   input D;
   input SI;
   (* assert_shift = 1*) input SE;
   assign Din = SE ? SI : D;
   always @ (posedge CK) begin
   Q <= Din;
endmodule
```

Using Libcomp, both the Verilog hardware and assertions are translated to Tessent syntax.

```
model mux_scan
   (Q, CK, D, SI, SE)
(
   model_source = verilog_module;
   input (CK) ( )
   input (D) ( )
   input (SI) ( )
   input (SE) (  assert_shift = 1)
   output (Q) ()
(
   primitive = _mux  mlc_mux_1 ( D, SI, SE, Din );
   primitive = _dff  mlc_dff_1 ( , , CK, Din, Q,  );
)
)
```

Netlist instances with assert_shift are checked against stable shift simulation values for compliance. Violations default to “error”, which halts the tool and returns to the prompt. You can use set_drc_handling to change the default to Warning, Note, or Ignore.

Example:

```
//  command: set_drc_handling assert1 warn verbose
```

An example violation after the above setting follows:

```
//  Warning: assert_shift=Stable but sim val = "X" (not known and
//  constant) from driver '/clk_01' (11)
//           Assertion on instance/pin 'ff10/CLK'
//           Assertion came from pin 'CLK' of model 'sff'
//           line 500 file 'data/adk.atpg' (ASSERT1-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The report_drc_rules command supports assert_shift (ASSERT1) as follows:

report drc rules assert1

```
//  Warning: assert_shift=Stable but sim val = "X" (not known and
//  constant) from driver '/clk_01' (11)
//           Assertion on instance/pin 'ff10/CLK'
//           Assertion came from pin 'CLK' of model 'sff'
//           line 500 file 'data/adk.atpg' (ASSERT1-1)
```

The report_gates command also supports ASSERT1 (assert_shift) after issuing:

set_gate_report drc_pattern stable_shift

You can have an assert_shift and an assert_capture both on the same input, as shown below:

```
(* assert_shift = 1, assert_capture = 0 *) input SE;  // Verilog syntax
input (SE) (assert_shift = 1; assert_capture = 0)  // Tessent syntax
```

ASSERT2

Category: DRC Simulation Value Checks

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Values defined on Tessent cell library model pins via assert_capture should be as defined.

Values which should occur on all instances of a model throughout capture should be specified by an assert_capture on the input of the model. These values can be specified in Tessent cell library syntax or as Verilog assertions using Verilog syntax. Libcomp can be used to translate the Verilog assertions into Tessent syntax. For example:

```
module mux_scan (Q, CK, D, SI, SE);
   output Q;
   reg Q;
   input CK;
   input D;
   input SI;
   (*assert_capture = 0*) input SE;
   assign Din = SE ? SI : D;
   always @ (posedge CK) begin
   Q <= Din;
endmodule
```

Using Libcomp, both the Verilog hardware and assertions are translated to Tessent syntax.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
model mux_scan
   (Q, CK, D, SI, SE)
(
   model_source = verilog_module;
   input (CK) ( )
   input (D) ( )
   input (SI) ( )
   input (SE) (  assert_capture = 0)
   output (Q) ()
(
   primitive = _mux  mlc_mux_1 ( D, SI, SE, Din );
   primitive = _dff  mlc_dff_1 ( , , CK, Din, Q,  );
)
)
```

Netlist instances with assert_capture are checked against stable capture simulation values for compliance. Violations default to “error”, which halts the tool and returns to the prompt. You can use set_drc_handling to change the default to Warning, Note, or Ignore.

Example:

set_drc_handling assert2 note verbose

An example violation after the above setting follows:

```
//  Note: assert_capture=0 but sim val = "X" (not known and constant) from
//  driver '/d1' (1)
//        Assertion on instance/pin 'ff10/D'
//        Assertion came from pin 'D' of model 'sff'
//        line 500 file 'data/adk.atpg' (ASSERT2-1)
```

The report_drc_rules command supports assert_capture (ASSERT2) as follows:

report drc rules assert2

```
//  Note: assert_capture=0 but sim val = "X" (not known and constant) from
//  driver '/d1' (1)
//        Assertion on instance/pin 'ff10/D'
//        Assertion came from pin 'D' of model 'sff'
//        line 500 file 'data/adk.atpg' (ASSERT2-1)
```

The report_gates command also supports ASSERT2 (assert_capture) after issuing:

set_gate_report drc_pattern stable_capture

You can have an assert_shift and an assert_capture both on the same input, as shown in the following:

```
(* assert_shift = 1, assert_capture = 0 *) input SE;  // Verilog syntax
input (SE) (assert_shift = 1; assert_capture = 0)  // Tessent syntax
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ASSERT3

Category: DRC Simulation Value Checks

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Values defined on Tessent cell library model pins through assert_pattern should be as defined.

Values that should occur on all instances of a model throughout the entire pattern should be specified by an assert_pattern on the input of the model. These values can be specified in Tessent cell library syntax or as Verilog assertions using Verilog syntax. You can use Libcomp to translate the Verilog assertions into Tessent syntax. For example:

```
primitive prim_not(out, in);
  (* Assert_capture = "1'b0", ASSERT_pattern = "stable" *)
  input in;
  (* assert_capture = "Stable", assert_pattern = "Z", assert_shift = 1'bz *)
  output out;
      table
      // in : out
          0 : 1 ;
          1 : 0 ;
          X : X ;
      endtable
endprimitive
```

Using Libcomp, both the Verilog hardware and assertions are translated to Tessent syntax.

```
model prim_not
  (out, in)
(
  model_source = verilog_udp;

  input (in) (  Assert_capture = 1'b0;   ASSERT_pattern = stable;  )
  output (out) (   assert_capture = Stable;   assert_pattern = Z;   assert_shift = 1'bz;
)  (
  primitive = _inv mlc_gate0 (in, out);
  )
)
```

Netlist instances with assert_pattern are checked against stable pattern simulation values for compliance. Violations default to error which halts the tool and returns to the prompt. The set_drc_handling command is used to change the default to Warning, Note or Ignore.

Example:

set_drc_handling assert3 warn verbose

Two example violations after the above setting follow. Note that the second example says to see stable_shift, because the pattern violation occurred within shift. This is more helpful for finding the root cause of the violation to fix it. If the violation is only in capture, it indicates that. The

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

first violation shown below does not occur within shift or within capture, so it says to see stable_after_setup which shows values across the entire pattern.

```
//  Warning: assert_pattern=1 but sim val = "X" (not known and constant)
//  (see stable_after_setup) from driver '/don_in' (10)
//          Assertion on instance/pin 'not1/A'
//          Assertion came from pin 'A' of model 'inv01'
//          line 266 file 'data/adk.atpg' (ASSERT3-1)
//  Warning: assert_pattern=1 but sim val = "0" (see stable_shift) from
//  driver '/not1/Y' (64)
//          Assertion on instance/pin 'not2/A'
//          Assertion came from pin 'A' of model 'inv01'
//          line 266 file 'data/adk.atpg' (ASSERT3-2)
```

The report_drc_rules command supports assert_pattern (ASSERT3) as follows:

```
//  command: report drc rules assert3
//  Warning: assert_pattern=1 but sim val = "X" (not known and constant)
// (see stable_after_setup) from driver '/don_in' (10)
//          Assertion on instance/pin 'not1/A'
//          Assertion came from pin 'A' of model 'inv01'
//          line 266 file 'data/adk.atpg' (ASSERT3-1)
```

The report_gates command also supports ASSERT3 (assert_pattern) after issuing:

set_gate_report drc_pattern stable_after_setup

However, as stated above, it may be more helpful to use stable_shift or stable_capture, and the violation message indicates when this is true.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BIST Rules (B Rules)

Whenever LFSRs are defined, the tool performs BIST rules checking to ensure proper application of BIST patterns to the circuit. Except for rule B2 or B7, you cannot change the handling of the BIST rule from the default condition of error.

B2 3211
B4 3211
B5 3212
B6 3212
B7 3214
B17 3216

B2

Category: BIST

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Every scan chain input pin must connect to an LFSR. Correct this error condition by connecting the scan chain input pin of the indicated chain to an LFSR.

You can change how the rules checker handles this check with the set_drc_handling command.

The error message is:

```
Input of chain C has no LFSR connection. (B2-1)
```

C is the name of the scan chain.

B4

Category: BIST

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Every scan chain output pin must connect to an MISR. Correct this error condition by connecting the scan chain output pin of the indicated chain to an MISR.

The error message is:

```
Output of chain C has no MISR connection. (B4)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

C is the name of the scan chain.

B5

Category: BIST

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

When observation scan is active, validates that the observation scan cells are connected in their own scan chain, distinct from other scan cells. The tool reports failing scan cells that are not identified as observation scan cells but placed on observation scan chains.

The only allowed handling of this DRC is Error. Any scan cells that the tool does not identify as observation scan cells should not be part of an observation scan chain.

The error message for the DRC violation is:

```
//  Error: Identified observation scan chain ‘chain_name' contains regular
//  scan cell 'scan_cell_name'. (B5-1)
```

The summary message is:

```
//  Error: There were N B5 violations (Observation scan cells are combined
//  with regular scan cells).
```

Where *N* is the number of B5 violations.

Use Tessent Visualizer to view B5 errors with the analyze_drc_violation command. The tool displays the corresponding scan cell that fails this check.

B6

Category: BIST

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

When observation scan is active, validates the connectivity of the observation scan OPs to the observation scan cells to ensure that the scan cells can capture data correctly for both shift and capture cycles. The tool reports failing scan cells along with the identified observe point locations.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Error Messages

The default handling of this DRC is Error. You can change handling to Warning, Note, or Ignore. When set to Ignore, the tool skips this DRC analysis.

The summary message is:

```
//  Error: There were N B6 violations (Observation scan cell connectivity
//  validation).
```

Where *N* is the number of B6 violations.

The B6 rule can identify the following types of errors:

- Type 1: The observation scan cell failed to capture the simulated values on its identified OP location during shift or capture.
- Type 2: DRC analysis cannot identify an OP location for the observation scan cell.

Each type of failure produces different violation messages, as shown in the following examples:

Example 1

This example shows DRC violation messages for shift and capture cycle simulation failures. Although the DRC analysis successfully identified the OP location for the obs_scan cell, the analysis found unexpected values simulated on the cell.

```
// Error: Observation scan cell '/ts_0_osp_496smodp1_i/
// ts_0_logic_1000fsffp1_i' failed to capture value simulated on its OP
// location 'ts_0_osp_496smodp1_i/ts_0_logic_1000and_usr2s0_usr0s0_i/A0'
// during capture. (B6-2)

// Error: Observation scan cell '/ts_0_osp_504smodp1_i/
// ts_0_logic_1000fsffp1_i' failed to capture value simulated on its OP
// location '/ts_0_osp_504smodp1_i/ts_0_logic_1000and_usr2s0_usr0s0_i/A0'
// during shift. (B6-3)
```

Example 2

This example shows DRC violation messages for failure to identify an OP location.

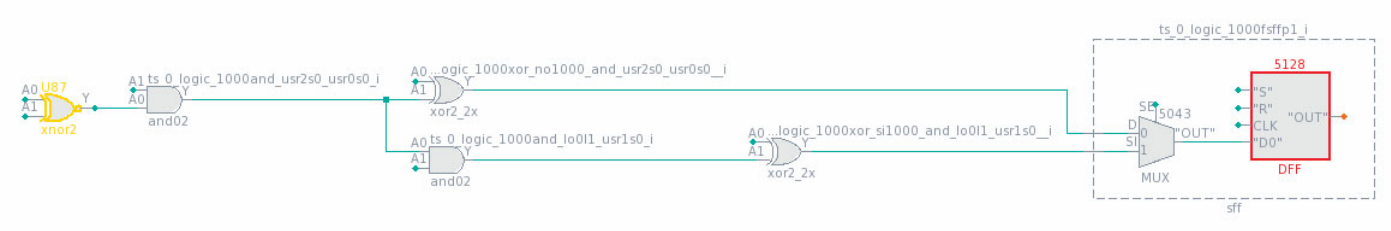
```
// Error: Failed to identify the OP location for observation scan cell
// '/ts_0_osp_497smodp1_i/ts_0_logic_1000fsffp1_i'. (B6-2)
```

Tessent Visualizer

Use Tessent Visualizer to view B6 violations with the analyze_drc_violation command. For a Type 1 violation, the paths between the failed observation scan cell and its identified OP

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

location display, as shown in the figure below. For other types of violations, only the failed observation scan cell displays.



B7

Category: BIST

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

An IJTAG TDR should drive LogicBIST-related signals to ensure the signals are constrained during fault simulation. This rule attempts to back-trace the signal in the flat model or ICL model network to determine if a TDR drives the signal.

Note

The rule flags a violation for any signal that traces back to a child physical block.

The rule traces the following signals:

- LogicBIST-related DFT signals (specified by the add_dft_signals command).
- Static enable signals used in the Hybrid TK/LBIST flow (specified by the set_dft_enable_options command).

Use the set_drc_handling command to change how the rule checker handles this check. You can change it to Warning, Note, or Ignore; however, if you do so, you must manually ensure that the signal is constrained.

The error message is:

```
// Error: LBIST related signal 'signal_name'
//        located at 'signal_location' is not driven by TDR.
//        Being driven by TDR is required in order to ensure the signal is
//        constrained. (B7-1)
```

The summary message is:

```
// Error: There were N B7 violations (LBIST related control signal is not
//  driven by TDR).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Where *N* is the number of B7 violations.

How to Debug with Tessent Visualizer

You can debug a specific occurrence of a B7 violation using Tessent Visualizer as follows:

1. Issue the command set_system_mode analysis, then open Tessent Visualizer with the open_visualizer command.
2. Choose the **DRC Violations** tab in the Design Browser window and locate the DRCs of interest in the table. Double-click the listed violation or right-click and choose **Visualize DRC** from the popup menu to analyze the violation.

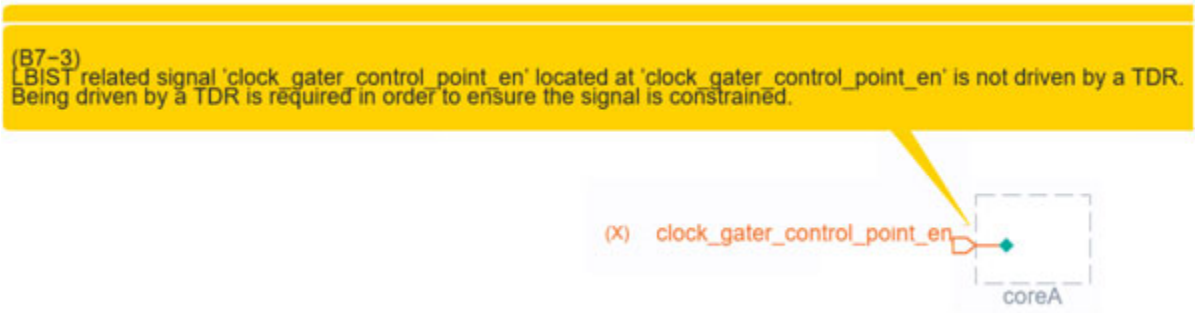
Alternatively, you can run the analyze_drc_violation command with a *<rule_id>-<occurrence#>* argument from the command line. For example: "analyze_drc_violation b7-1".

Tessent Visualizer displays the visualized B7 violation in the Hierarchical Schematic window as the highlighted location of the signal along with the error message. Additionally, if available, the tool displays a path visited by the tracer when it attempted to trace back from the signal location to a TDR. The values seen by the tracer at subsequent gate pins are also shown.

Examples

Example 1

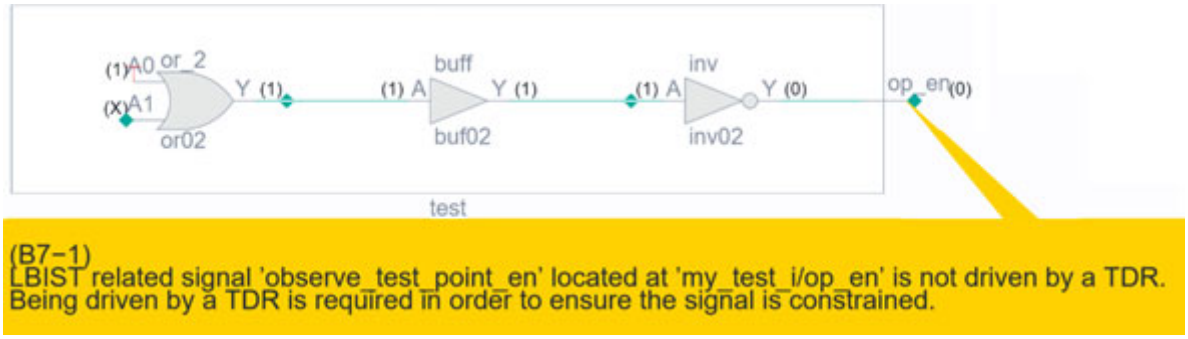
This is an example of a signal driven by a Primary Input.



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

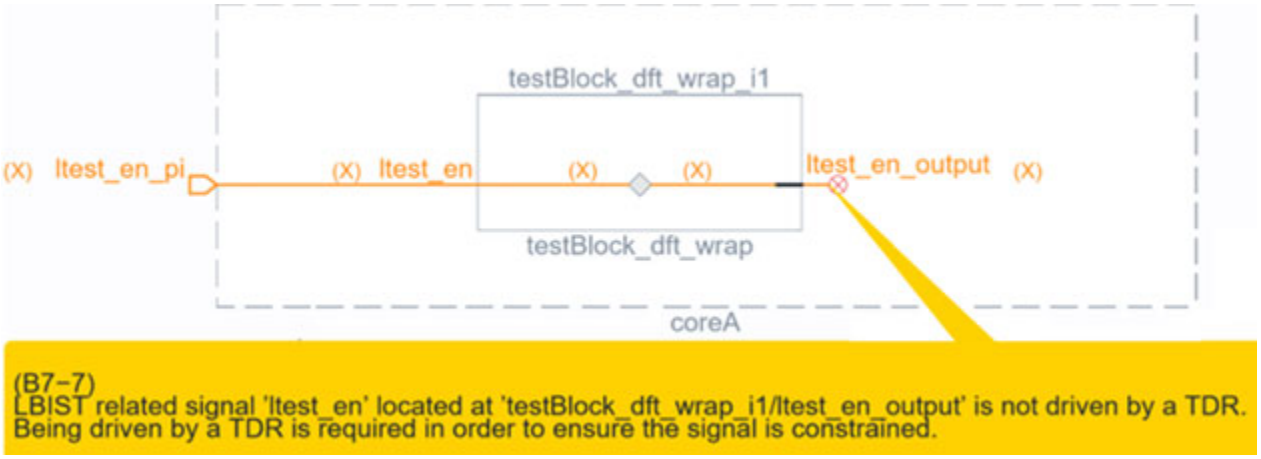
Example 2

This is an example of a signal driven by a constant through some logic.



Example 3

This is an example of a signal driven by a Primary Input (via a submodule).



B17

Category: BIST

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Verifies that the timing of the scan chain is consistent with the DftSpecification. The default handling of this rule violation is error. You can downgrade it to warning, note, or ignore. When set to ignore, the check is not performed.

The tool checks this rule during LBIST fault simulation to verify that the scan chain timing for the first and last scan cells is correct with respect to the leading and trailing edges of the EDT clock.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Error Messages

If the first scan cell capture time is not LE, this DRC reports the following error message:

```
// Error: Chain 'chain_name' input capture time 'time'(TE) is
inconsistent with the chain endpoints specified during LBIST/EDT
controller generation (LE): 'time'. (B17-1)
```

If the last scan cell change time is not TE, the following message is reported:

```
// Error: Chain 'chain_name' output change time 'time'(LE) is
inconsistent with the chain endpoints specified during LBIST/EDT
controller generation (TE): 'time'. (B17-2)
```

Because the tool builds the EDT logic based on 'le_te' chain endpoints, proceeding with failures or mismatches after any timing check failure on first or last scan cells may lead to simulation mismatches on LBIST patterns or LBIST single chain diagnosis patterns. Therefore, if the DRC handling is warning or note, the tool does not generate LBIST patterns or LBIST single chain diagnosis patterns if any of the first or last scan cells fail this DRC. If the DRC handling is ignore, the tool skips this DRC, and you can generate both LBIST patterns and LBIST single chain diagnosis patterns.

For this particular DRC, you must define the EDT clock and pulse it during the shift procedure. If you do not pulse the edt_clock during the shift procedure, this DRC reports the following error message:

```
// Error: Unable to identify edt_clock timing for block 'block_name'
during chain endpoints validation for B17 DRC.
// Please make sure to pulse edt_clock during shift or use
'set_drc_handling B17 ignore' to skip this DRC.
```

Note

This error cannot be downgraded to warning or note, but can be ignored with “set_drc_handling B17 ignore”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BSCAN Rules

The BSCAN DRCs create and use the dft_bscan simulation context. In this context, the tool applies the compliance enables as simulation forces.

BSCAN1 3218
BSCAN2 3219
BSCAN3 3219
BSCAN4 3219
BSCAN5 3220
BSCAN6 3220
BSCAN7 3220
BSCAN8 3220
BSCAN9 3221

BSCAN1

Category: BSCAN

Contexts Supported: dft_bscan

Default Handling: Error (not downgradable)

report_drc_rules: Supported

This DRC identifies the BoundaryScan TAP controller. It performs a controlling trace on the flat model from TDI to the TAP controller, and from the TAP controller to TDO.

The tool reports an error when the DRC cannot find a TAP controller while tracing from TDI, or when the TAP controller cannot be traced to TDO. The DRC adds simulation forces on the TDO enable of all discovered TAP controllers to enable the path to TDO.

Only one TAP controller may exist between TDI and TDO. Once the DRC identifies a TAP controller, it determines the ICL host scan interface for BoundaryScan. The ICL host scan interface must either be named “host_bscan,” or have its “tessent_is_bscan_host” attribute set to “true.”

The DRC forces the select signal of the host scan interface to on for subsequent DRCs.

Note

Daisy-chained TAP controllers and similar topologies are not supported.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BSCAN2

Category: BSCAN

Contexts Supported: dft_bscan

Default Handling: Error (not downgradable)

report_drc_rules: Supported

This DRC identifies the BoundaryScan register. It performs a controlling backward trace from the BoundaryScan scan interface of the TAP controller.

If the DRC finds the scan out signal of a tcd_bscan instance, the trace continues to the scan in signal. This trace should find either TDI or the TAP controller. The DRC reports an error if the BoundaryScan register is incorrect. When the DRC successfully traces a BoundaryScan register, it checks that the external ports for all discovered instances are connected to top level ports.

BSCAN3

Category: BSCAN

Contexts Supported: dft_bscan

Default Handling: Error (not downgradable)

report_drc_rules: Supported

This DRC reports all tcd_bscan instances that are not part of the BoundaryScan register. You can exclude certain instances by setting the “ignore_during_boundary_scan_extraction” attribute.

BSCAN4

Category: BSCAN

Contexts Supported: dft_bscan

Default Handling: Error (not downgradable)

report_drc_rules: Supported

This DRC checks any bonding configuration bypasses in the design. It reports bypass enable signals that are not found in the flat model, as well as those that have an undefined value in the dft_bscan simulation context.

If the DRC identifies a segment selection with unneeded enable signals, no error is reported for any missing or undefined enable signals. This DRC also reports an error if it cannot match a selection when all of the enable signals are set to known values.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BSCAN5

Category: BSCAN

Contexts Supported: dft_bscan

Default Handling: Error (not downgradable)

report_drc_rules: Supported

This DRC checks the control signal connections between the TAP controller and the tcd_bscan instances. It performs a controlling backward trace from the tcd_bscan instances, and reports an error if the connection cannot be traced or if the TAP controller pin does not match the function of the start pin. It also reports an error if the TAP controller pin belongs to the wrong host scan interface.

BSCAN6

Category: BSCAN

Contexts Supported: dft_bscan

Default Handling: Error (not downgradable)

report_drc_rules: Supported

The Tessent Multi-die 2.5D BSDL extraction flow uses this DRC to check that the TCK, TMS, and TRST die pins connect to package-level ports.

BSCAN7

Category: BSCAN

Contexts Supported: dft_bscan

Default Handling: Error (not downgradable)

report_drc_rules: Supported

The Tessent Multi-die 2.5D BSDL extraction flow uses this DRC to check that the compliance enables connect to package-level ports and that they can be set with values suitable for all dies.

BSCAN8

Category: BSCAN

Contexts Supported: dft_bscan

Default Handling: Error (not downgradable)

report_drc_rules: Not Fully Supported

This DRC checks and extracts the pad information for the BoundaryScan DFT insertion and BSDL extraction.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

You can use the report_drc_rules command to display a summary of DRC violations (fails) or to report a given error, warning, or note message again.

For BSCAN8, the tool returns messages followed by a generic "The BoundaryScan pad extraction has errors" in certain error states. In this case, only that message is counted and reported. As such, the message count might be lower than expected, and the special violation report only shows the generic message. However, the error count is guaranteed to be greater than 0 if an error occurred.

Refer to the "Pad Cell Library" section of the *Tessent BoundaryScan User's Manual* for more information.

BSCAN9

Category: BSCAN

Contexts Supported: dft_bscan

Default Handling: Error (not downgradable)

report_drc_rules: Supported

BSDL extraction uses this DRC to validate instruction uniqueness and length.

If you add instruction opcodes using the add_bsdl_extraction_user_instruction command, the tool checks that the instruction opcodes are valid and unique, and match the specified length of the instruction register. It also checks for the validity of the IDCODE instruction and device identification register.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Clock Rules (C Rules)

The application checks the scan clocks to ensure their proper definition and operation. You may select the handling of any clock rule to be error, warning, note, or ignore.

The following subsections describe the clock rules and the special handling you can set for them.

Tip

In Tessent Visualizer, you can press Ctrl + R to execute a report_gates command on selected gates.

Clock Terminology 3222
The ATPG Analysis Option 3227
C1. 3227
C2. 3232
C3. 3233
C4. 3239
C5. 3244
C6. 3246
C7. 3248
C8. 3249
C9. 3250
C10. 3252
C16. 3253
C17. 3254
C19. 3255
C21. 3257
C22. 3257
C23. 3260
C24. 3261
C25. 3262
C26. 3263
C27. 3265

Clock Terminology

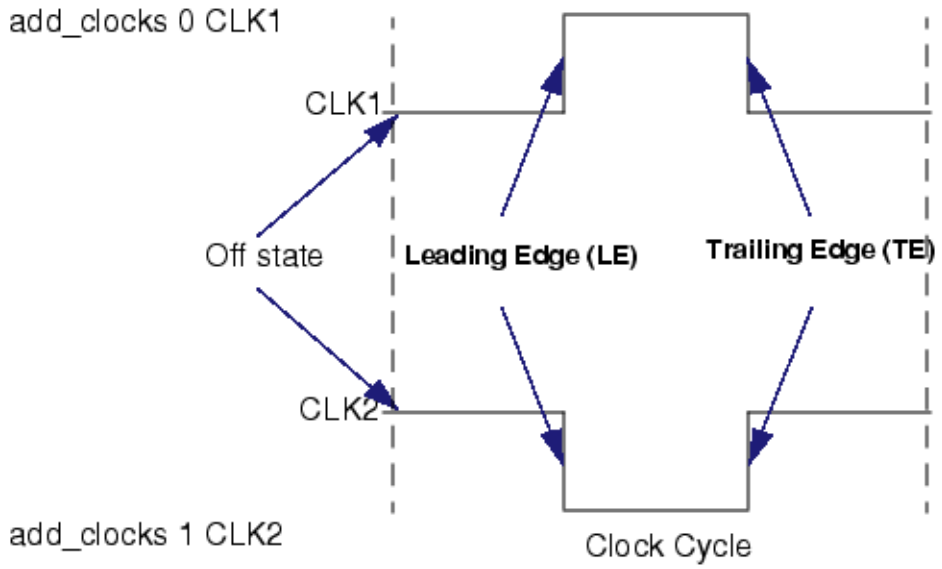
The clock rule information contains a couple of recurring concepts. To make optimal use of the information, you should understand these concepts.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Clock Signals

The tool considers any signal to be a clock if it can change the state of a sequential element, including system clocks, sets, and resets. When you define each clock primary input (a required setup step you perform using the add_clock command or "analyze_control_signals -Auto_fix"), a key piece of information specified with either of these commands is the pin value that represents the clock's off state. Two important terms arise out of this definition, as shown in the following figure.

Figure 10-4. Clock Cycle Terminology



The transition of the clock from the off state to the on state is considered the leading edge of the clock, while the transition from the on state to the off state is considered the trailing edge of the clock.

Clock Cones and Effect Cones

A gate pin or output pin is considered to be in the clock cone of a clock signal when they are connected through combinational logic gates and transparent latches (TLAs) to the clock primary input.

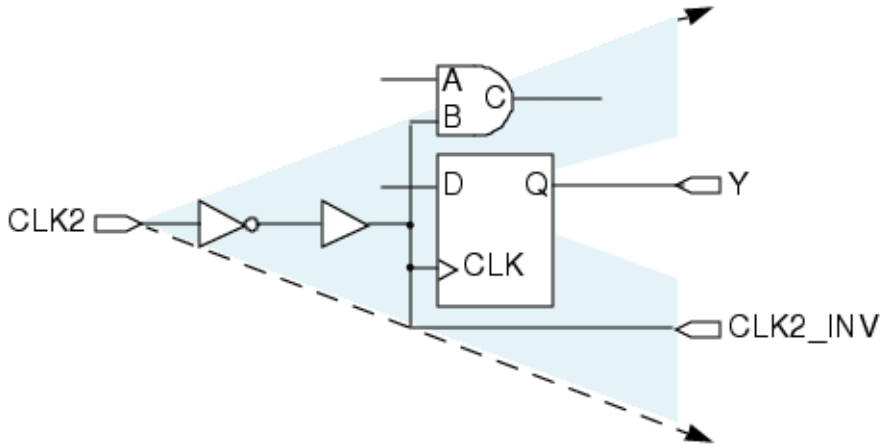
Note

The tools treat a latch as a transparent latch (TLA) rather than as a normal level-sensitive sequential element (LA) if it is continuously enabled at the start of the capture cycle when all clocks are at their defined off values. This treatment therefore depends on tool setups and how the latch is wired into the design. A TLA passes values without holding state and thus acts like a buffer. For more information, refer to "Simulation Primitives of the Flattened Model" in the *Tessent Scan and ATPG User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

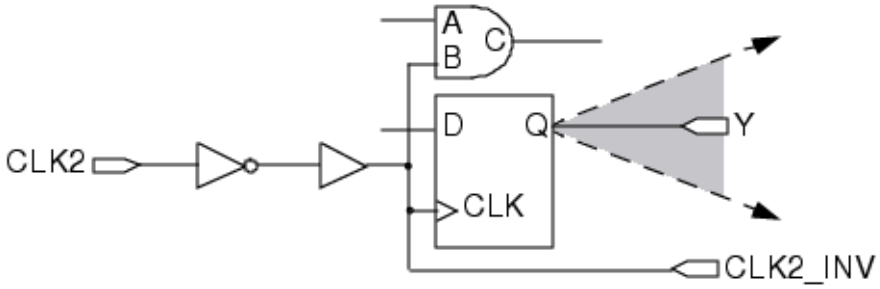
The clock cone is basically the fanout of the clock signal through strictly combinational logic and TLAs. Pins B, C, CLK and CLK2_INV in the circuit excerpt shown in the following figure are all in CLK2's clock cone.

Figure 10-5. Example Clock Cone



The tools consider a gate pin or output pin to be in a clock's effect cone if there is a sequential element between the clock net and the gate pin or output pin. In the following figure, pin Y is in the effect cone of CLK2.

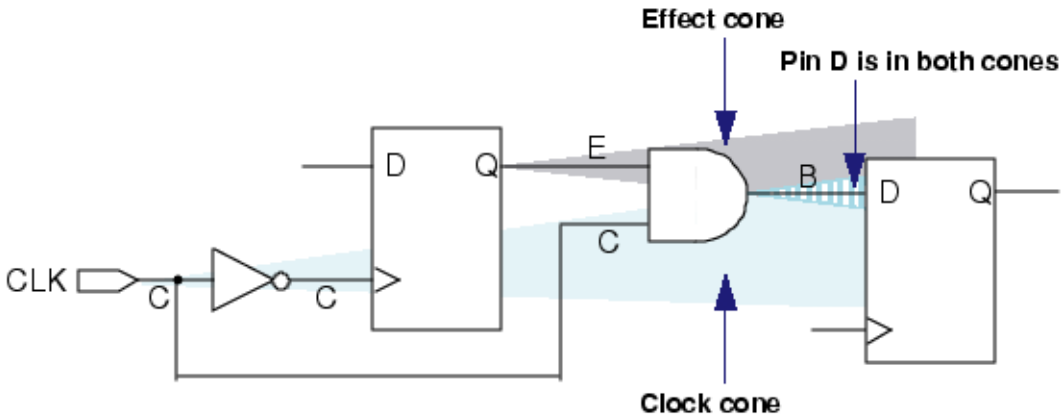
Figure 10-6. Example Effect Cone



A pin can be in both the clock cone and the effect cone as shown for pin D on one of the flip-flops in the following figure.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-7. Pin in Both Cones

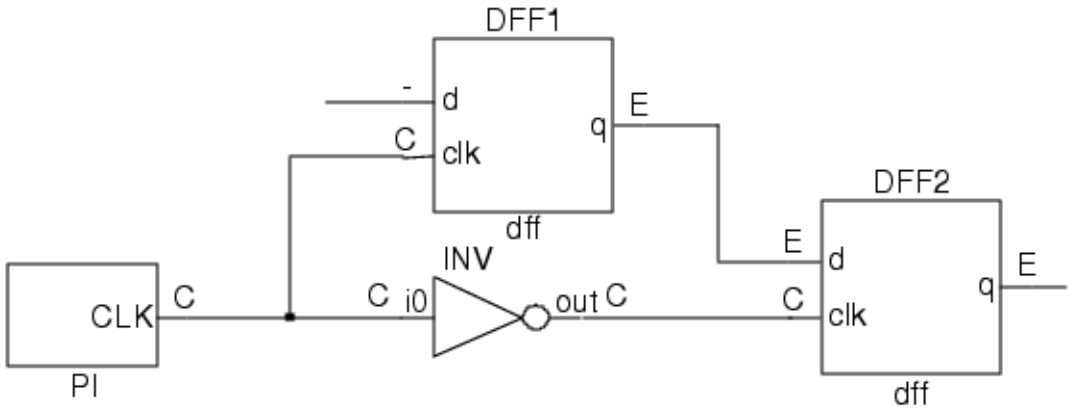


You can get the Flat Schematic window of Tessent Visualizer to show a C, E or B near device pins that are in the clock (C), effect (E), or both (B) cones of a particular clock by issuing the set_gate_report command with the Clock_cone option after the tool has flattened the design. For example:

```
create_flat_model
set_gate_report clock_cone CLK
```

Figure 10-8 shows an example of how this might look for a circuit fragment in the Flat Schematic window of Tessent Visualizer.

Figure 10-8. Tessent Visualizer Flat Schematic Window View Showing Clock Cones



When the Clock_cone option is in effect, the output of the report_gates command also displays clock cone data:

```
report_gates /DFF1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /DFF1  dff
//    clkI  (C)2-/CLK
//    d  I  (-)1-/C
//    q  O  (E)5-
```

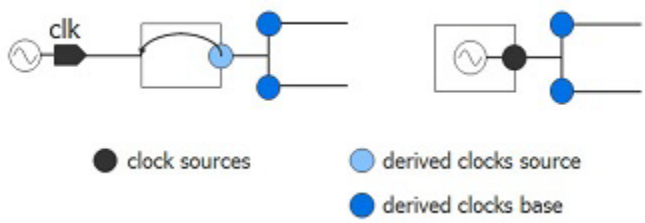
Note

The tool shows clock and effect cone data for just one clock at a time—the clock whose primary input pin_name you specified in the last “set_gate_report Clock_cone” command.

Clock Source and Derived Clocks

For clock rule checks, Scan Cell Data Rules (D rules) through DFT_C5 (Memory BIST clock tracing), three different types of clocks can be defined at primary ports or instance pins: synchronous and asynchronous clock sources, derived clock sources, and derived clock branches. The differences between these clocks types are shown in the following figure:

Figure 10-9. Clock Source And Derived Clocks



The different clock types are described as follows:

- **clock source (synchronous or asynchronous)** — is defined at a primary or instance pin where a clock pulse is available in the design. It can be a well-defined pin where an external clock generator is connected, or it can be a well-defined instance pin where an internal clock generator produces a clock signal.
- **derived clock source** — is a defined instance pin where a reference pin has to be specified because a tracing path to the corresponding “clock source” is blocked. One typical case is a PLL instance represented as a blackbox in the design. Using the add_clocks -reference option, you can link the derived clock source instance pin to a clock source.
- **derived clock branch** — a derived clock branch describes the case in which the clock tree of a design, sometimes different branches of the tree, can be unbalanced even if they are derived from the same clock source. That is, clock branches are not guaranteed to be synchronous between these clock branches. You define such a clock branch using the add_clocks -branch option. A derived clock branch must always be driven by a “clock source” or a “derived clock source”. Clock branches can also be used for later network manipulations.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The ATPG Analysis Option

Clock rules C1, C3, C4, and C5 can run full ATPG analysis during their checks.

For more information on ATPG analysis, refer to "How to Turn on ATPG Analysis."

The Memory BIST Analysis Option

Clock rule checks DFT_C1 through DFT_C5 run only in the dft context, and only if there are memory instances in the design and the "set_dft_specification_requirements -memory_test" option is set to "on".

These clock rule checks are skipped entirely for all other cases.

C1

Category: Clock

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

A scan or non-scan cell must not capture data when all specified clocks are set to their off states.

When all clocks are at their off state as defined with the add_clocks command, all clock inputs (including sets and resets) of scan and non-scan cells must be at their off state. For non-scan cell violations, the tool converts these to TIEX.

In the case of a DFF scan cell, the clock input must be a stable binary value (0 or 1).

Note

If the clock input value of a DFF scan cell simulates as X, the tool by default performs additional analysis to determine if the X represents a stable binary value. An X that represents a stable binary value satisfies the C1 rule's clock input requirement for a DFF. (To turn off the additional analysis, use the set_drc_handling command's Noatpg_analysis argument.)

The tool performs this check using the simulated values that result when all defined clocks are at their defined off states, constrained pins are set to their constrained values, initialized non-scan cells (initialized during test_setup or load_unload) are set to their stable states, and C0, C0DX, C1, and C1DX cell constraint values are applied. The rule violation occurs if any clock input, including set and reset lines, of any scan cell memory element is not at its off state.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When the C1 violation occurs on a set/reset port or a level-sensitive latch port, the tool does not issue an error message and does not stop the DRC process. Instead, the tool issues the following summary messages to indicate such situation exist:

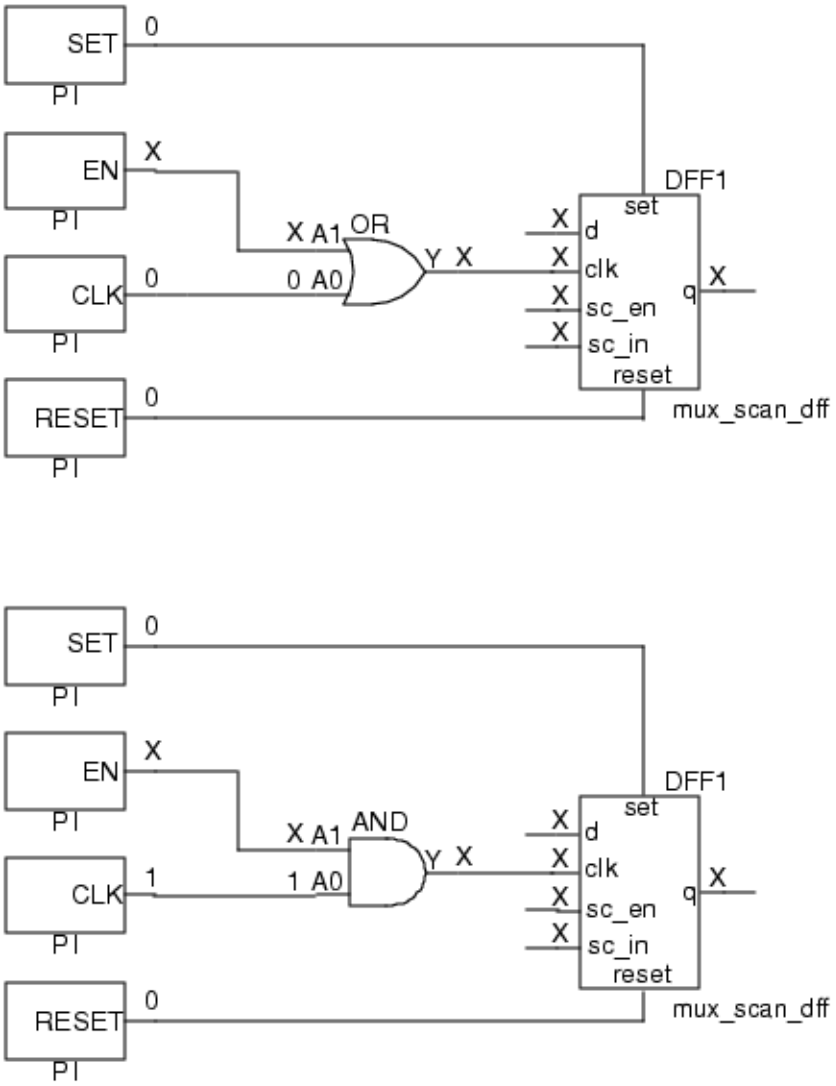
```
// Note: n scan cells whose clock ports are not off when all clocks are
// off will be handled through additional ATPG effort.

// The scan cells have been excluded from the list of C1 DRC violations.
// Use "report_drc_rules C1 -excluded" to report these C1 violations
```

If the number of such scan cells is large, ATPG runtime is increased. One common root cause of these C1 violations is a set/reset pin not defined as a clock. You can define a set/reset pin as a clock to fix the problem and avoid ATPG performance impact.

Figure 10-10 shows an example of a circuit segment that produces a C1 violation due to an indeterminate value (X) at the clk input of DFF1 when the off state of the clocks at the PIs is defined as a logic 0 and all defined clocks are off.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-10. C1 Violation Example



The clock input value to the scan cell is X because the EN signal value is X. If EN is left at X, the signal arriving at the clk input of the scan cell cannot be held off.

Effect on Testability

Failure to satisfy this rule can result in unstable scan and non-scan cell values going into or coming out of load_unload, or between cycles. This can result in lower test coverage.

How to Debug C1 Violations

The occurrence message is:

```
// Clock PIs off failed to force off a clock line of T N (G). (C1-1).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

T is the type of scan cell (MASTER, SLAVE, etc.), N is the instance name of the gate, and G is the gate ID number.

The summary message is:

```
// There were N clock rule C1 fails (unstable scan cells when clocks off).
```

N is the number of occurrences of rules violation C1.

For debugging, be sure C1 handling is set to error (the invocation default setting) before you enter a “set_system_mode analysis” command to leave setup mode. Use the following command to check the setting:

report_drc_rules -fails_summary

If the handling is other than error, you can change it back to error with:

set_drc_handling c1 error

When C1 handling is set to error, the first C1 violation that occurs stops the DRC process and returns the tool to Setup mode.

Tip

The C1 violation on which the tool stops is the C1 you must debug. Fixing the C1 violations the tool singles out in this manner often fixes other C1’s with no extra effort on your part, so it is very efficient.

The tool transcribes details about the violation and saves the simulation data related to the violation, so you can report it.

You can then debug the violation using Tessent Visualizer or by issuing commands from the command line. Examples of both methods follow.

How to Debug with Tessent Visualizer

To view the location of a C1 DRC error using Tessent Visualizer, use the following command:

open_visualizer

Choose the **DRC Browser** tab, then find the violation ID of the desired DRC violation in the table. Double clicking the listed violation, or right clicking and selecting **Visualize DRC** from the popup menu analyzes the violation.The scan cell affected by that occurrence is displayed in the Flat Schematic.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tip

Alternatively, you can issue the analyze_drc_violation command with the C1-1 argument at the tool command line. For example:

```
analyze_drc_violation c1-1
```

Once the offending scan cell is displayed, trace backward from the cell's clock input to the primary input that drives the clock. Ensure the clock's off state, which you defined using the add_clocks command, is correct. To change the defined off state, delete the old definition with the delete_clocks command, then reissue add_clocks. To see a clock's current defined off state, use the report_clocks command.

How to Debug from the Tool Command Line

To view the location of a C1 DRC error from the command line, use the following example command steps:

1. set_drc_handling C1 Error
2. set_gate_report Error_pattern
3. set_system_mode analysis
4. report_drc_rules C1-
5. report_gates *offending_gate's_id#*
6. report_gates -Endpoints -Backward *id#_of_gate_connected_to_offending_gate's_clock*

The following transcript excerpt shows an example of the use of this command sequence starting at step 4:

report_drc_rules c1-1

```
// Error: Clock PIs off failed to force off a clock line of /U$1/DFF1
// (14). (C1-1)
```

report_gates 14

```
//  /U$1/DFF1 (14) DFF
//       "S"   I  (0) 10-
//       "R"   I  (0) 11-
//       clk   I  (X) 12-/OR/Y
//       d     I  (X) 13-/MUX/out
//       "OUT" O  (X) 8-   9-
```

report_gates -endpoints -backward 12

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  ---------------------------------------------------------------------
//  Begin backward trace for gate /OR (12).
//  ---------------------------------------------------------------------
//  /CLK (3)  PI
//      CLK     O  (0)  12-/OR/A0
//  /EN (4)  PI
//      EN     O  (X)  12-/OR/A1
//  Number gates in trace = 2.
```

Possible Resolutions

If your debugging effort shows an offending clock's off state is incorrect, use the add_clocks command to redefine the clock with the correct off state. If the off state is correct, the problem typically is due to an indeterminate value (X) in related logic that causes the clock's value at a scan cell to be X. This is the case in the example shown in the figure above. You can fix these by using the add_input_constraints command to constrain a primary input pin to a value that removes the X:

add_input_constraints -c0 EN

This enables the tool to consider the CLK signal to be the only clock signal to the clk input of the scan cell.

For further explanation and examples of possible C1 rule violations, refer to the "Clock Gaters" appendix in the *Tessent Scan and ATPG User's Manual*.

C2

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Each clock must have a structural path to the clock port of at least one memory element. The application performs this check by a backward tracing from the clock port, SET/RESET port, and read/write enables of all memory elements toward primary inputs. The rule violation occurs if a clock primary input cannot be reached.

When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command. Defining a pin to be a clock, when it does not behave as a clock, is the most usual cause of this error condition. The default handling for this rule violation in all contexts is warning. Failure to satisfy this rule indicates a defined clock cannot capture data, thus reducing test coverage.

When a floating pin is defined as a clock, it is suppressed from C2 violation when the design also defines some internal clocks. This exception prevents false C2 violations for a design with PLL clocking where the external reference clock is disconnected from the netlist in the flat model.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The occurrence message is:

```
Clock P cannot reach the clock port of any memory element. (C2-1)
```

P is the pin name of the clock.

The summary message is:

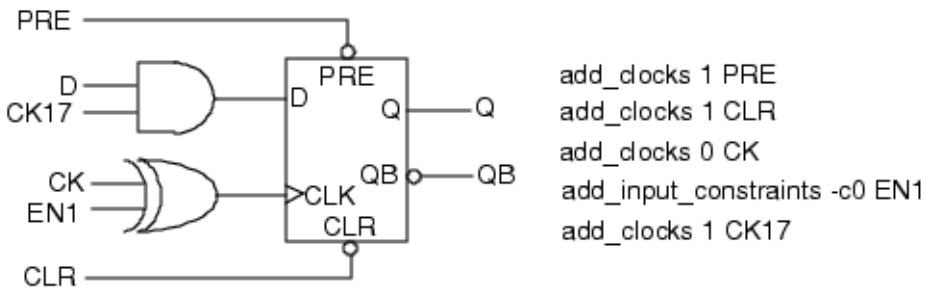
```
There were N clock rule C2 fails (reachability check).
```

N is the number of occurrences of rules violation C2.

C2 Rule Violation Example

Figure 10-11 shows an example circuit and circuit setup specified in the tool.

Figure 10-11. C2 Rule Example Circuit



If you run rules checking on this design, you get a C2 rules violation because while the CK17 signal appears to be a clock (due to its name), it cannot be reached from the clock port or SET/RESET ports of the flip-flop. To fix this problem, add the command:

delete_clocks CK17

Then, you must re-run checks.

C3

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

When a sequential element (or RAM) source and sink are clocked by the same clock and the sink captures data from the source, a potential exists for the captured data to pass through both the source and sink in the same clock cycle. That is, the sink might capture the source's new data (captured in the same cycle) instead of the source's old data (captured in the previous cycle).

Table 10-1 lists four clocking relationships that can produce this problem.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 10-1. Clocking That Can Result in a C3 Signal Race

Source	Sink
LS	LS
LS	TE
LE	LS
LE	TE

This condition violates the tool's default assumption that the sink captures only old data from the source, so results in a C3 violation and may require special handling. Failure to address this issue can result in simulation mismatches when you verify the test patterns in a timing-based simulator. Each sequential element is either level sensitive (LS), leading edge triggered (LE), or trailing edge triggered (TE).

Figure 10-12 shows an example Tessent Visualizer Flat Schematic display of a circuit segment that produces a C3 violation when the off state of the clock at the PI is defined as a logic 0. In Tessent FastScan and Tessent TestKompress, you define the off state of each clock with the add_clocks command or with analyze_control_signals -Auto_fix as a part of required tool setups. DFF1 updates on the leading edge (LE) of the clock, while DFF2 updates on the clock's trailing edge (TE) due to the inverter. Under certain timing conditions (if captured data propagates through DFF1 to its q output in less than half a clock cycle for example), data passes through both DFF1 and DFF2 in a single clock cycle.

Note

The same situation would occur if the clock pin was connected directly (rather than through an inverter) to DFF2, and DFF2 was negative-edge triggered.

Figure 10-12. C3 Violation Example

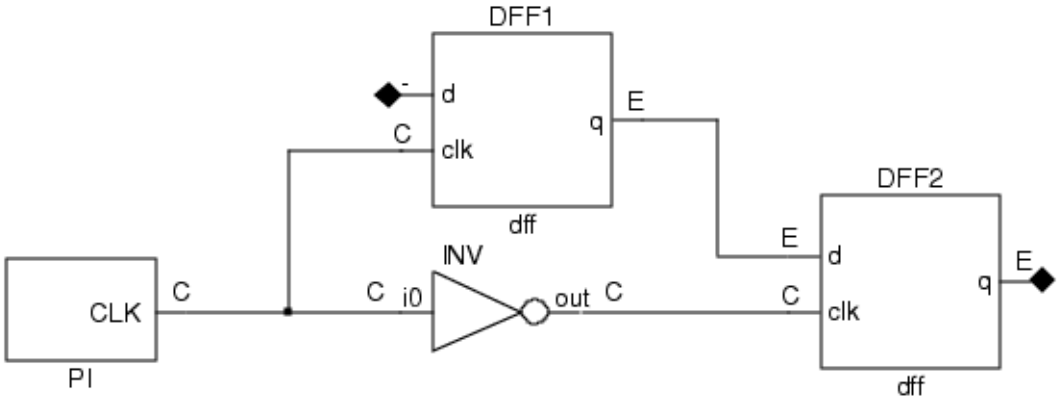
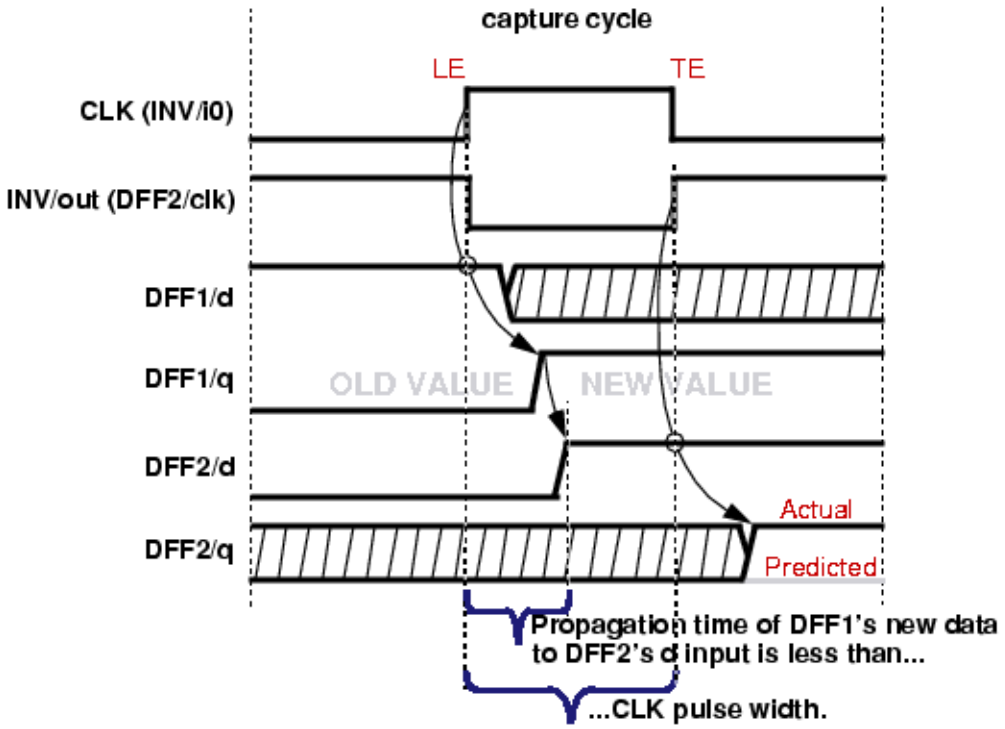


Figure 10-13 shows a possible timing scenario for the preceding circuit segment.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-13. Example Timing That Allows Sink to Capture Source's New Data



The tool performs this check by determining the forward cones of influence for a clock pin (its clock cones). For an introduction to clock and effect cones, refer to the clock cones and effect cones discussion under "Clock Signals." The bounds for the clock cones are scan cells and circuitry set to a fixed value when the constrained pins are set to their constrained values and the initialized non-scan cells are set to their stable states. The clock cone stops at read ports of RAMs that have the read_off attribute set to hold, and then the effect cone propagates from its outputs.

Note

In the C3 violation example, the data input for DFF2 is in the effect cone of CLK and its clock input is in the clock cone of CLK as indicated by the E and the C in the Flat Schematic display of Tessent Visualizer.

The rule violation may occur for a clock if one of the following is true:

- The clock input of a DFF state element is in the clock cone, its input data is in the effect cone, the element is trailing edge triggered, and the effect data comes from an element that is leading edge triggered.
- The clock input of a LA state element is in the clock cone and its input data is in the effect cone.
- The write input of a RAM is in the clock cone and a data input or address input of the associated write port is in the effect cone.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The read input of a RAM is in the clock cone and an address input of the associated read port is in the effect cone.

The tool performs a mutual exclusivity check to determine if the clock/write/read inputs associated with the failure can be active at the same time. To obtain the most benefit from this check, turn on ATPG analysis prior to DRC by issuing the set_drc_handling command with the Atpg_analysis argument. By default, the rules checker performs a partial ATPG analysis to find potential C3 rule violations. This partial analysis justifies clock/data conflicts in the affected circuitry, but stops at decision nodes, RAM, ROM, TIEX, TLA, and all other non-scan state element gates. With complete ATPG analysis explicitly turned on, the rules checker justifies the conflicting values back to PIs or scan cells.

Note

In some situations, violations of this rule may occur when there is no real problem with the design. For information on performing enhanced checking to screen out these false violations, refer to "Screening Out False C3 Violations."

Effect on Testability

Failure to satisfy this rule can result in inaccurate test pattern simulation results (simulation mismatches) during verification in a timing-based simulator and failing patterns on the tester.

How to Debug C3 Violations

For C3 violations, the tool transcribes a message similar to the following:

```
//  Note: There were N clock rule C3 fails (clock may capture data
//        affected by its captured data).
```

N is the number of times a C3 violation occurred.

Use the report_drc_rules command to obtain additional information about the violations. For example, to report the occurrence messages for all the C3s, issue "report_drc_rules C3." The occurrence messages list gate names and gate IDs you can copy and paste into commands during later debugging. You can report on a specific occurrence by issuing the command with "C3-" and the occurrence number. For example:

report_drc_rules c3-1

```
// Note:  Clock /CLK failed rule C3 on input 3 of /DFF2 (9). (C3-1)
//        Source of violation: input 3 of /DFF1 (8).
```

You can debug a specific occurrence of a C3 violation using Tessent Visualizer or by issuing commands from the tool's command line. Examples of both methods follow.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

How to Debug with Tessent Visualizer

To view the location of a C3 DRC violation using Tessent Visualizer, use the following command steps:

1. set_system_mode analysis
2. Choose **Open > DRC browser** from the Tessent Visualizer menu and double-click the desired C3 DRC violation in the table (or right-click and select **Visualize DRC** from the popup menu). The gates between the source cell and the failing cell, including the clock cone data for the failing clock, are displayed in the Flat Schematic.

Tip

Alternatively, you can issue the analyze_drc_violation command with the *rule_id-occurrence#* argument at the tool's command line. For example: analyze_drc_violation c3-1

Note

In some situations, the tool's analysis may require significant CPU run time. You can interrupt the process and return to the command prompt using the Control-C key (intermediate results are not retained if you interrupt the analysis), or you can display periodic progress using the -Interval switch with the set_drc_handling command.

How to Debug from the Tessent Tool Command Line

To view the location of the first occurrence of a C3 DRC violation from the tool command line, use the following example command steps:

1. set_system_mode analysis
2. report_drc_rules C3-*occurrence#*
3. set_gate_report Clock_cone *pin_name*
4. report_gates *source_gate_id sink_gate_id*

The following transcript excerpt shows an example of the use of this command sequence starting at step 2:

report_drc_rules c3-1

```
// Note: Clock /CLK failed rule C3 on input 3 of /DFF2 (9). (C3-1)
// Source of violation: input 3 of /DFF1 (8)
```

set_gate_report clock_cone /CLK
report_gates /DFF1 /DFF2

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /DFF1  dff
//    "IO"  I   (-)  4-
//    "I1"  I   (-)  3-
//    clk   I   (C)  2- /CLK
//    d     I   (-)  1- /C
//    "OUT" O   (E)  5-
//  /DFF2 (9) dff
//    "IO"  I   (-)  4-
//    "I1"  I   (-)  3-
//    clk   I   (C)  7- /INV/out
//    d     I   (E)  5- /DFF1/q
//    "OUT" O   (E)  6-
```

Screening Out False C3 Violations

For performance reasons, the tool's default DRC may occasionally report a false C3 violation. There are three additional analyses you can do to screen out these false C3s:

- Use the set_drc_handling command with the Atpg_analysis option. This option causes DRC to additionally check the clocks of the source and sink to see if they are gated off.
- For LSSD based designs, use the -Mode A option to the set_drc_handling command.

  When you specify this option for a selected clock, the rules checker evaluates all latches associated with the specified clock and categorizes their clock ports. It then uses these categories to determine if a violation exists. The following list describes each of the clock port categories:

  - **Inactive low (IL)** — When the selected clock is low, the clock port of the latch is inactive.
  - **Inactive high (IH)** — When the selected clock is high, the clock port of the latch is inactive.
  - **Active high slave (AHS)** — When the selected clock is high, the clock port of the latch is active. The data line of this latch connects (through buffers and inverters) to another latch called the data latch. When the clock port of the latch is active, all clock inputs of the data latch must be inactive. When the clock port of the latch is inactive, at least one clock input of the data latch must be active. Finally, non-clock primary inputs must not affect the clock inputs of the data latch.
  - **Active low slave (ALS)** — When the selected clock is low, the clock port of the latch is active. The data line of this latch connects (through buffers and inverters) to another latch called the data latch. When the clock port of the latch is active, all clock inputs of the data latch must be inactive. When the clock port of the latch is inactive, at least one clock input of the data latch must be active. Finally, non-clock primary inputs must not affect the clock inputs of the data latch.

  During this evaluation, the rules checker prints a summary message that identifies the number of latches with clock ports placed in each category. If you enable learn reporting

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

with "set_learn_report On," you can then use report_gates to report on the individual latches in these categories.

You can screen out false violations of the C3 rule by issuing the set_drc_handling command before rules checking. The command usage in this context is:

set_drc_handling C3 [-Mode A *clock_name*]

The tool ignores violations of the C3 rule if the following conditions are true:

- The failing latch port is IL or AHS, and the source latch port is IH or ALS.
- The failing latch port is IH or ALS, and the source latch port is IL or AHS.
- All clock, set, and reset inputs of the failing latch are low in the case that all defined clocks are off, the violation source clock is high, and all other clock, set, and reset inputs of the source latch are low.

C4

Category: Clock

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

When a sequential element (or RAM) source and sink are clocked by the same clock and the output of the source is connected through combinational logic to the clock port of the sink, a potential exists for the source's captured data to alter the clocking of the sink in the same clock cycle. That is, the source's new value (captured in the current cycle) might propagate through the connected logic and affect the clocking of the sink. By default, the tool expects the source's old value (captured in the previous cycle) in the downstream combinational logic.

Table 10-2 lists four clocking relationships that can produce this problem.

Table 10-2. Clocking that Can Result in a C4 Signal Race

Source	Sink
LS	LS
LS	TE
LE	LS
LE	TE

This condition violates the tool's default assumption that the sink's clocking is affected by only old data from the source, so results in a C4 violation and may require special handling. Failure to address this issue can result in simulation mismatches when you verify the test patterns in a timing-based simulator. Each sequential element is either level sensitive (LS), leading edge triggered (LE), or trailing edge triggered (TE).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-14 shows an example Tessent Visualizer Flat Schematic display of a circuit segment that produces a C4 violation when the off state of the clock at the PI is defined as a logic 0. In Tessent FastScan and Tessent TestKompress, you define the off state of each clock with the add_clocks command or with analyze_control_signals -Auto_fix as a part of required tool setups.

Figure 10-14. C4 Violation Example

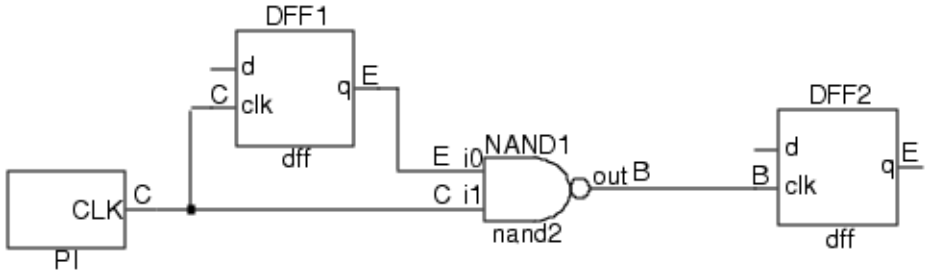
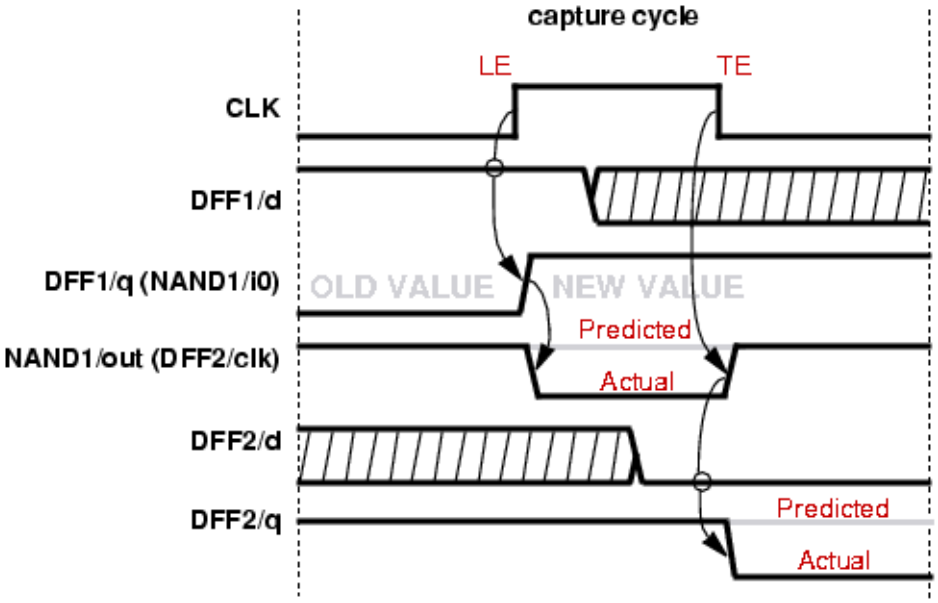


Figure 10-15 shows a possible timing scenario for the preceding circuit segment, where DFF1 updates on the leading edge (LE) of the clock, while DFF2 updates on the clock's trailing edge (TE) due to the gating of the CLK signal with the output of DFF1. Notice that the old value of DFF1/q disables the clock input to DFF2. The tool's default simulation therefore predicts there is no clock pulse to DFF2 and that it holds its state as a result. But the leading edge of CLK almost immediately enables the clock gate and DFF2's clock port does receive a pulse. So rather than holding state as the tool expects, DFF2 captures its D input value on the TE of the now enabled clock gate.

Figure 10-15. Example Where Actual Behavior Differs from Tool's Prediction



The tool performs this check by determining the forward cones of influence for a clock pin (its clock cones) and the forward cones of influence for each scan cell influenced by the clock pin

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

(effect cones). For an introduction to clock and effect cones, refer to the clock cones and effect cones discussion under "Clock Signals." The bounds for the cones are scan cells and circuitry set to a fixed value when the constrained pins are set to their constrained values and the initialized non-scan cells are set to their stable states.

Note

In the C4 violation example, the clock input of DFF2 is in both the clock and effect cones of CLK as indicated by the B in the Flat Schematic window display. This is the main difference from a C3 violation in which the clock input of the sink is in the clock cone but not in the effect cone and the data input is in the effect cone.

The rule violation occurs for a clock if one of the following is true:

- The clock input of a DFF state element is in both the clock and effect cones, the element is trailing edge triggered, and the effect data comes from an element that is leading edge triggered.
- The clock input of a LA state element is in both the clock and effect cones.
- The write input of a RAM is in both the clock and effect cones.
- The read input of a RAM is in both the clock and effect cones.

The tool performs a mutual exclusivity check to determine if the clock/write/read inputs associated with the failure can be active at the same time. To obtain the most benefit from this check, turn on ATPG analysis prior to DRC by issuing the set_drc_handling command with the Atpg_analysis argument. By default, the rules checker performs a partial ATPG analysis to find potential C4 rule violations. This partial analysis justifies clock/data conflicts in the affected circuitry, but stops at decision nodes, RAM, ROM, TIEX, TLA, and all other non-scan state element gates. With complete ATPG analysis explicitly turned on, the rules checker justifies the conflicting values back to PIs or scan cells.

Note

In some situations, violations of this rule may occur when there is no real problem with the design. For information on performing enhanced checking to screen out these false violations, refer to "Screening Out False C4 Violations."

How to Debug C4 Violations

For C4 violations, the tool transcribes a message similar to the following:

```
//  Warning: There were N clock rule C4 fails (clock may be affected by its
//  captured data).
```

N is the number of times a C4 violation occurred.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Use the report_drc_rules command to obtain additional information about the violations. For example, to report the occurrence messages for all the C4s, use:

report_drc_rules c4

The occurrence messages list gate names and gate IDs you can copy and paste into commands during later debugging. You can report on a specific occurrence by issuing the command with "c4-" and the occurrence number. For example:

report_drc_rules c4-1

```
//  Warning: Clock /CLK failed rule C4 on input 3 of /DFF2 (9). (C4-1)
//  Source of violation: input 3 of /DFF1 (8).
```

You can debug a specific occurrence of a C4 violation using Tessent Visualizer or by issuing commands from the tool's command line.

How to Debug with Tessent Visualizer

To view the location of a C4 DRC violation using Tessent Visualizer, use the following command steps:

1. set_system_mode analysis
2. open_visualizer

Choose the **DRC Violations** tab in the Design Browser, then locate the DRCs of interest in the table. Double clicking the listed violation or right-clicking and selecting **Visualize DRC** from the popup menu analyzes the violation. The gates between the source cell and the failing cell, including the clock cone data for the failing clock, are displayed in the Flat Schematic.

Tip

Alternatively, you can issue the analyze_drc_violation command with the *rule_id-occurrence#* argument at the tool's command line. For example:
analyze_drc_violation c4-1

Note

In some situations, the tool's analysis may require significant CPU run time. You can interrupt the process and return to the command prompt using the Control-C key (intermediate results are not retained if you interrupt the analysis), or you can display periodic progress using the -Interval switch with the set_drc_handling command.

How to Debug from the Tool Command Line

To view the location of the first occurrence of a C4 DRC violation from the tool command line, use the following example command steps:

1. set_system_mode analysis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

2. report_drc_rules C4-*occurrence#*
3. set_gate_report Clock_cone *pin_name*
4. report_gates *source_gate_id sink_gate_id*

The following transcript excerpt shows an example of the use of this command sequence starting at step 2:

report_drc_rules c4-1

```
// Warning: Clock /CLK failed rule C4 on input 3 of /DFF2 (10). (C4-1)
// Source of violation: input 3 of /DFF1 (9)
```

set_gate_report clock_cone /CLK
report_gates /DFF1 /DFF2

```
//  /DFF1 dff
//      clk    I  (C)  /CLK
//      d      I  (-)  1-
//      q      O  (E)  5-/DFF2/d /NAND1/i0
//  /DFF2 dff
//      clk    I  (B)  7-/NAND1/out
//      d      I  (-)  5-/DFF1/q
//      q      O  (-)  6-/q2
```

You can see that the DRC violation is occurring because the clock input of /DFF2 is in both the clock and effect cones of /CLK.

Screening Out False C4 Violations

For performance reasons, the default DRC may occasionally report a false C4 violation. There are three additional analyses you can do to screen out these false C4s:

- Use the set_drc_handling command with the Atpg_analysis option. This option causes DRC to additionally check the clocks of the source and sink to see if they are gated off.
- For LSSD based designs, use the -Mode A option to the set_drc_handling command.

  When you specify this option for a selected clock, the rules checker evaluates all latches associated with the specified clock and categorizes their clock ports. It then uses these categories to determine if a violation exists. The following list describes each of the clock port categories:

  - **Inactive low (IL)** — When the selected clock is low, the clock port of the latch is inactive.
  - **Inactive high (IH)** — When the selected clock is high, the clock port of the latch is inactive.
  - **Active high slave (AHS)** — When the selected clock is high, the clock port of the latch is active. The data line of this latch connects (through buffers and inverters) to another latch called the data latch. When the clock port of the latch is active, all

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

clock inputs of the data latch must be inactive. When the clock port of the latch is inactive, at least one clock input of the data latch must be active. Finally, non-clock primary inputs must not affect the clock inputs of the data latch.

- **Active low slave (ALS)** — When the selected clock is low, the clock port of the latch is active. The data line of this latch connects (through buffers and inverters) to another latch called the data latch. When the clock port of the latch is active, all clock inputs of the data latch must be inactive. When the clock port of the latch is inactive, at least one clock input of the data latch must be active. Finally, non-clock primary inputs must not affect the clock inputs of the data latch.

During this evaluation, the rules checker prints a summary message that identifies the number of latches with clock ports placed in each category. If you enable learn reporting with "set_learn_report on," you can then use report_gates to report on the individual latches in these categories.

You can screen out false violations of the C4 rule by issuing the set_drc_handling command before rules checking. The command usage in this context is:

set_drc_handling C4 [-Mode A *clock_name*]

The tool ignores violations of the C4 rule if the following conditions are true:

- The source latch port is IL or AHS and all paths from the source latch to the failing latch are blocked when the selected clock is high.
- The failing latch port is IH or ALS and all paths from the source latch to the failing latch are blocked when the selected clock is low.
- The violation source clock input is high and all other clock, set, and reset inputs are low for the source latch.

C5

Category: Clock

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

A clock pin must not be capable of simultaneously capturing data on multiple ports of the same scannable memory element. The application performs this check by determining the forward cone of influence for a clock pin (clock cone). The bounds for the cone of influence are scan cells and circuitry set to a fixed value when constrained pins are set to their constrained value, and initialized non-scan cells are set to their stable state.

The rule violation occurs on a clock pin when multiple clock inputs of a scannable memory element are in the same clock cone and the clock inputs may be on at the same time. The tool performs a mutual exclusivity check to determine if both clock inputs associated with the failure

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

can be active at the same time. If the justification results in a conflict without justifying decision nodes, it is not considered a rules violation.

The default handling for this rule violation is warning. Failure to satisfy this rule may result in a race condition that creates inaccurate simulation results. When an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, E indicates effect cone, B indicates both, and "-" indicates no cone.

The occurrence message is:

```
Clock P failed rule C5 on input I of N (G). (C5-1)
```

P is the pin name of the clock, C5 is the rule ID number, I is the gate input number of the clock line, N is the instance name of the gate, and G is the gate ID number.

The summary message is:

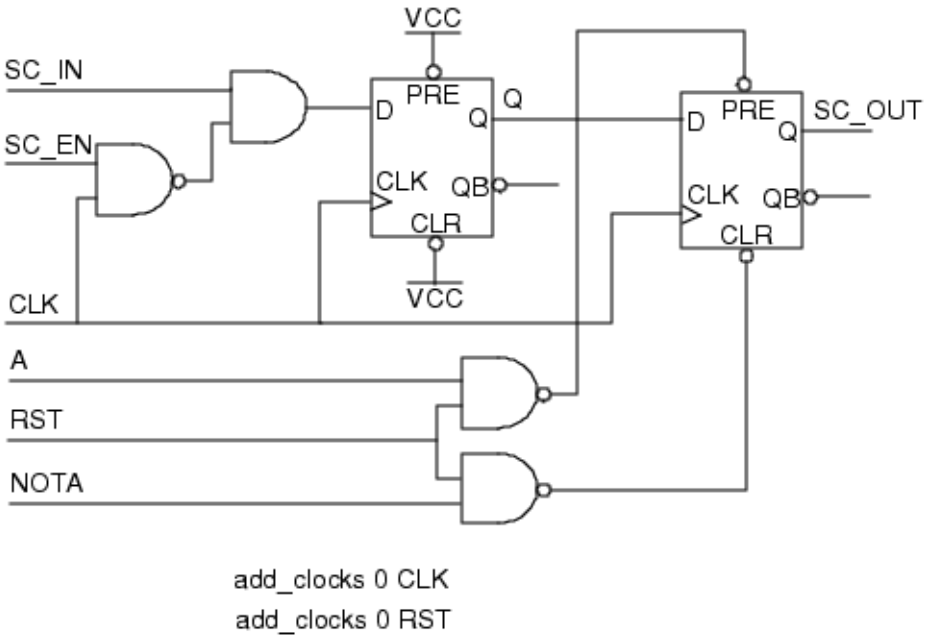
```
There were N clock rule C5 fails (clock is connected to multiple ports of
same latch).
```

N is the number of occurrences of rules violation C5.

C5 Rule Violation Example

Figure 10-16 shows an example circuit and circuit setup specified in the tool.

Figure 10-16. C5 Rule Example Circuit



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If you run rules checking on this design given the setup commands shown, you get a C5 rules violation. In this design, the RST signal connects to both the PRE and CLR pins of the second flip-flop. If you examine the inputs, you should see that A and NOTA should always have opposite values. If the tool knows this information, it knows that only one of the CLR or PRE signals can be active at any given time. To specify this information and fix the C5 rules violation, add the command:

add_input_constraints a -inv nota

C6

Category: Clock

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

A clock must not affect data that it is capturing. If it does, a race condition may result that produces inaccurate simulation results.

The application performs this check by determining the forward cone of influence for a clock pin (clock cone). The bounds for the clock cone are scan cells and circuitry set to a fixed value when constrained pins are set to their constrained values and initialized non-scan cells are set to their stable states. The rule violation occurs on a clock pin when a clock input of a scannable memory element and its data line are in the same clock cone.

The default handling for this rule violation is warning. When a violation occurs, you can access the clock cone data by the command "set_gate_report drc C6-N," where N is the c6 fail occurrence number, then issuing the report_gates command for the gate ID number displayed in the occurrence message. This identifies the problem input. By tracing back from this input, you can identify how to correct the problem. C indicates clock cone, E indicates effect cone, B indicates both, and "-" indicates no cone.

By default, C6 analysis treats clocks in the same synchronous clock group as if they are equivalent. You can use the "set_drc_handling C6 -ignore_synchronous_clock_groups on" command to turn off this behavior.

For more information on synchronous clock groups, see add_synchronous_clock_group.

Note

Default C6 checking may miss some C6 violations when "set_clock_restriction off" is used. The "set_drc_handling C6 -CONSERvative ON" command enables a more conservative analysis to identify those C6 violations. Siemens EDA recommends using "set_drc_handling C6 -CONSERvative ON" instead of turning off clock restrictions.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The occurrence message is:

```
Clock P failed rule C6 on input I of N (G). (C6-1)
```

P is the pin name of the clock, C6 is the rule ID number, I is the gate input number of the clock line, N is the instance name of the gate, and G is the gate ID number.

The summary message is:

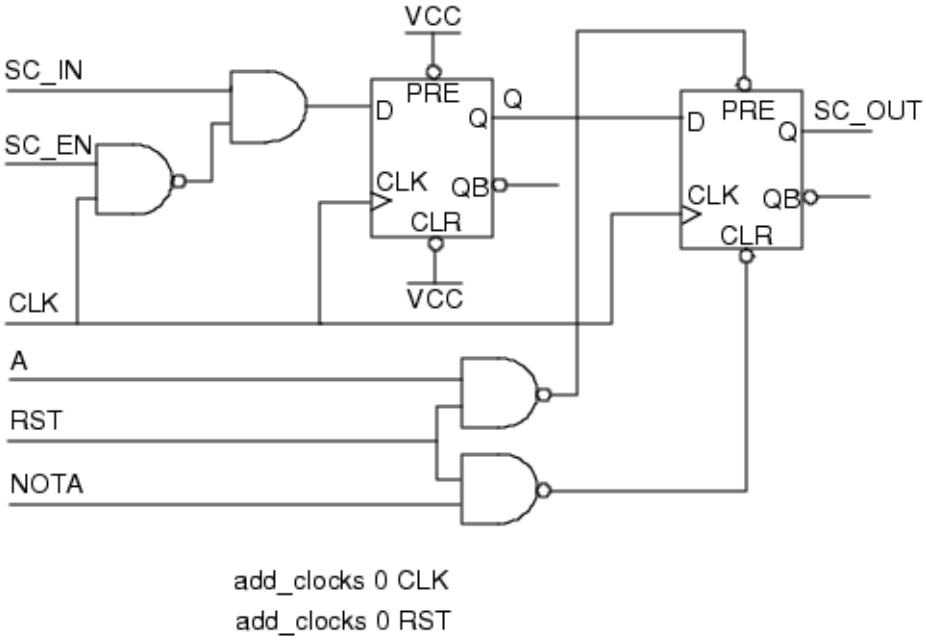
```
There were N clock rule C6 fails (clock may capture data affected by
itself).
```

N is the number of times a C6 rule violation occurred.

C6 Rule Violation Example

Figure 10-17 shows an example circuit and circuit setup specified in the tool.

Figure 10-17. C6 Rule Example Circuit



If you ran rules checking on this design, given the setup commands shown, you would get a C6 rules violation. The default handling for this rule violation is warning. In this design, the CLK signal goes to both the CLK and D inputs of the first flip-flop. Thus, data can be captured in this flip-flop that may be affected by the capturing clock. To prevent the CLK signal from influencing the data, add the command:

add_input_constraints -c0 SC_EN

Constraining the SC_EN signal to 0 ensures that changes in the clock do not change the data.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For ATPG, you can handle C6 rule violations by issuing a “set_clock_off_simulation on” command prior to creating patterns. See the set_clock_off_simulation command description for additional information.

In some designs, different flip-flops may capture the off state and on state of the clock, causing the simulation to fail. In this case, you can use the following command to mask the signal creating the C6 violation:

set_simulation_options -C6_mask_races ON

For more information, see the set_simulation_options command.

C7

Category: Clock

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Each clock input (not including set and reset lines) of a scan or non-scan cell memory element must be capable of capturing data when a single clock primary input line is on and all other clocks are off. It is acceptable that this may require placing values on non-clock primary inputs or scan cells. The application performs this check using the simulated values that result when one clock is set to X, all other defined clocks are at their off-state, the constrained pins are set to their constrained values, and the initialized non-scan cells are set to their stable states. The rule violation occurs when a clock input of a scan cell always remains off.

The default handling for this rule violation is warning. Failure to satisfy this rule indicates a scan cell clock input cannot capture data, resulting in some loss of test coverage.

The occurrence message is:

```
Clock input I of N (G) cannot capture data with a single  clock on. (C7-1)
```

I is the input number, N is the instance name, G is its gate index number, and C7 is the rule ID number.

The summary message is:

```
There were N clock rule C7 fails (scan cell capture ability  check).
```

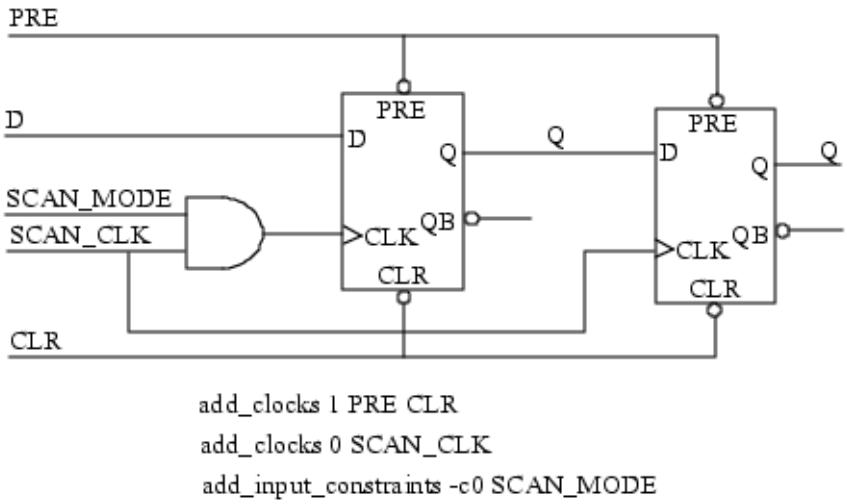
N is the number of occurrences of rules violation C7.

C7 Rule Violation Example

Figure 10-18 shows an example circuit and circuit setup specified in the tool.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-18. C7 Rule Example Circuit



If you run rules checking on this design given the setup commands shown, you get a C7 rules violation. This type of error commonly occurs when incorrect clock or set/reset gating occurs in the design. This design constrains the SCAN_MODE signal to a constant 0 during ATPG. This constraint prevents the first flip-flop from ever being clocked, and from ever capturing data. To fix this problem, delete the pin constraint:

delete_input_constraints -all

C8

Category: Clock

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

You may not directly connect a clock to a primary output (PO). The application performs this check by determining the forward cone of influence for a clock pin (clock cone). The bounds for the cone of influence are scan cells and circuitry set to a fixed value when constrained pins are set to their constrained values, and initialized non-scan cells are set to their stable states. The rule violation occurs when a primary output is in the clock cone.

The default handling for this rule violation is warning. Failure to satisfy this rule results in the occasional usage of a different type of scan pattern, in which the tool observes only the POs directly connected to clocks. There is no loss of test coverage or risk of inaccurate simulation results. When an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, E indicates effect cone, B indicates both, and “-” indicates no cone.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The occurrence message is:

```
Primary output P is connected to clock C. (C8-1)
```

P is the pin name of the primary output and C is the pin name of the clock.

The summary message is:

```
There were N clock rule C8 fails (PO connected to a clock  line).
```

N is the number of occurrences of rules violation C8.

C8 Rule Violation Example

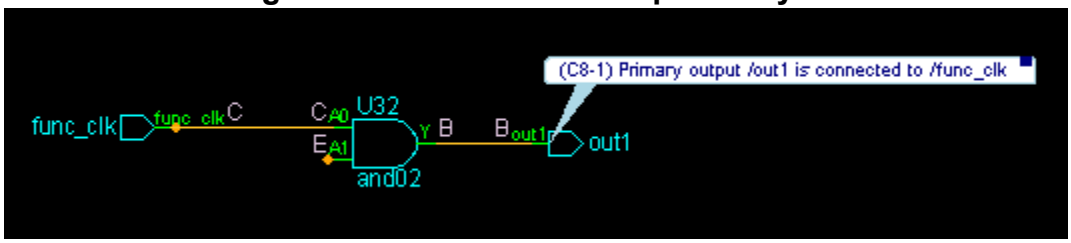
The tool setup for the clocks is:

add_clocks 0 func_clk

If you run rules checking on this design given the setup command shown, you get a C8 rule violation. This type of violation is not usually a problem.

Figure 10-19 shows the results of the C8 DRC analysis in Tessent Visualizer.

Figure 10-19. C8 Rule Example Analysis



In this example, C8 is caused by the connection of the primary output, out1, through U32 to a clk, func_clk. An acceptable solution is to accept this warning, or turn it off with the command:

set_drc_handling c8 ignore

C9

Category: Clock

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Data captured by any clock with a direct path (through combinational logic only) to a primary output must not affect the direct path to a primary output of that same clock. The application performs this check by determining the forward cone of influence for a clock pin (clock cone) and for each scannable memory element influenced by the clock pin (effect cone). The bounds

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

for the cones of influence are scan cells and circuitry set to a fixed value when constrained pins are set to their constrained values and initialized non-scan cells are set to their stable states. The rule violation occurs on a clock pin when a primary output is in both the clock cone and the effect cone.

The default handling for this rule violation is warning. Failure to satisfy this rule may result in a small loss in test coverage. There is no risk of inaccurate simulation results. When an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, E indicates effect cone, B indicates both, and "-" indicates no cone.

The occurrence message is:

```
PO P path from clock C is gated by scan cell that uses same  clock. (C9-1)
```

P is the pin name of the primary output and C is the pin name of the clock.

The summary message is:

```
There were N clock rule C9 fails (PO connected to a clock  line gated by
scan  cell that uses same clock).
```

N is the number of occurrences of rules violation C9.

C9 Rule Violation Example

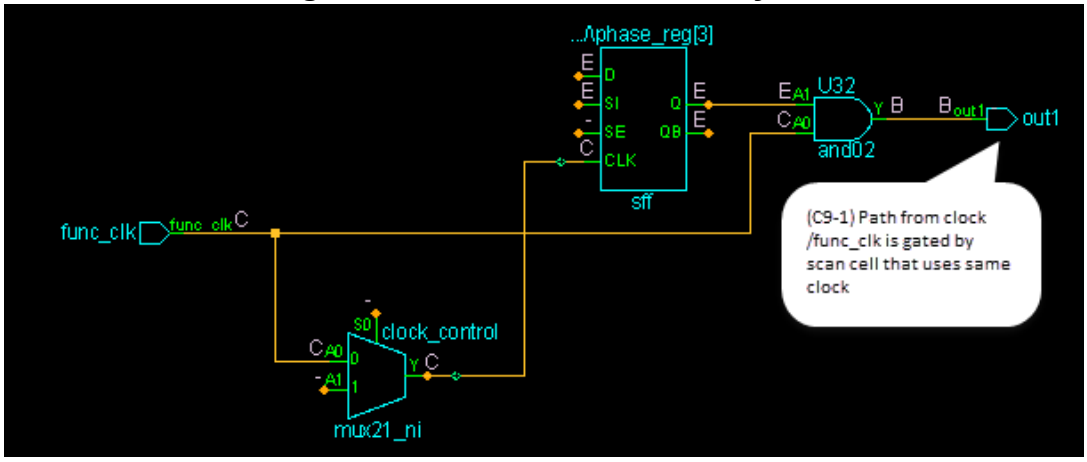
The tool setup for the clocks is:

add_clocks 0 func_clk

If you run rules checking on this design given the setup command shown, you get a C9 rule violation. The results of performing DRC Analysis in Tessent Visualizer on the C9 DRC in this example is shown in Figure 10-20

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-20. C9 Rule DRC Analysis



This type of violation is not usually a problem. However, a C9 violation can result in reduced coverage because it may introduce sequential effects into the generated clock patterns. In this case, the violation is at the out1 line. The func_clk signal can affect the data of the flip-flop and the gate at the output of the scan cell. An acceptable solution is to accept this warning, or turn it off with the command:

set_drc_handling c9 ignore

C10

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

A sequential element (latch or flip-flop) can only be clocked once in any one pattern cycle. The tool only pulses a clock primary input once in each cycle, or apply a clock procedure only once, therefore a failure of this rule implies that the circuit generates two or more internal pulses from a single clock pulse or clock procedure.

The default handling of this rule violation is error. Failure to satisfy this rule results in creation of patterns which are likely to be incorrect and to fail both in verification and on the tester.

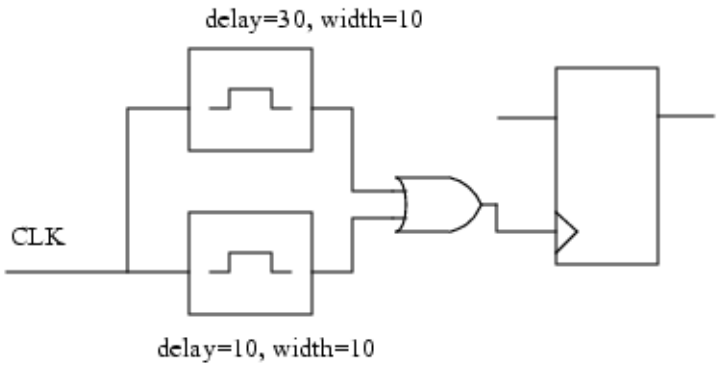
The occurrence message is:

```
Cell c might capture more than once by applying clock ck. (C10-1)
```

Figure 10-21 shows an example circuit which generates a C10 error.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-21. C10 Rule Example Circuit



In order to remove the violation, the circuit must be modified. Test logic can be added to block the path from one pulse generator to the OR gate during test mode.

Note

It is also possible to violate this rule by OR-ing together two clocks which are pulsed at different times in a clock procedure.

C16

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

When switching from the load_unload phase to the capture phase, sometimes the loading values of scan cells can be disturbed due to pin constraints specified in the capture phase. The C16 clock rule checks for this situation and issues a violation if it occurs.

For example, consider a DFF scan cell with two clock ports: the first port is used for scan loading and the second is used for normal operation. Assume the second port's clock signal (whose off state is defined as 1) is ANDed with an enable signal to produce the actual clock input value. If in the load_unload procedure the enable is forced to 0, the load value cannot be disturbed by the second clock port during scan loading. However, if you ran an add_input_constraints command in setup mode to set the enable signal to 1, the scan load value can be disturbed at the start of the capture cycle when switching from load_unload to capture. This is because the clock input at the second port changes from 0 to 1 when the enable changes from 0 to 1 as a result of the pin constraint.

Note

The C16 rule check ignores an X at a scan cell's clock port, as it is possible to assign a proper value at the PIs to keep the scan cell stable at the beginning of the capture phase. Xs on scan cell clock ports, however, cause C1 violations.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The default handling for rule C16 is error. Failure to satisfy this rule may result in simulation mismatches during verification.

To debug a C16 rule violation, issue a "set_gate_report error_pattern" command, then use report_gates for the gate instance named in the C16 occurrence message.

Caution

If you choose to ignore C16 violations (not recommended), you should add a CX cell constraint to each scan cell where a violation occurred—in order to avoid simulation mismatches. Be aware that each cell you constrain in this manner reduces test coverage. You can obtain the instance name of the scan cell from the occurrence message and use the add_cell_constraints command to apply the constraint to the cell's output pin. For example:

```
report drc rule c16

// Warning: The load value of scan cell /dff2/dff1/Q_reg (47) is disturbed
// by clock input 3 at the beginning of the capture phase when pin
// constraints are forced. (C16-1)

add_cell_constraints /dff2/dff1/Q_reg/Q CX
```

The occurrence message is:

```
The load value of scan cell N is disturbed by clock input I at the
beginning of the capture phase when pin constraints are forced. (C16-1)
```

N is the instance name of the gate, I is the gate input number of the clock line, and C16 is the rule ID number.

The summary message is:

```
There were N clock rule C16 fails (scan cell load value disturbed at
beginning of capture phase when pin constraints are forced).
```

N is the number of occurrences of rules violation C16.

C17

Category: Clock

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore for patterns -scan context. Warning for dft -scan and dft -test_points contexts.

report_drc_rules: Supported

The default handling for the C17 rule in patterns -scan context is Ignore. Setting the handling to Ignore bypasses the C17 rule completely, cutting run time.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The default handling for the C17 rule in the dft -scan and dft -test_points contexts is Warning. When handling is set to Warning, all internally-driven glitches are identified (as C17 violations) and are also remodeled as possible X sources during DRC and pattern generation.

You can change the DRC handing for C17 in any of the supported contexts. In addition to setting handling to Ignore and Warning, you can also set handling to Error or Note. Setting handling to Error reports the first violation and then aborts the system mode transition. When handling is set to Note, all internally-driven glitch sources in a design are identified (as C17 violations), but they are not remodeled and maintain a constant value during DRC and pattern generation.

An internally driven glitch source consists of either a two-input AND gate, or a two-input OR gate where both inputs are driven from a common source (either directly or through buffers and inverters), such that one path is inverting and the other path is not. This type of circuit maintains the same constant output value regardless of the state of the driving node, but the output probably contains a glitch when the driving node changes state.

With this rule, gates are identified as internally-driven glitch sources if they are both driven by a sequential element and also drive the clock or set/reset line of a sequential element.

Note

Internally-driven is defined as a glitch generator driven from a sequential element, and not from a primary output.

Remodeling these internally-driven glitch sources as a constant value (TIE0 or TIE1) may cause failures during scan chain tracing. For this reason, if handling is set to Warning, C17 identifies these internally-driven glitch sources and remodels them as an X source (TIEX gate).

You can use the report_drc_rules command to display a list of the internally-driven glitch sources identified in your design.

C19

Category: Clock

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

The C19 rule checks any clock gating structure where a leading edge (LE) or trailing edge (TE) flop is gating a clock path that drives any state elements that can capture a value at the same edge.

Note that the violation can lead to simulation mismatch due to a race condition between the clock enable path and the data ports of the controlled state elements. You should investigate and fix C19 violations by changing the design or adding constraints to prevent the clock enable

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

signal from changing the value while the data ports of the controlled state elements are capturing data.

To debug a C19 rule violation, issue the analyze_drc_violation command to display the following in the Flat Schematic window:

- path from the clock enable flop to the driving flop with the violation
- path from clock pin to enable flop
- constraint value
- clock enable flop and driving flop with the violation

Example of C19 Violation

The following example shows a design with 10 C19 violations:

SETUP> set_system_mode analysis
...

```
//  10 sequential elements are gating clock path and driving sequential
//  elements on the same clock edge which can create a race condition.
//  (C19)
```

ANALYSIS> report_drc_rules c19-2

```
//  Warning: Active-high latch /Ibasic_gates/Igated_C3/ (30365) is gating
//  clock pin /clk_3 (17) and driving some leading-edge flops which can
//  create a race condition. (C19-2)

//  The first traced sequential element driven by /Ibasic_gates/Igated_C3/
//  (30365) that can create a race condition is
//  /Ibasic_gates/Iflop2o_ne_p23s_o_2S3/ (30334).
```

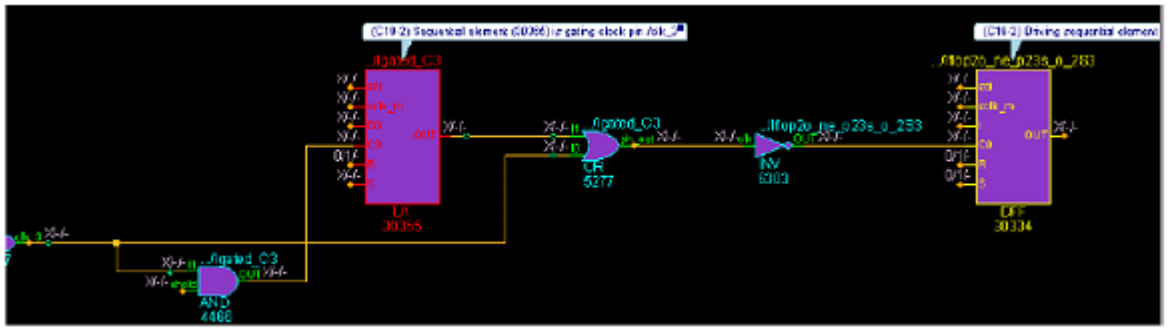
ANALYSIS> report_drc_rules
...

```
C19: #fails=10  handling=warning/atpg (clock gater race condition check)
...
```

ANALYSIS> analyze_drc_violation C19-1

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This command produces the following display in the Flat Schematic window:



C21

Category: Clock

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

The C21 DRC detects when multiple active shift clocks converge at the same combinational gate.

When two or more active shift clocks converge at a combinational gate pin, and there is an unblocked path to at least one sequential element, the tool issues the C21 DRC violation. C21 identifies the gate pin location and lists the clocks that converge at that gate pin.

The default handling of C21 is Note, but you can set it to Warning, Ignore, or Error.

The occurrence message is:

```
//  Error: At least 2 active shift clocks (/clk1, /clk2) converge at gate
//         '/gate13/Y' (33) during shift. (C21-1)
```

The summary message is:

```
C21:  #fails=1 handling=error/verbose (multiple shift clocks converge at
      same gate)
```

C22

Category: Clock

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The C22 DRC reports a violation when the tool identifies asynchronous cross domain clock paths where the launch and capture clocks are forced to pulse, which could lead to simulation mismatches.

C22 checks a pair of clocks, one of which must be pulse-always or pulse-in-capture, whether there is any interaction between the pair and they are not defined within a synchronous clock group.

C22 also checks if any interaction exists between an internal clock of a clock_control procedure and its source clock when the clocks are not defined within a synchronous clock group. In such a case, C22 reports a violation.

When you issue a read_procfile command that adds, deletes, or modifies the clock_control procedures, the C22 DRC is re-analyzed and any false paths derived by C22 are updated.

Figure 10-22 and Figure 10-23 include examples of cross clock domain paths.

The default C22 DRC handling is Warning, and can be set to Error, Warning, Note, or Ignore.

The occurrence message is similar to the following:

```
//  Warning: Pulse-always clock '/clk2' (2) interacts with clock domain
//  '/clk1' (1) and may cause simulation mismatches. Use
//  "report_false_paths -verbose -drc C22-1" to see the clock
//  interactions. (C22-1)
```

The summary message is:

```
//  Warning: There were 2 C22 violations (interacting clocks not in
//  synchronous clock group but forced to pulse concurrently)
```

Unless the C22 handling is set to Error, the tool adds a false path between the pair of clocks to prevent a simulation mismatch due to C22. If set to Ignore, no false paths are added, because no C22 is flagged. The -mask_interactions and -interactions switches of the set_drc_handling command enable you to disable the false path addition and to determine the type of clock interaction for C22.

You can report the number of sequential elements impacted by each C22 violated clock pair using this command in ANALYSIS mode:

report_clock_domains -race_points clk1 clk2

```
//  /clk_gater_2/pipe_dff1_1/ (98) scan
//  1 state element causes clocks '/clk1' (7) and '/clk2' (8) to be
//  incompatible.
```

With the information returned by the report_clock_domains command, you can trace backwards to locate the clocks using the report_gates command or within Tessent Visualizer. Figure 10-22 shows the most basic C22 DRC violation example. The C22 violation is between CLK1 (a pulse-always clock) and CCB2 because of the cross domain path from SFF1 to SFF2.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-22. C22 DRC Example Design 1

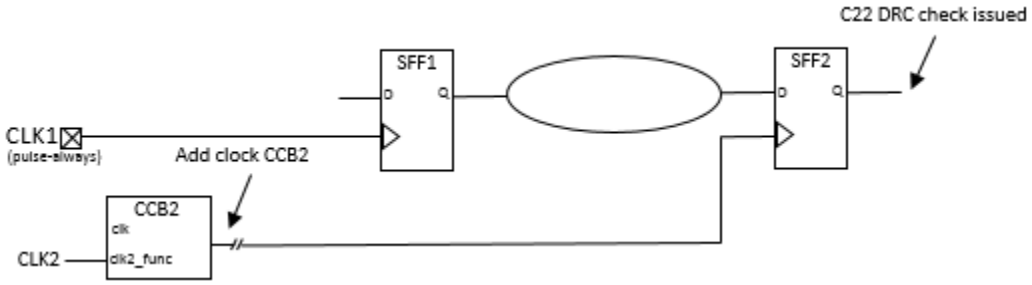


Figure 10-22 also demonstrates the interaction of an internal clock of a clock_control procedure (CCB2) and its source (CLK1) when the clocks are not defined within a synchronous clock group. This is the clock_control procedure for this example:

```
clock_control CCB2 =
  source_clock CLK1;
  atpg_cycle 0 =
   condition /sff2_CC/Q 1;
  end ;
  atpg_cycle 1 =
   condition /sff1_CC/Q 1;
  end;
end;
```

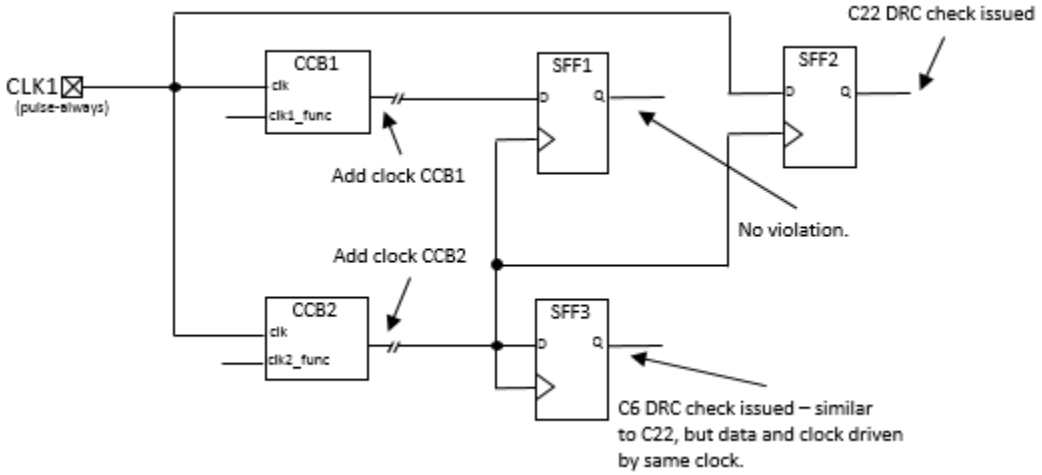
When a source and internal clock pair are defined in a clock_control procedure, it is not necessary for the source (CLK1, in this case) to be defined as a pulse-always or pulse-in-capture for C22 to trigger a violation.

Figure 10-23 shows another C22 DRC violation example. The C22 violation is between CLK1 and CCB2 and their interaction on SFF2. SFF1 does not violate C22 because it is driven by the added clocks CCB1 and CCB2, neither of which is pulse-always nor pulse-in-capture.

A C6 DRC is issued for SFF3 and is shown in this example to demonstrate that while C22 is similar to C6, C22 flags different clocks and C6 involves the same clock.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-23. C22 DRC Example Design 2



Possible Resolutions For C22 Violation

One possible resolution is to change the design to eliminate the cross domain path. If the clocks are synchronous, another solution would be to use the add_synchronous_clock_group command to place the clock pair in the same synchronous clock group in Setup mode.

C23

Category: Clock

Contexts Supported: dft -edt, dft -logic_bist, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

This rule checks if a scan cell cannot capture fault effects because its clock port is constrained to 0 or 1.

The information provided by C23 helps you debug low fault coverage caused by clocking issues. To check for C23, the tool forces and propagates constant values such as pin constraints, tied non-scan elements, and constant cell constraints. It also calculates forbidden values. The tool then processes each clock port, except those meant to be used only for scan shift, and checks for these conditions:

- Are the clock ports of level-sensitive latches constrained to 0?
- Are the clock ports of edge-triggered DFFs constrained to 0 or 1?

If all the clock ports of a scan cell satisfies either of these conditions, a C23 violation is reported because none of its clock ports can be turned on.

To minimize the number of violations reported, the tool traces from rule violation clock ports to a common source point, which can be a primary input or internal pin. The violating sources are

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reported in order by number of affected sequential elements with the first violation having the most, and so on. This is an example of the reported message:

report_drc_rule C23

```
// Warning: Primary input '/clk (4)' has a constrained value of 0. This
// prevents capture by 32 scan cell elements, one of which is
// '/inst/sdff1 (880)'.  (C23-1)

// Warning: Internal pin '/inst1/udp/A (567)' has a constrained value of
// 1. This prevents capture by 18 scan cell, one of which is
// '/inst/sdff2 (990)'.  (C23-2)
```

The default handling of C23 is Warning. It is performed during system mode transition from Setup to Analysis. The summary message is:

```
// Warning: There are N clock rule C23 fails (scan cell elements cannot
// capture or launch transition because clock ports are constrained).
```

To debug C23, use the "set_gate_report -constrain_value on" command to view the constrained values for the clock ports of affected scan cells.

C24

Category: Clock

Contexts Supported: dft -edt, dft -logic_bist, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

This rule checks if a scan cell cannot capture fault effects because its clock port cannot be set to an active state. A scan cell must first pass the C23 DRC in order to be checked by C24.

The information provided by C24 helps you debug low fault coverage caused by clocking issues. To check for C24, the tool forces and propagates constant values such as pin constraints, tied non-scan elements, and constant cell constraints. It also calculates forbidden values. The tool then processes each clock port, except those meant to be used only for scan shift, and checks these for conditions:

- The constrained value of the clock ports of level-sensitive latches is X, but its forbidden value is 1, or the clock cannot be justified to 1.
- The constrained value of the clock ports of edge_triggered latches is X, but its forbidden value is either 0, 1, or both, or the clock cannot be justified to both 0 and 1.

If all the clock ports of a scan cell satisfies either of these conditions, a C24 violation is reported because none of its clock ports can be turned on.

To minimize the number of violations reported, the tool traces from rule violation clock ports to a common source point, which can be a primary input or internal pin. The violating sources are

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reported in order by number of affected sequential elements with the first violation having the most, and so on. This is an example of the reported message:

report_drc_rule C24

```
// Warning: Primary input '/clk (4)' has a value of X and cannot be
// justified to 1. This prevents capture by 32 scan cell elements, one of
// which is '/inst/sdff1 (880)'.  (C24-1)

// Warning: Internal pin '/inst1/udp/A (567)' has a value of X and cannot
// be justified to both 0 and 1. This prevents capture by 18 scan cell
// elements, one of which is '/inst/sdff2 (990)'.  (C24-2)
```

The default handling of C24 is Warning. It is performed during system mode transition from Setup to Analysis. The summary message is:

```
// Warning: There are N clock rule C24 fails (scan cell elements cannot
// capture or launch transition because clock ports cannot be fully
// controlled).
```

To debug C24 violations, use the "set_gate_report -constrain_value on" command to look at the constrained values on clock ports of affected scan cells.

C25

Category: Clock

Contexts Supported: dft -edt, dft -logic_bist, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

This rule checks if a scan cell cannot capture fault effects because its asynchronous controls (SET and RESET ports) cannot be set to the off state.

To check for C25, the tool forces and propagates constant values such as pin constraints, tied non-scan elements, and constant cell constraints. It also calculates forbidden values. The tool then processes the set and reset ports of every scan cell and checks for these conditions:

- If the port's constrained value is 0, it passes.
- If the port's constrained value is a 1, it fails because the scan cell is always setting or resetting and is unable to capture fault effects through its data port.
- If the ports constrained value is X, and its forbidden value is 0 or cannot be justified to 0, it fails, because the port cannot be at its off state. Otherwise, it passes, because ATPG can justify the port to its off state and allow the cell to capture.

To minimize the number of violations reported, the tool traces back constant or X paths to a common source point, which can be a primary input or internal pin. The violating sources are

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reported in order by number of affected sequential elements with the first violation having the most, and so on. This is an example of the reported message:

report_drc_rule C25

```
// Warning: Primary input '/clk (4)' has a constrained value of 1. As a
// result, set or reset ports cannot be turned off for 32 scan cell
// elements, one of which is '/inst/sdff1 (880)'.  (C25-1)

// Warning: Internal pin '/inst1/udp/A (567)' has a value of X and cannot
// be justified to 0. As a result, set or reset ports cannot be turned off
// for 18 scan cell elements, one of which is '/inst/sdff2 (990)'.
// (C25-2)
```

The default handling of C25 is Warning. It is performed during system mode transition from Setup to Analysis. This is the summary message:

```
// Warning: There are N clock rule C25 fails (scan cell elements'
// set/reset port cannot be justified to off state).
```

To debug C25 violations, use the "set_gate_report -constrain_value on" command to look at the constrained values on set and reset ports of affected scan cells.

C26

Category: Clock

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Both the clock and data ports of non-scan and shadow cells must not be controlled by the effect cone of a single clock or of multiple clocks in a synchronous clock group. This can result in a race condition that produces inaccurate simulation results.

The application performs this check by determining the effect cone of compatible clock pins. The bounds for the effect clock cone are the combinational fanout cone of sequential elements launched by the compatible clock pins when constrained. The rule violation occurs on non-scan cells and shadow cells when both the clock port and data port of a cell can change their values at the same clock edge (either the clock leading or trailing edge) due to the effect cone transitions.

The default handling for this rule violation is Warning. The following is an example of the reported message:

report_drc_rule C26

```
//  Warning: Non-scan flop '/udff21/' (128) has potential race on data and
//  clock ports. (C26-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The summary message is as follows:

```
//  Warning: There were <N> C26 violations (sequential element with race
//   on data and clock ports).
```

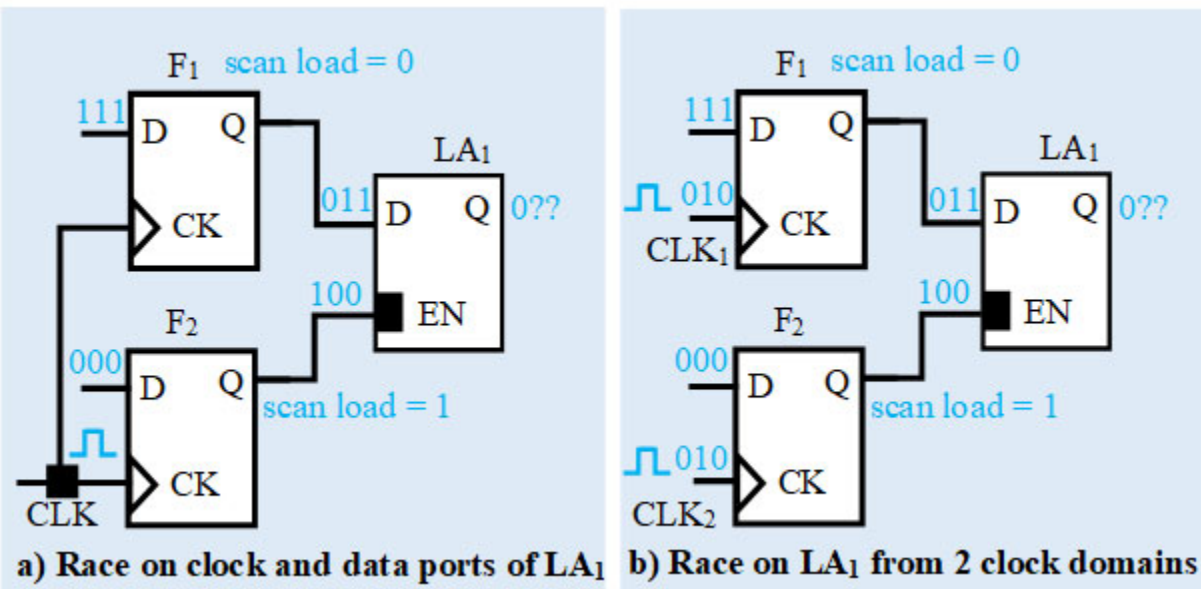
You can use one of the following commands to mask the capture values when such a race condition occurs:

```
set_simulation_options -C26_mask_races cross_domain_only
# enabled by default

set_simulation_options -C26_mask_races on
```

The “cross_domain_only” value is the default and masks the capture value of state elements only for cross-domain race conditions. The “on” value masks the capture value for all race conditions.

Example

The following figure shows two similar race conditions:



In (a), Latch LA_1 is controlled by the CLK clock domain flop F_2. The data port of LA_1 is driven by another flop, F_1, in the same clock domain. This results in a race condition that can cause simulation mismatches.

Similarly, in (b), if CLK_1 and CLK_2 are defined as a synchronous clock group, CLK_1 and CLK_2 are pulsed together and another race condition results.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

C27

Category: Clock

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

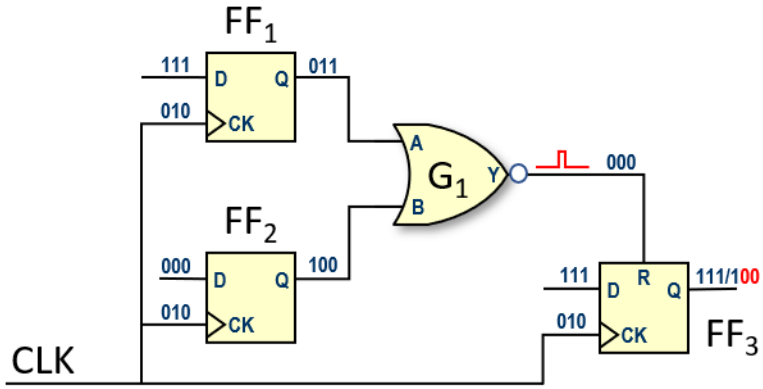
report_drc_rules: Supported

Analyzes logic signals that drive the set or reset ports of sequential elements. The analysis detects potential glitches when multiple signal transitions occur on the same clock edge. It prints a violation when a potential glitch can cause the set or reset of sequential elements.

Figure 10-24 shows a simple example of logic driving a reset port that can potentially glitch to reset the sequential element and cause simulation mismatches.

In the figure, FF_1 and FF_2 are in the same clock domain and transition on the same clock pulse. They drive through the NOR gate G_1 to the reset port of FF_3. This creates a potential glitch (010) that causes a reset. The capture value by FF_3 depends on whether the glitch activates reset, which can cause simulation mismatches.

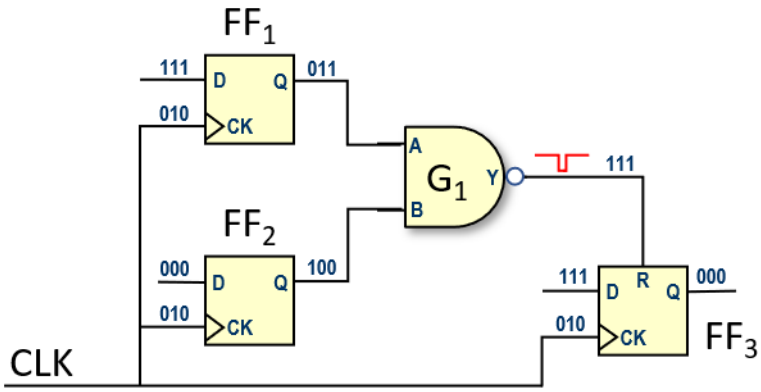
Figure 10-24. C27 Design Rule Violation



Potential glitches (101) to the non-active state of the reset port are not an issue. They do not activate a reset and do not violate the C27 design rule check. The capture value of FF_3 is not impacted, as shown in Figure 10-25.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-25. No Violation of the C27 Design Rule



The check runs during the system mode transition from Setup to Analysis mode for all pattern contexts when sequential elements have set or reset ports. The following is an example of a C27 rule violation message:

```
// Note: Internal pin '/inst1/udp/A (567)' may generate a glitch that
propagates to reset ports of 4 sequential elements causing simulation
mismatches, one of which is '/inst/sdff1 (880)'. (C27-1).
```

The following is an example summary message for C27 violations:

```
// Note: There were 2 C27 violations (potential glitches on set/reset).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Scan Cell Data Rules (D Rules)

The DFT tools check scan cells to ensure they are able to properly control and observe their data. You may select the handling of these scan cell data rules to be error, warning, note, or ignore.

The following subsections describe these rules.

Tip

In Tessent Visualizer, you can press Ctrl + R to execute a report_gates command on selected gates. For more information, see "How to Report Gate Data."

D1 3267
D2 3271
D3 3272
D4 3273
D5 3274
D6 3275
D7 3276
D8 3277
D12 3278

D1

Category: Data

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Checks for the possible disturbance of data values loaded or captured into scan cells.

During the application of the test procedures, if other circuitry disturbs the data values loaded or captured into scan cells, those cells cannot be controlled or observed. The tool performs this check using the simulated values of each time period of the test procedures. A violation occurs if any clock input (including set and reset lines) of any scan cell allows data capture at an inappropriate time, in which case the control value loaded or the capture value unloaded from the scan cell cannot be trusted. Common sources of D1 violations are undefined clocks, problems in the test procedure file, or possibly design errors.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Effect on Testability

Failure to satisfy this rule can result in simulation mismatches when you verify the test patterns in a timing-based simulator. If the violations occur during the shift procedure, you may see related simulation mismatches in the serial pattern testbench only, not the parallel testbench.

If the number of D1 violations is small compared to the number of scan cells in the design, you may choose to handle this rule violation as a warning by issuing the set_drc_handling D1 Warning command prior to DRC. When "set_drc_handling d1" warning is in effect for DRC, the tool automatically applies an XX cell constraint to each D1 failing scan cell. This prevents the mismatches, but may reduce test coverage slightly. See the add_cell_constraints command description for information about the XX cell constraint.

Note

If "set_drc_handling d1" note is in effect for DRC, the tool does not apply any XX cell constraints for D1 violations, which ensures test coverage is not affected. However, you should use "note" handling only for cell architectures you know do not produce mismatches.

If the number of D1 violations is large, you should isolate and resolve the source of the failure(s) to avoid a major impact on test coverage.

How to Debug D1 Violations

The occurrence message is:

```
// N (G) disturbed during time T of P procedure. (D1-1)
```

N is the instance name of the scan cell memory element, G is the gate ID number, T is the time period, and P is the group test procedure name.

The summary message is:

```
// There were N occurrences of scan cell disturbs. (D1)
```

N is the number of occurrences of rules violation D1.

Use the report_drc_rules command to obtain additional information about the violations. For example, to view the occurrence messages for all D1s, use:

report_drc_rules d1

The occurrence messages list gate names and gate IDs you can copy and paste into commands during later debugging. You can report on a specific occurrence by issuing the command with "D1-" and the occurrence number. For example:

report_drc_rules d1-1

```
//  Error: /dataout/reg_q_0_ (373) disturbed during time 0 of grp1
//  load_unload procedure. (D1-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When a D1 error occurs, you can access the simulated values at the gate where the error occurred by issuing the command, "set_gate_report Error_pattern", then using the report_gates command to report the gate whose ID number is displayed in the error message. In the resultant display, you can identify the clock input not held at its off state. By tracing back from this input, you can usually identify how to correct the problem.

You can debug a specific occurrence of a D1 violation using Tessent Visualizer or by issuing commands from the tool's command line. Examples of both methods follow.

How to Debug with Tessent Visualizer

To view the location of a D1 violation using Tessent Visualizer, use the following command steps:

1. open_visualizer
2. set_gate_level Primitive

Choose the **DRC Browser** tab from the Tessent Visualizer menu and find the DRC(s) of interest in the table. Double clicking the listed violation, or right clicking and selecting **Visualize DRC** from the popup menu analyzes the violation. Tessent Visualizer displays the scan cell affected by that occurrence.

Tip

Alternatively, you can issue the analyze_drc_violation command with the *rule_id-occurrence#* argument at the tool's command line. For example, to display the first occurrence of a D1 violation:

```
analyze_drc_violation d1-1
```

Once the offending scan cell is displayed, trace back from the cell's clock input to the primary input that drives the clock. Ensure the clock's off state, which you defined using the add_clocks command, is correct.

How to Debug from the Tool Command Line

To view the location of a D1 violation from the command line, use the following example command steps:

1. set_gate_report Error_pattern
2. set_gate_level Primitive
3. report_drc_rules D1-*occurrence#*
4. report_gates *offending_gate's_id#*

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following transcript excerpt shows an example of the use of this command sequence:

set_gate_report error_pattern
set_gate_level primitive
report_drc_rules d1-1

```
//  ERROR: /dataot/reg_q_0_ (373) disturbed during time 0 of grp1
//         load_unload procedure. (D1-1)
```

report_gates 373

```
//  /dataot/reg_q_0_ (373)  DFF
//      "S"    I  (0)  43-
//      R      I  (0)  157-
//      CLK    I  (1)  313-/dataot/ix104/Y
//      "D0"   I  (X)  255-
//      "OUT"  O  (X)  127-  128-
```

You would then trace back from the CLK input to try to determine why it is in an active state, which disturbed the cell's value.

In some of the more complex cases, the problem can be a clock disturb early in the capture cycle. In these cases, the error report does not provide enough detail to isolate the failure. To get more information, you can use the following command steps:

1. set_gate_report Drc_pattern State_stability

2. set_system_mode analysis

3. report_gates *offending_gate's_id#*

The following shows an example of the use of this command sequence:

set_gate_report drc_pattern state_stability

```
//  Creating schematic for 3 instances (0 were compacted).
```

set_system_mode analysis

```
//  ---------------------------------------------------------------------
//  Begin scan chain identification process, memory elements = 56.
//  ---------------------------------------------------------------------
//  Reading group test procedure file fast.testproc.
//  WARNING: Pin tclk is used for both write control and pulse clock
//  Simulating load/unload procedure in grp1 test procedure file.
//  Chain = chain1 successfully traced with scan_cells = 12.
//  Chain = chain4 successfully traced with scan_cells = 12.
//  Chain = chain3 successfully traced with scan_cells = 12.
//  Chain = chain2 successfully traced with scan_cells = 12.
//  48 scan cells have been identified in 4 scan chains.
//  Longest scan chain has 12 scan cells.
//  WARNING: 8 edge-triggered clock ports set to stable high. (D7)
//  ERROR: /dataot/reg_q_0_ (373) disturbed during time 0 of grp1
//         load_unload procedure. (D1-1)
//  ERROR: Rules checking unsuccessful, cannot exit SETUP mode.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_gates 373

```
//  /dataot/reg_q_0_ (373)  DFF
//                  (ts)(ld)(shift)(cap)(stbl)
//      "S"     I   ( 0)( 0)(000~0)(000)(   0)   43-
//      R       I   ( 0)( 0)(000~0)(XXX)(   X)   157-
//      CLK     I   ( 1)( 1)(101~1)(XXX)(   X)   313-  /dataot/ix104/Y
//      "D0"    I   ( X)( X)(XXX~X)(XXX)(   X)   255-
//      "OUT"   O   ( X)( X)(XXX~X)(XXX)(   X)   127-  128-
```

The item to notice is the clock values during the capture cycle. In this case the value of the clock is (XXX). This indicates the clock is not at its off state at the beginning or end of the capture cycle. The correct values should be either:

- (0X0) — For a leading edge device with an off state of 0 for the clock, or
- (1X1) — For a trailing edge device with an off state of 1 for the clock

Possible Resolutions

If your debugging effort shows an offending clock's off state is incorrect, use the add_clocks command to redefine the clock with the correct off state. To change the defined off state, first delete the old definition with the delete_clocks command, then reissue add_clocks. To see a clock's current defined off state, use the report_clocks command.

If the clocks are working correctly, examine the test procedure file. By tracing through the design, you may find an incorrect constraint or force value that can be fixed in the test procedure file.

Note

If you make any changes in the test procedure file, execute the read_procfile command with the correct filename before re-checking the design rule checks.

D2

Category: Data

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

During system data capture, if the scan path from a MASTER or SLAVE element to its COPY is sensitized, it must be unique. The tool performs this check by comparing the inputs of the COPY with its associated memory element. A violation occurs if there are multiple paths from the associated memory element (MASTER or SLAVE) to the COPY and these paths can be sensitized at the same time.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The application checks for the following conditions:

- The value on the data line of the COPY element must be able to be propagated back to its associated memory element along the scan path when constrained pins are set.
- The scan path, when sensitized from the associated memory element to the COPY, must be unique.
- The COPY and its associated memory element may have only a single clock port.
- For a COPY and its associated memory element, all non-tied clock, set, and reset inputs must have the same or equivalent source.

Note

DRC does not consider it a D2 violation if a COPY element can capture a value through a different path other than from its associated memory element during system capture.

Failure to satisfy this rule usually reduces test coverage. You can avoid the coverage loss by using the command, “set_split_capture_cycle On”.

The occurrence message is:

```
COPY N (G) failed data capture check. (D2-1)
```

N is the instance name of the COPY memory element, and G is the gate ID number.

The summary message is:

```
N COPY scan elements failed data capture check. (D2)
```

N is the number of occurrences of rules violation D2.

D3

Category: Data

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

For all scan cells that contain a SLAVE, the master_observe procedure must propagate the data value of the MASTER memory element to the SLAVE. The application performs this check using the simulated values of each time period of the master_observe procedure to trace back from the SLAVE to the MASTER. The rule violation occurs if the master_observe procedure does not properly sensitize the path between the SLAVE and MASTER.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The default handling for this rule violation is error. You may ignore this error condition by issuing the set_drc_handling as follows:

set_drc_handling D3 Warning

When the handling is set to other than error, the tool automatically makes the necessary MASTER unobservable to prevent a potential simulation mismatch. This applies only to the MASTER of the scan cell containing a SLAVE. The scan cell without a SLAVE retains its original observability. Due to the loss of observability on some MASTERs, test coverage may be reduced.

When an error condition occurs, you can access the simulated values by setting the gate reporting to drc_pattern (with the master_observe argument and the desired time) and using the report_gates command for the gates in the SLAVE to MASTER path. This identifies the location of the blockage; by tracing back from the inputs, you can identify how to correct the problem.

The occurrence message is:

```
N (G) not successfully observed by master_observe  procedure. (D3-1)
```

N is the instance name of the MASTER memory element, and G is the gate ID number.

The summary message is:

```
N MASTERs not successfully observed by master_observe  procedure. (D3)
```

N is the number of occurrences of rules violation D3.

D4

Category: Data

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

If you define the skew_load procedure, it must propagate the data value of the preceding scan cell (or scan chain input pin) to a MASTER memory element. The application performs this check using the simulated values of each time period of the skew_load procedure to trace back from a MASTER to its preceding scan cell output or scan chain input pin. The rule violation occurs if the skew_load procedure does not properly sensitize the path.

The default handling for this rule violation is error. Failure to satisfy this rule may result in inaccurate simulation results when you use the skew load option. The skew_load procedure is optional and you can avoid rules violations by removing the procedure definition.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When an error condition occurs, you can access the simulated values by setting the gate reporting to drc_pattern (with the skew_load argument and the desired time) and using the report_gates command for the gates in the path. This identifies where the blockage occurred; by tracing back from the inputs, you can identify how to correct the problem.

The occurrence message is:

```
Skew_load procedure not successful for MASTER %N (G). (D4-1)
```

N is the instance name of the MASTER memory element and G is the gate ID number.

The summary message is:

```
Skew_load procedure not successful for N MASTERs. (D4)
```

N is the number of occurrences of rules violation D4.

D5

Category: Data

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

All memory elements (latches and flip-flops) must be scannable. The application performs this check after identifying all scan memory elements. The rule violation occurs for all memory elements not identified as part of a scan cell.

Note

The D5 check simulates the circuit as purely combinational by using a sequential depth of 0 or 1 (as if the “set_pattern_type -Sequential” command is set to 0 or 1).

When the tool identifies a non-scan memory element, the tool classifies each element into one of the following types:

- INIT-0 — If it is at 0 at the beginning of the first capture cycle.
- INIT-1— If it is at 1 at the beginning of the first capture cycle.
- INIT-X — If its state is unknown at the beginning of the first capture cycle and it may go to any state during capture.
- TIE-0 — If it is always at 0 during capture.
- TIE-1 — If it is always at 1 during capture.
- TIE-X — If it is always at an unknown state during capture.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- TLA — If it is always transparent when its clock is at its off state.

The model used (and reported in the D5 violation message) is determined from the element's value in the D5 simulation, at the end of load_unload and before entering capture. For example, if the element's value is 1 at that time, the tool models the element as a TIE-1. Note, however, that the tool simulates the TIE-0, TIE-1, and TIE-X models as INIT-0, INIT-1, and INIT-X models, respectively. This means, for example, that a non-scan TIE-0 model can be evaluated after the beginning of the first capture cycle and get a value other than 0.

Latches modeled as TIE-X gates become candidates for transparent latches, sequential transparent cells, or clocked sequential cells if you set the pattern type appropriately with the set_pattern_type command.

The default handling for this rule violation is warning. Failure to satisfy this rule results in some loss of test coverage.

The occurrence message is:

```
N (G) is a non-scan T1 converted to T2. (D5-1)
```

N is the instance name of the non-scan memory element, G is the gate ID number, T1 is the gate type (latch or flip-flop), and T2 is the gate type that models it (TIEX, TIE0, or TIE1).

The summary message is:

```
N non-scan memory elements converted to T gates. (D5)
```

N is the number of occurrences of rules violation D5, and T is the gate type that models the non-scan cell. The tool displays a summary message for each remodeled gate type.

D6

Category: Data

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

All non-scan latches must behave as transparent latches. The application performs this check for all nonscan latches that are not set to a stable binary value. The rule violation occurs if a candidate latch fails one of the following conditions:

- If the latch creates a potential feedback path, that path must be broken by scan cells or non-scan cells other than transparent latches. For more information, refer to the set_tla_loop_handling command.
- The latch must have a propagable path to an observable point.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The latch must be capable of passing a value when all defined clocks are at their off-state.
- All clock, set, and reset inputs of the latch must either be set to a determinate state when all clocks are off and pin constraints are set, or must not connect to defined clocks.
- The latch must not have more than one set/reset/clock input on when all defined clocks are at their off-state.

Failure to satisfy this rule can reduce test coverage. The default handling for this rule violation is warning. If you set the handling for this rule to ignore, the tool does not perform this check and the design's latches are not checked for transparency.

For scan insertion in the dft -scan context, if you want the tool to consider non-transparent latches as scan candidates, you must turn test logic on (with the set_test_logic command) and do one of two things: 1) set the handling of D6 to ignore, in which case the tool does not perform the transparency check and automatically considers the non-scannable latches for scan insertion; or 2) use the set_latch_handling Scan command, in which case the tool performs the check and considers non-transparent latches for scan insertion.

The occurrence message is:

```
Latch N (G) not transparent due to R. (D6-1)
```

N is the instance name of the non-scan latch, G is the gate ID number, R is the reason it cannot be transparent, and D6 is the rule ID number.

The summary message is:

```
N latches not transparent due to R. (D6)
```

N is the number of occurrences of rules violation D6, and R is the reason. The application displays a summary message for each reason.

D7

Category: Data

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

At the end of the shift procedure, the clock inputs of scan flip-flops must not be set to a one state. The application performs this check using the simulated values of the last time period of the shift procedure. The rule violation occurs if any clock input (not including set and reset lines) of any scan flip-flop (except COPY) is set to 1. A possible cause of a rules violation is an incorrect definition of the off-state of a clock.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Some design practices consider this condition acceptable.

The default handling for this rule violation is warning. Failure to satisfy this rule results in scan cells capturing data on the trailing edge of the capture clock pulse.

The occurrence message is:

```
Flip-flop N (G) has clock port set to stable high. (D7-1)
```

N is the instance name of the non-scan memory element, G is the gate ID number, and D7 is the rule ID number.

The summary message is:

```
N edge-triggered clock ports set to stable high. (D7)
```

N is the number of occurrences of rules violation D7.

D8

Category: Data

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

If a MASTER latch only propagates to a SLAVE and can only capture data when the SLAVE is inactive, a clock input of the MASTER latch must not be active when all clocks are off. The system uses the master_observe procedure to observe the values placed into the scan cell and no longer considers the SLAVE to be observable.

The application performs this check using the simulated values that result when all defined clocks are at their off-state, the constrained pins are set to their constrained values, and the initialized non-scan cells are set to their stable states.

The rule violation occurs if a clock input of a MASTER latch is not off, the MASTER latch only propagates to a SLAVE, and can only capture data when the SLAVE is inactive.

Note

Some design practices consider this condition acceptable.

When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the clock input not held off, and by tracing back from this input, you can identify how to correct the problem.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The default handling for this rule violation is error because for certain rare LSSD scan cell designs it can result in a simulation mismatch during pattern verification in a timing-based simulator. Failure to satisfy this rule results in data captured into the MASTER without the application of a capture clock.

Note

If "set_drc_handling d8 warning" is in effect for DRC, the tool automatically applies an SX cell constraint to each D8 failing master cell. This prevents the mismatches, but may reduce coverage. See the add_cell_constraints command description for information about the SX cell constraint.

If "set_drc_handling d8 note" is in effect for DRC, the tool does not apply any SX cell constraints for D8 violations, which ensures test coverage is not affected. However, you should use "note" handling only for cell architectures you know do not produce mismatches.

The occurrence message is:

```
MASTER latch N (G) allows data capture while clocks off. (D8-1)
```

N is the instance name of the non-scan memory element, and G is the gate ID number.

The summary message is:

```
Clocks at off-state allow data capture for N MASTER latches.  (D8)
```

N is the number of occurrences of rules violation D8.

D12

Contexts Supported: patterns -scan, patterns -scan_diagnosis, dft -scan

Default Handling: Warning

report_drc_rules: Supported

In the state stability analysis stage, DRC simulates a number of shift cycles to calculate the stable value (load value) of nonscan memory elements at the beginning of capture.

If some nonscan memory elements need more than one shift cycle to be initialized, and its load value is observed into a scan cell, the parallel simulation testbench may have simulation mismatches. This is because in parallel simulation testbench the tool generally only has one parallel shift cycle, leading to insufficient shift cycles to initialize those nonscan elements.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If state stability analysis takes more than one cycle to calculate nonscan load value, then the tool issues the following warning message:

```
//  Warning: DRC simulated N shift cycles to initialize some non-scan
//  memory elements to their load values. (D12-1)
//  Note: In order to avoid potential parallel simulation mismatches,
//  M post shift cycles will be automatically added to the testbench as
//  if the SIM_POST_SHIFT parameter keyword were set to M.
//  This will slow down parallel simulation. To avoid slowing down
//  parallel simulation, the number of cycles DRC simulates can be
//  restricted using "set_stability_check -max_shift_cycles 2". However,
//  initializing fewer non-scan memory elements may lead to lower test
//  coverage.
//  Note: You may use 'report_drc_rule D12 -verbose' for a list of those
//  initialized non-scan memory elements.
```

<*N*> is number of shift cycles and <*M*> is the number of shifts required.

To learn the specific cells that caused the D12 violation, use the following command:

report_drc_rules D12 -verbose

```
//  Warning: DRC simulated 3 shift cycles to initialize the following 3 non-scan
//  memory elements to their load values
//       /u_LSI_BISR_POWERONSM/LSI_MBISRC_MODE_PWR_reg_0_0/UDP1 (30733)
//       /u_LSI_BISR_POWERONSM/LSI_MBISRC_MODE_PWR_reg_1_0/UDP1 (30734)
//       /u_LSI_BISR_POWERONSM/LSI_MBISRC_MODE_PWR_reg_2_0/UDP1 (30735). (D12-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Pre-DFT Clock Rules (DFT_C Rules)

The tool uses the DFT_C rules to verify the clock specification and circuitry before DFT structures such as memory BIST, logic BIST, boundary scan, EDT, and OCC are inserted.

The tool invokes the DFT_C rules when the check_design_rules command is run if the context has been specified as either dft -rtl or dft -no_rtl, with the set_context command, and the set_dft_specification_requirements -memory_test option and/or -logic_test options are set to "on".

The following subsections describe these rules.

Simulation Contexts for Pre-DFT DRCs 3280
DFT_C1 3284
DFT_C2 3287
DFT_C3 3289
DFT_C4 3290
DFT_C5 3290
DFT_C6 3291
DFT_C7 3293
DFT_C8 3294
DFT_C9 3295
DFT_C10 3299
DFT_C11 3306
DFT_C12 3307
DFT_C13 3308

Simulation Contexts for Pre-DFT DRCs

This section describes the simulation contexts which are created by the DFT_C rules. This DRC always runs when transitioning from setup to analysis mode in the DFT context when no sub context is specified.

The created simulation contexts are used by the pre-DFT DRC rules described in the following sections. They can also be re-used by custom DRC rules you may have registered using the register_drc command.

Listed below are the simulation contexts created when transitioning from setup to analysis mode in the DFT context when no sub context is specified. You can obtain this list by using the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_simulation_context_list command. To display those values in the GUI, you need to issue the following two commands:

```
set_gate_report simulation_context
set_current_simulation_context simulation_context_name
```

The ***simulation_context_name*** string is one of the names returned by the get_simulation_context_list command. When you have DRC violations for any of the DFT_C* types, the associated simulation context is set when you issue the "analyze_drc_violation ***drc_name***" command or when you issue the "set_gate_report ***drc_name***" command.

The simulation contexts are all derived from the stable_after_setup simulation context. This context is created by simulating the test_setup procedure followed by a state stability analysis using the specified input constraints. When ICL is elaborated, the reset and control signals of the IJTAG scan interface are auto constrained off to make sure the values loaded in the "scan resource instrument" side of the network are not disturbed by the state stability analysis. Refer to the Sib wrapper section for an explanation of the difference between the "scan resource instrument" and the "scan tested instrument" sides of the network.

When ICL is elaborated, the IJTAG network is always reset even if no set_test_setup_icall or set_static_dft_signal_values commands are run. The DFT signals defined on the DataOutPorts of IJTAG nodes are forced directly on the pin of the associated instances because the pins need to be set to different values in different contexts. You typically only need to use the set_test_setup_icall or set_static_dft_signal_values commands when using "set_dft_specification_requirements -logic_test on" because you need to configure the child blocks so that sub_chain tracing can be done on them. When only using "set_dft_specification_requirements -memory_test on", you typically do not need any test_setup iCall. The network is also typically not even created at that point. If the network is already created and you do need to use a test_setup iCall, you must request the IJTAG network to be synthesized if you are running in -rtl mode. The test_setup simulator only knows how to simulate gates. You typically get E14 violations if you try to use an iCall in the test_setup and the IJTAG network is not synthesized. Use the following command to get the IJTAG network synthesized. You do not need these commands when "set_dft_specification_requirements -logic_test on" because it already causes the entire design to be synthesized.

```
set_attribute_value -name synthesize_before_analysis \
                    [get_modules -of_instances [get_ijtag_instances]]
```

In the following section, the DFT signals are mentioned and their attributes are referenced. For a description of their usage, reset_value, default_value_in_all_test and value_in_pre_scan_drc property, refer to the description of the add_dft_signals and the register_static_dft_signal_names commands.

Simulation Contexts

- dft_glb

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This simulation context is created on top of the stable_after_setup simulation context. The DFT signals with usage “global_dft_control” are forced to their “default_value_in_all_test” value. The targets of the add_dft_control_points command using a DFT signal with usage “global_dft_control” are also forced taken into account the -inverse_dft_signal_source switch. All clocks (source, generated and branch) are forced to X. This enables better isolation of the violations. For example, if the source of a branch clock is driven by a constant, it is reported once by DFT_C2, yet DFT_C1 and DFT_C6 still enable checking that the memories and the scan elements are properly driven by the branch clock.

- dft_glb_fe

  This simulation context is created on top of the dft_glb simulation context. All func_en pins of clock gaters are forced to their active value. The func_en pins are typically learned or specified in the Tessent cell library. When the clock gaters are not cells described in the Tessent cell library, they are declared using the “add_dft_clock_enables -usage func_en” command. The tool only forces the clock gaters in the combinational fan-in of memory clocks, branch clocks, or the reference of generated clocks. The other clock gaters are not forced so as not to disturb the test_setup logic that may be holding their state using those other clock gaters. This is because those clock gaters are needed to remain off to maintain the value of the test_setup. If you need to have one of those forced, use the add_dft_clock_enables command to point directly in the func_en pin you want asserted anyway.

- dft_glb_ns

  This simulation context is created on top of the dft_glb simulation context. The “nonscan_test” DFT signal is forced to its active value. The DFT signals with usage “logic_test_control” are forced to their reset value. The “scan_en” and the “edt_update” DFT signals are also forced to their inactive value. The target of the add_dft_control_points command using any of those DFT signals are forced accordingly taking into account the -inverse_dft_signal_source switch.

- dft_glb_ns_fe

  This simulation context is created on top of the dft_glb_ns simulation context. All func_en pins of clock gaters are forced to their active value. The func_en pins are typically learned or specified in the Tessent cell library. When the clock gaters are not cells described in the Tessent cell library, they are declared using the “add_dft_clock_enables -usage func_en” command. The tool only forces the clock gaters in the combinational fan-in of memory clocks, branch clocks, or the reference of generated clocks. The other clock gaters are not forced so as not to disturb the test_setup logic that may be holding their state using those other clock gaters. This is because those clock gaters are needed to remain off to maintain the value of the test_setup. If you need to have one of those forced, use the add_dft_clock_enables command to point directly in the func_en pin you want asserted.

- dft_glb_ns_fe_tck

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This simulation context is created on top of the dft_glb_ns_fe simulation context. The tck_select DFT signal is forced to its active value, justifying TCK onto the functional clock trees.

- dft_glb_ns_fe_x

  This simulation context is created on top of the dft_glb_ns simulation context. All func_en pins of clock gaters are forced to X. The func_en pins are typically learned or specified in the Tessent cell library. When the clock gaters are not cells described in the Tessent cell library, they are declared using the "add_dft_clock_enables -usage func_en" command. The tool only forces the clock gaters in the combinational fan-in of memory clocks, branch clocks, or the reference of generated clocks. The other clock gaters are not forced so as not to disturb the test_setup logic that may be holding their state using those other clock gaters. This is because those clock gaters are needed to remain off to maintain the value of the test_setup. If you need to have one of those forced, use the add_dft_clock_enables command to point directly in the func_en pin you want asserted.

- dft_glb_s

  This simulation context is created on top of the dft_glb simulation context. All DFT signals with usage "logic_test_control" are forced to their "value_during_pre_scan_drc" value. All DFT signal with usage "scan_mode" are forced to X. The DFT signal "edt_update" is forced to 0.

- dft_glb_s_se_0

  This simulation context is created on top of the dft_glb_s simulation context. The "scan_en" DFT signal is forced to 0. The target of the add_dft_control_points command using the "scan_en" DFT signals are forced accordingly taking into account the -inverse_dft_signal_source switch.

- dft_glb_s_se_0_te_1

  This simulation context is created on top of the dft_glb_s_se_0 simulation context. All test_en pins of clock gaters are forced to their active value. The test_en pins are typically learned or specified in the Tessent cell library. When the clock gaters are not cells described in the Tessent cell library, they are declared using the "add_dft_clock_enables -usage test_en" command. Only the clock gaters that fanout to at least one DFF primitive with the is_non_scannable attribute equal to false are forced. The others clock gaters are not forced so as not to disturbed test_setup logic that may be holding their state using those other clock gaters.This is because those clock gaters are needed to remain off to maintain the value of the test_setup. If you need to have one of those forced, use the add_dft_clock_enables command to point directly in the test_en pin you want asserted.

- dft_glb_s_se_0_te_x

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This simulation context is created on top of the dft_glb_s_se_0 simulation context. All test_en pins of clock gaters are forced to X. The test_en pins are typically learned or specified in the Tessent cell library. When the clock gaters are not cells described in the Tessent cell library, they are declared using the "add_dft_clock_enables -usage test_en" command. Only the clock gaters that fanout to at least one DFF primitive with the is_non_scannable attribute equal to false are forced. The others clock gaters are not forced so as not to disturbed test_setup logic that may be holding their state using those other clock gaters. This is because those clock gaters are needed to remain off to maintain the value of the test_setup. If you need to have one of those forced, use the add_dft_clock_enables command to point directly in the test_en pin you want asserted.

- dft_glb_s_se_x

  This simulation context is created on top of the dft_glb_s simulation context. The "scan_en" DFT signal is forced to X. The target of the add_dft_control_points command using the scan_en DFT signals are also forced to X.

- dft_glb_s_se_1

  This simulation context is created on top of the dft_glb_s simulation context. The "scan_en" DFT signal is forced to 1. The target of the add_dft_control_points command using the scan_en DFT signals are forced accordingly taking into account the -inverse_dft_signal_source switch.

DFT_C1

Category: Clock

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

Verifies that each memory clock port has a defined clock in its controlling fan-in. The default handling of this rule violation is error and cannot be changed.

This rule check is run only for designs that have memory instances, as reported by the report_memory_instances, or when you run the "set_dft_specification_requirements -memory_test on" command. It is not run in any other case or context. This DRC rule is performed using the dft_glb_ns_fe simulation context.

This rule check ensures that each clock pin on a memory instance is driven by a clock source that was previously defined with the add_clocks command. The check is performed with a backward trace from each memory instance clock pin. A rule violation occurs if the backward trace stops at a port or at a pin that has not been declared as a clock using the add_clocks

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

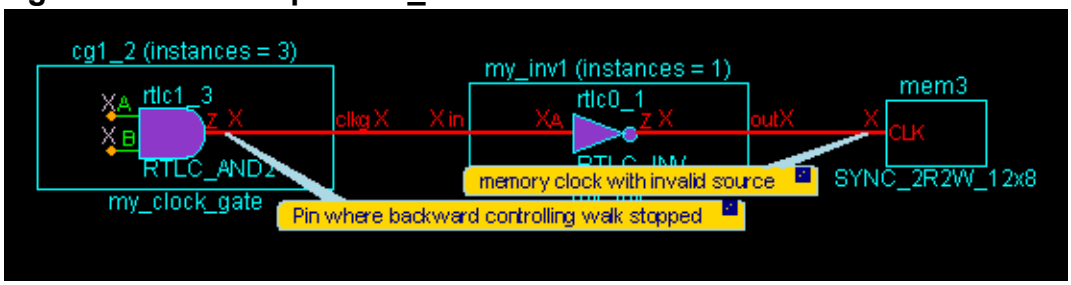
command. The tool then reports this port or pin in an error message similar to the one shown here:

```
//  Error: The memory clock pin 'blockA_I1/mem1/CLKW' (23.28) source
//  tracing stopped at pin 'pll_Inst/pll_out_2' (13.0)
//  Correct the block conditions or use the 'add_clocks -period' if this
//  is an embedded oscillator or 'add_clocks -reference' if this an active
//  PLL or clock divider. (DFT_C1-1)
```

This type of trace violation can have many different reasons. You should check your test_setup procedure and/or design constraints and ensure that they are properly sensitizing a controlling backward path from the memory instance clock pin to the design's clock source. If the blockage is caused by a clock gating cell, you can use the add_dft_clock_enables command to point to the functional enable pin. If the blockage is caused by a clock multiplexer where the select is controlled by uninitialized functional logic, you use the add_dft_control_points command.

Another typical situation that causes a backward path to be blocked is the presence of a black box instance in the path; this is shown in Figure 10-26 where a PLL instance is black boxed. In this type of case, you can use the -reference or -reference_inv option of the add_clocks command to bridge the black box tracing gap. Assuming that this black box was an internal clock oscillator, you can use the add_clocks -period option to generate an internal clock source (for example, add_clocks pll_Inst/pll_out_2 -period 10ns). If you have an associated ICL module for the PLL or embedded oscillator module, the add_clocks command is automatically inferred as described in the Description section of the add_clocks command.

Figure 10-26. Example DFT_C1 Violation in the Presence of a Blackbox



The error messages generated by those checks are listed in Table 10-3.

Table 10-3. Error Messages Generated by the Clock Tracing Checks

Type 1: The <clock> source tracing stopped at <location>. Correct the blocking conditions or use the 'add_clocks -period' if this is an embedded oscillator or add_clocks -reference' if this is an active PLL or clock divider.
Type 2: The <clock> is sourced by port <post> but its period was not defined using the 'add_clocks -period' command.
Type 3: The <clock> is sourced by a constant value of '0'.
Type 4: The<clock> has no source.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 10-3. Error Messages Generated by the Clock Tracing Checks (cont.)

Type 5: The memory <clock> source tracing stopped at <location> and this node was the target of an add_dft_clock_mux. An add_dft_clock_mux node can only be behind a clock, and cannot feed directly to the memory clocks.

If a clock becomes differential, the differential buffer must be modeled with a WIRE gate as shown below:

```
model diff_buf (out_p, out_n, in_p, in_n) (
  input (in_p, in_n) ( )
  output (out_p, out_n) (
    primitive = _inv (in_n, in_n_inv);
    primitive = _buf (in_p, in_p_buf);
    primitive = _wire (in_p_buf, in_n_inv, out_p);
    primitive = _inv (out_p, out_n);
  )
)
```

Figure 10-28 on page 3287 shows the schematic view of a differential clock path as well as the flat model representation. When a backward trace reaches a WIRE gate, a set of checks is perform to make sure the differential clock path is correct. The error messages generated by those checks are listed in Table 10-4 on page 3286.

Table 10-4. Error Messages Generated by Differential Clock Checks

Type 1: The <clock> became differential but failed a rule along the differential clock path.A side of the differential pair traced to an add_dft_clock_mux location <location> which is illegal for a differential signal.
Type 2: The <clock> became differential but only one side of the differential source traced to a defined clock source. When one element of a differential clock reaches a defined clock, the other must too.
Type 3: The <clock> became differential but failed a rule along the differential clock path. A side of the differential pair traced to a constant value.
Type 4: The <clock> became differential but it reach a source on one of the differential side that is not a declared clock nor a WIRE gate used to model a differential buffer.
Type 5: The <clock> became differential but reached two different WIRE gates. A differential buffer must be modeled with a single 2 input WIRE gate.
Type 6: The <clock> became differential but the inversion from the two inputs of one WIRE gate was different when reaching the other WIRE gate. A Differential clock buffer is modeled with a WIRE gate and there must be no relative inversion between the two input of the WIRE gate and the WIRE gate sourcing them.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 10-4. Error Messages Generated by Differential Clock Checks (cont.)

Type 7: The memories on clock domain <domain_label> are sourced by two different differential clock buffers: <differential buffer 1> <differential buffer 2> You must separate them into two clock domains using 'add_clock <clock> -branch' on at least one of of them.
Type 8: The differential clock <p_side>' and <n_side> reached a WIRE gate but the inversion from both of them was not different when reaching that WIRE gate.

Figure 10-27 shows example of a violation that is generated when an inverter is added on one side of the differential clock path and not on the other.

Figure 10-27. Example DFT_C1 Violation on Differential Clocks

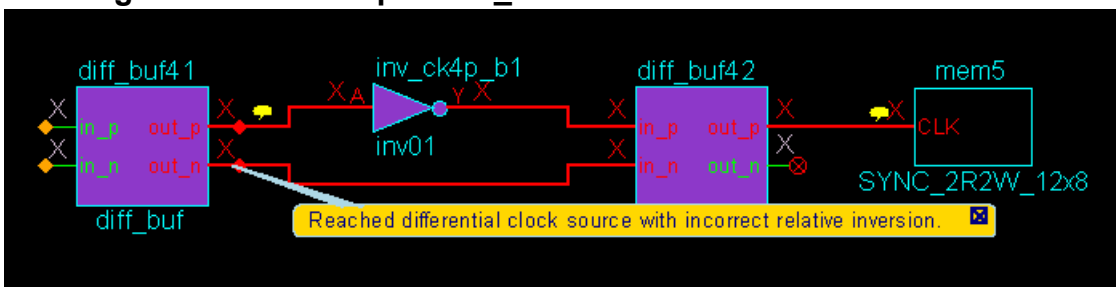
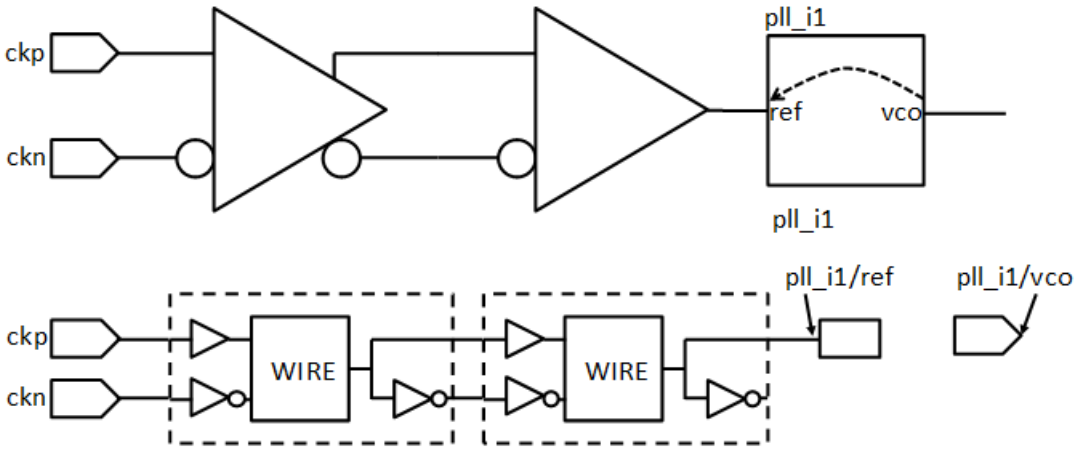


Figure 10-28. Schematic and Flat Model View of a Differential Clock Path



DFT_C2

Category: Clock

Context Supported: dft

Default Handling: Error

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_drc_rules: Supported

Verifies that each clock of type “branch” has a defined clock in its controlling fanin. The default handling of this rule violation is error and cannot be changed.

This rule check is executed only for designs that have designed branch clocks (as defined using the add_clocks -branch command) and when the -memory_test and/or -logic_test option of the set_dft_specification_requirements command is set to “on”. It is not executed in any other case or context. It is not executed in any other case or context. This DRC rule is performed using the dft_glb_fe simulation context.

This rule check ensures that each identified clock of type branch (as defined by the add_clocks -branch option) is driven by a clock source defined using the add_clocks command. The check is performed using a backward trace from each clock branch. A rule violation occurs if the backward trace ends at a port or instance pin that has not been declared as a clock using the add_clocks command as shown in Figure 10-27. The tool reports the port or pin where the traced stopped as shown in the error message below:

```
//  Error: The clock branch 'clk_B_buf_2/Y' (1706.0) is sourced by
//  'clk_B' (2.0) but the source was not declared as a clock using the
//  'add_clocks' command. (DFT_C2-1)
```

This type of a trace violation can have many different reasons. You should check your test_setup procedure and/or design constraints to ensure that they are properly sensitizing a controlling backward path from the memory instance clock pin to the design’s clock source. If the blockage is caused by a clock gating cell, you can use the add_dft_clock_enables command to point to its functional enable pin. If the blockage is caused by a clock multiplexer where the select is controlled by uninitialized functional logic, you can use the add_dft_control_points command. Finally, to inject an alternative source to a clock net, you use the add_dft_clock_mux command.

Another typical situation that causes a backward path to be blocked is the presence of a black box instance in the path. In this case, you can use the -reference or -reference_inv option of the add_clocks command to bridge the black box tracing gap. Assuming that the black box was an internal clock oscillator, you can use the add_clocks -period option to generate an internal clock source (for example, add_clocks pll_Inst/pll_out_2 -period 10ns). If you have an associated ICL module for the PLL or embedded oscillator modules, the add_clocks command is automatically be inferred described in the Description section of the add_clocks command.

See Figure 10-26and Figure 10-27 in the DFT_C1 section for examples showing how the violations are displayed in the schematic. The error messages generated when checking the rules are listed in Table 10-3 and Table 10-4. The bottom error message in Table 10-3 and the bottom two error messages in Table 10-4 are only relevant to DFT_C1 and do not apply to DFT_C2.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DFT_C3

Category: Clock

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

Verifies that each generated clocks has a defined clock in the controlling fanin of its reference. The default handling of this rule violation is error and cannot be changed. For a description of what constitute a generated clock, see the description of the add_clocks command.

This rule check is executed only for designs that have defined, generated clocks (as defined using the add_clocks -reference command) and when the -memory_test and/or the -logic_test options of the set_dft_specification_requirements command is set to “on”. It is not executed in any other case or context. This DRC rule is performed using the dft_glb_fe simulation context.

The check is performed using a backward trace from the specified reference and optional reference inverse pin. A rule violation occurs if the backward trace ends at a port or a pin that has not been declared as a clock using the add_clocks command. The tool reports the port or pin where the trace stopped in an error message similar to the one shown here:

```
//  Error: The reference pin 'clk_REF' (3.0) of the derived clock source
//  at 'pll_Inst/pll_out_2' (5.0) is sourced by 'clk_REF' (3.0)
//  but the source was not declared as a clock using the 'add_clocks'
//  command. (DFT_C3-1)
```

Note

When backtracing generated clocks defined with a dividing/multiplying ratio of 1, the DFT_C3 DRC forces SE to 0.

This type of a trace violation can have many different reasons. You should check your test_setup procedure and design constraints to ensure that they are properly sensitizing a controlling backward path from the memory instance clock pin to the design’s clock source. If the blockage is caused by a clock gating cell, you can use the add_dft_clock_enables command to point to its functional enable pin. If the blockage is caused by a clock multiplexer where the select is controlled by uninitialized functional logic, you can use the add_dft_control_points command. Finally, to inject an alternative source to a clock net, you use the add_dft_clock_mux command.

Another typical situation that causes a backward path to be blocked is the presence of a black box instance in the path. In this case, you can use the -reference or -reference_inv option of the add_clocks command to bridge the black box tracing gap. Assuming that the black box was an internal clock oscillator, you can use the add_clocks -period option to generate an internal clock source (for example, add_clocks pll_Inst/pll_out_2 -period 10ns). If you have an associated ICL module for the PLL or embedded oscillator modules, the add_clocks command is automatically inferred as described in the Description section of the add_clocks command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

See Figure 10-26 and Figure 10-27 in the DFT_C1 section for examples showing how the violations are displayed in the schematic. The error messages generated when checking the rules are listed in Table 10-3 and Table 10-4. The bottom error message in Table 10-3 and the bottom two error messages in Table 10-4 are only relevant to DFT_C1 and do not apply to DFT_C3.

DFT_C4

Category: Clock

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

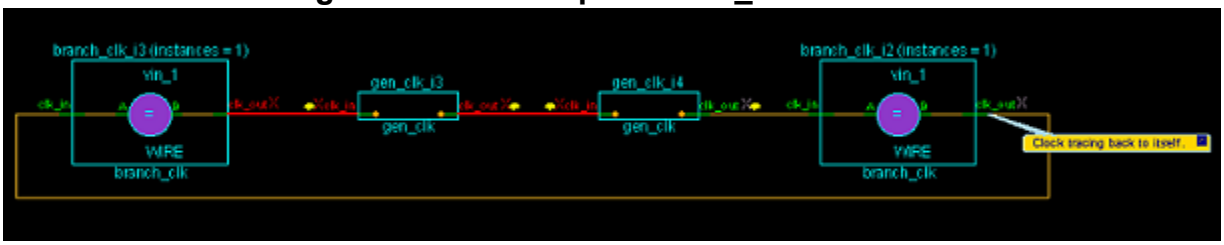
Verifies that a clock of type “generated” or “branch” is not itself in the fanin of its reference source. A generated clock is a clock that was defined with the add_clocks -reference option specified. The default handling of this rule violation is error and cannot be changed.

This DRC rule is performed using the dft_glb_ns_fe simulation context. This rule check is executed only for designs that have declared branch clocks (as defined using the add_clocks -branch command) or generated clocks (as defined using the add_clocks -reference command) and when the -memory_test option of the set_dft_specification_requirements command is set to “on”. It is not executed in any other case or context.

This rule check ensures that the reference of each generated clock source traces to a source that is not affected by itself. A rule violation occurs if a loop is detected in the fanin of the referenced pin. The tool reports the violation as shown below. The schematic that is shown you issue the analyze_drc_violation command highlights the circular path:

```
//  Error: The reference pin 'clk_pll_buf/Y' (1683.0) of the derived clock
//  source at 'pll_Inst/pll_out_3' (5.0)
//  is driven by itself via the clock source. (DFT_C4-1)
```

Figure 10-29. Example of DFT_C4 Violation



DFT_C5

Category: Clock

Context Supported: dft

Default Handling: Error

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_drc_rules: Supported

Verifies that each clock source has a specified period set using the "add_clocks -period" command. The default handling of this rule violation is error and cannot be changed.

This rule check is executed only for designs that have memory instances or declared generated or branch clocks and when the -memory_test option of the set_dft_specification_requirements is set to "on". It is not executed in any other case or context. This rule does not use a simulation context. It simply verifies the source of the clock extracted by DFT_C1, DFT_C2, and DFT_C3 have a specified period so that the SDC clocks with a known period can be created later in the flow when running the extract_sdc command. The clock periods are also used by the create_patterns_specification when creating the signoff and manufacturing patterns specification wrappers.

This rule check ensures that each clock has a well-defined period.

A rule violation occurs if a clock source is found that does not have a well-defined period, as shown in this example:

```
add_clocks clk_REF
# add_clocks clk_REF -period 100ns
add_clocks clk_A -pulse_always -period 10ns
add_clocks clk_B -period 15ns

add_clocks pll_Inst/pll_out_1 -reference_pin clk_REF
add_clocks pll_Inst/pll_out_2 -reference_pin clk_REF

check_design_rules
```

Notice that the clock port "clk_REF" does not have a defined period. However, it is used as the reference of the generated clocks pll_Inst/pll_out_1 and pll_Inst/pll_out_2. The tool reports a violation for the above example as shown here:

```
//  Error: The clock 'clk_REF' must be defined with a period.
//         The clock is used by memory_bist. (DFT_C5-1)
```

DFT_C6

Category: Clock

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

Verifies that each scannable flip-flop clock port has a defined clock in its controlling fan-in. The default handling of this rule violation is error and cannot be changed.

Note

Scan segments inside tcd_scan defined modules are also taken into account, if enabled. See "Clock/used_for_capture : on | off | auto;" on page 4800 for details.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This rule check is run when the set_dft_specification_requirements -logic_test command option is set to "on". It is not run in any other case or context. This DRC rule is performed using the dft_glb_s_se_0_te_1 simulation context.

This rule check ensures that a previously-defined clock source (using the add_clocks command) drives each clock pin on a scannable flip-flop. When the "set_dft_specification_requirements -memory_test on" command is specified together with the "-logic_test on" switch, the clock input of memories that are considered for Memory BIST are also checked with the DFT_C6 rule. This is because the Memory BIST circuity insertion adds scannable flip-flops, requiring controllable clocks during logic test. If you follow the standard flow described in the Tessent Shell Workflows chapter of the *Tessent Shell User's Manual*, you do not specify the "-memory_test on" switch when doing the second DFT insertion pass. However, the Memory BIST hardware was already inserted during the first insertion pass, making the scannable flip-flops generated by Memory BIST already visible to the DFT_C6 rule.

The check is performed with a backward trace from each scannable flip-flop instance. The clock gaters, which become open because of the dft_glb_s_se_0_te_1 simulation context, are verified for correct operation in DFT_C8. A rule violation occurs if the backward trace stops at a port or a pin that has not been declared as a clock using the add_clocks command. The tool then reports this port or pin in an error message. For example:

```
// Error: The clock pin 'blockA_I1/mem1/CLKW' (23.28) source
// tracing stopped at pin 'pll_Inst/pll_out_2' (13.0)
// Correct the block conditions or use the 'add_clocks -period' if this
// is an embedded oscillator or 'add_clocks -reference' if this an active
// PLL or clock divider. (DFT_C6-1)
```

Note

A DFT_C6 violation root cause may affect more than one design node (flop, pin, net) at a time. The tool reports one DFT_C6 violation per root cause, but the error message only shows one node.

This type of trace violation can have many different reasons. Check your test_setup procedure or design constraints, and ensure that they are correctly sensitizing a controlling backward path from the scannable flip-flop instance clock pin to the clock source of the design.

- If the blockage is caused by a clock gating cell, use the "add_dft_clock_enables -usage test_en" command to point to the test_en enable pin.
- If the blockage is caused by a clock multiplexer, where the select is controlled by uninitialized functional logic, use the add_dft_control_points command.

Another typical situation that causes a backward path to be blocked is a black box instance in the path where a PLL instance is black-boxed. In this type of case, use the -reference and -reference_inv option of the add_clocks command to bridge the black box tracing gap. Assuming that this black box was an internal clock oscillator, you can use the add_clocks -period option to generate an internal clock source (for example, "add_clocks pll_Inst/pll_out_2

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

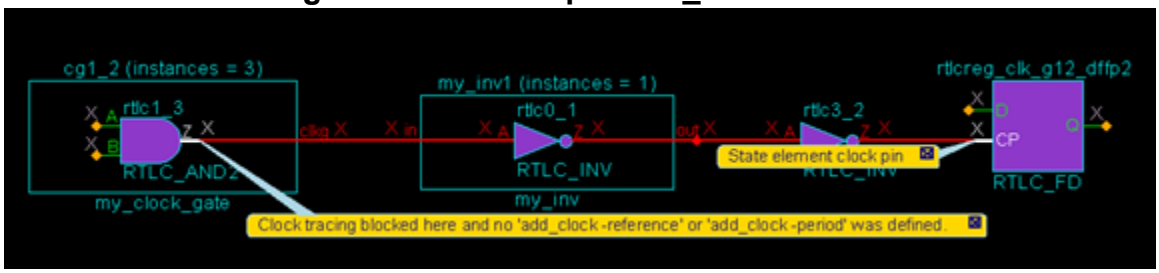
-period 10ns”). If you have an associated ICL module for the PLL or embedded oscillator module, the tool automatically infers the add_clocks command, as discussed in the description section of the add_clocks command. See the description of the ProcedureStep wrapper for an example of such an ICL module.

See Table 10-5 for the error messages generated by those checks. Figure 10-30 is an example of the schematic shown when the analyze_drc_violation command is run on a Type 1 violation.

Table 10-5. Error Messages Generated by the Clock Tracing Checks

Type 1: The <clock> source tracing stopped at <location>. Correct the blocking conditions or use the “add_clocks -period” command if this is an embedded oscillator or “add_clocks -reference” if this is an active PLL or clock divider.
Type 2: The <clock> is sourced by port <post> but its period was not defined using the “add_clocks -period” command.
Type 3: The <clock> is sourced by a constant value of “0”.
Type 4: The<clock> has no source.
Type 5: The scannable flip-flop <clock> source tracing stopped at <location> and this node was the target of an add_dft_clock_mux. An add_dft_clock_mux node can only be behind a clock, and cannot feed directly to the memory clocks.

Figure 10-30. Example DFT_C6 Violation



For differential paths, the same handling of different clocks shown in the DFT_C1 description applies to DFT_C6 except that the start points are clock pins of scannable flip-flops instead of memory clocks.

DFT_C7

Category: Clock

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

Verifies that the clock gating cells in the fanin of memory clocks are properly implemented.

This rule check is executed only for designs that have memory instances or declared generated or branch clocks and when the -memory_test option of the set_dft_specification_requirements

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is set to “on”. It is not executed in any other case or context. This DRC rule is performed using the dft_glb_s simulation context. Any gate that was marked between a clock and a memory clock pin, a branch clock or the reference of a generated clock which becomes blocking when the simulation context is switch from dft_glb_ns_fe to dft_glb_ns is checked in this DRC rule.

Several checks are performed to verify that the clock gating meets the specific checks to make sure that they remain compatible to logictest. You can use the add_dft_control_points command on the func_en pin to disable the clock gating completely. If you fail one of those violations, however, those clock gaters remain untestable during logictest. Only clock gaters disabled by a specified or inferred add_dft_clock_enables signal are verified by this DRC. See report_dft_clock_enables for more information. The error messages generated by those checks are listed in Table 10-6. Clock gaters feeding branch clocks or the reference of generated clocks are disabled with the “all_test” DFT signal because they need to be disabled in both the non-scan and scan based test modes.

Table 10-6. Error messages Generated by DFT_C7 Checks

A unique dft_clock_enable signal of type func_en could not be found in the fanin of <node>. The multiple dft_clock_enable signals of type func_en found in the fanins are: <node1> <node2> ...
The clock gating cell <instance> is a primitive cell of type <type>. A clock gating cell must be a primitive of type AND or OR.
The clock gating cell 'clock_gate_and_bad1/and3/Y' is a primitive cell of type 'AND (OR)' but it has 3 inputs. Only 2 input primitives are allowed as clock gating cells.
The clock gating cell <node> has no latch in its enable path.
The clock gating cell <node> is an AND gate but but the latch in its enable path is active high instead of active low.
The clock gating cell <node> has a latch in its enable path but its clock input is not connected to the same driver that feeds the clock input of the clock gating cell.
The clock gating cell <node> is an OR gate but the latch in its enable path is active low instead of active high.

DFT_C8

Category: Clock

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

Verifies that the clock gating cells in the fanin of scannable flip-flops are properly implemented. The default handling of this rule violation is error but it can be downgraded to warning.

This DRC rule is performed using the dft_glb_s_se_0_te_x simulation context. Any gate which was marked between a clock and a scan element that becomes blocking when the simulation

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

context is switched from dft_glb_s_se_0_te_1 to dft_glb_s_se_0_te_x is checked in this DRC rule.

This rule check executes only for designs that have scannable flip-flops and when you have set the -logic_test option of the set_dft_specification_requirements to “on”. It is not executed in any other case or context.

The rule performs several checks to verify that the clock gating remains compatible to logictest. If one of those checks fails, you can permanently disable the clock gater using “add_dft_control_points -dft_signal ltest_en” on the test_en pin of the clock gater. However, the fanin of the func_en pin of the clock gater is untestable during logictest. Only clock gaters disabled by a specified or inferred add_dft_clock_enables signal are verified by this DRC. Table 10-7 lists the error messages generated by those checks. Those checks run only if the clock gater is not a cell with the simulation_function attribute equal to any of the following: clock_gating_and, clock_gating_or, funct_only_clock_gating_and, and funct_only_clock_gating_or. Those clock gaters already go through a formal verification when the cell library loads and do not need to be checked again. The formal check of the library parser is more flexible and allows NAND and NOR gates.

Table 10-7. Error messages Generated by DFT_C8 Checks

The clock gating cell <instance> is a primitive cell of type <type>. A clock gating cell must be a primitive of type AND or OR.
The clock gating cell 'clock_gate_and_bad1/and3/Y' is a primitive cell of type 'AND (OR)' but it has 3 inputs. Only 2 input primitives are allowed as clock gating cells.
The clock gating cell <node> has no latch in its enable path.
The clock gating cell <node> is an AND gate but the latch in its enable path is active high instead of active low.
The clock gating cell <node> has a latch in its enable path but its clock input is not connected to the same driver that feeds the clock input of the clock gating cell.
The clock gating cell <node> is an OR gate but the latch in its enable path is active low instead of active high.

DFT_C9

Category: Asynchronous Set Reset

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

Verifies that it is possible to disable the asynchronous set or reset pin of every scannable flip-flop. The default handling of this rule violation is error, but it can be downgraded to ignore.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Scan segments inside tcd_scan defined modules are also taken into account, if enabled. See "Clock/used_for_capture : on | off | auto;" on page 4800 for details.

This rule check is executed only for designs that have scannable flip-flops, and when the -logic_test option of the set_dft_specification_requirements is set to "on". It is not executed in any other case or context. This DRC rule is performed using the dft_glb_s_se_1 simulation context.

Three types of errors can be generated by the DFT_C9 DRC, and they are all shown in Table 10-8.

The situation detected by the first two errors are not auto fixed and must be corrected explicitly.

The situation detected by the last error is auto fixed when you execute "set_drc_handling DFT_C9 -auto_fix ON". This is also the tool's default behavior in this case.

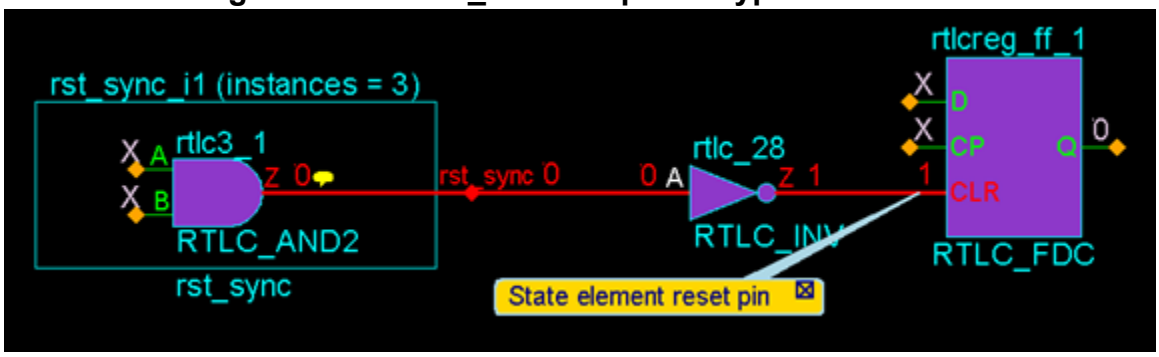
Table 10-8. Error Messages Generated by DFT_C9 Checks

Type 1: When the async_set_reset source <node_name> is asserted '0'|'1' <num1> set|reset pins on DFFs are forced off but <num2> set|reset pins on DFF are forced on. Use another 'add_dft_control_point' command to break the path to those flops or declare them as non-scannable using 'add_nonscan_instance' command. The list of set pins on DFF which are activated when the source is asserted '0'|'1' are: <pin1> < pin2> …
Type 2: The active high|low set|reset pin <gate_pin1> is sourced by <gate_pin2> and it has a constant simulated value of '1'|'0' which forces the set|reset pin active.
Type 3:The active high|low set|reset pin <gate_pin1> source tracing stopped at pin <gate_pin2>. Correct the blocking conditions or use 'add_dft_control_point -type async_set_reset' to make it controllable during logic test.

The first type of error reports nodes that source asynchronous set and reset of scannable flip-flops having different active polarity. You have two options to resolve this issue. Solution 1 is to use the "add_dft_control_points ***nodes*** -type async_set_reset" command to specify additional locations so that the active polarity of the asynchronous set and reset of all scannable flip-flop in the fanout of the specified ***nodes*** is the same. The other option is to declare the group of flip-flops listed in the error message as non-scannable using the add_nonscan_instances command. Figure 10-31 is an example of the schematic shown when using the analyze_drc_violation command on a violation of type 1. The path between the source and one of the flops that has its set or reset pin forced on is shown. The source node is simulated to the value that makes the set or reset pins on on the minority group of scannable flip-flops.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-31. DFT_C9 Example of Type 1 Violation



The second type of error reports nodes that sources asynchronous set and reset pins of scannable flip-flops which simulate to a constant value and the constant value activates the set and reset pins of scannable flip-flops. To correct this situation, you must verify your test_setup procedure or your specified add_dft_control_points specification to understand if they not responsible for those scannable flip-flops to be held in reset or set. If this is truly the functional intent, declare those flip-flops as non-scannable using the add_nonscan_instances command.

The second and third types of error do not occur unless you disable auto-fixing using the following command:

```
set_drc_handling DFT_C9 -auto_fix off
```

The tool traces the controlling fanin of the set/reset pins and infers the "add_dft_control_points -type async_set_reset" command and switch at the node where the trace stopped. The tool reports a summary message similar to the following:

```
//  Note: There were 9 'add_dft_control_points -type async_set_reset'
//  commands inferred from DRC. Use report_dft_control_points to see them.
```

Note

If the auto connection is inferred on a port of the current design and the port connects to the pad_io function of a pad cell, the DFT connection is inserted on the core side of the pad buffer. If the boundary scan cell is already inserted, it is connected on the core side of the boundary scan cell. If there is no boundary scan cell associated to the port, it connects to the from_sji_mux function when it exists, otherwise it connects to the from_pad function.

If you are running on a gate-level netlist, it is possible that you have more inferred DFT control than was actually needed. This happens when the synthesis tools perform complex restructuring and introduced X re-convergence logic in the set and reset logic paths. This is rare in RTL because the quick synthesis tool does not do any restructuring. If it happens in the RTL, then it means you have X re-convergence logic coded in your RTL. To remove this pessimism, explicitly specify the "add_dft_control_points -type async_set_reset" commands on the nodes where you want them to be, and the nodes are simulated to determine their effects. If the DRC is able to force the specified node to a value to make all the asynchronous set and reset of the

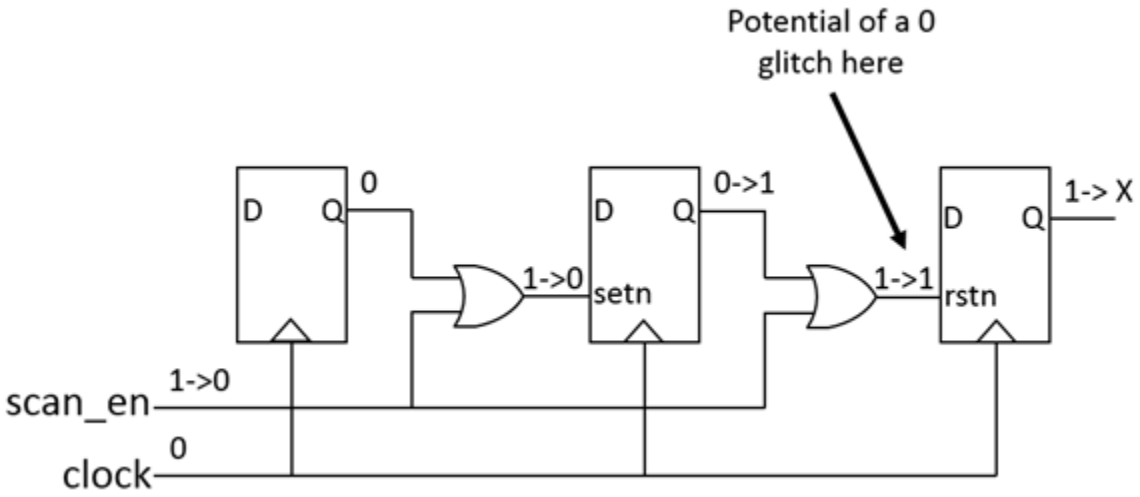
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

scannable flop-flop in its fanout inactive, then the presence of X re-convergence path does not matter.

If you have set or reset signals going to flip-flops which themselves are the source of the set or reset signals of other flip-flops, the tool performs an extra check to verify that no race condition can occur in the scan test when the scan_en signal is toggled from 1 to 0.

As shown in Figure 10-32, the race condition happens if the reset/set value of the flip-flops feeding the set or reset signals of the other flip-flops is the inactive value for the set or reset signals of the other flip-flops. When scan_en goes low, the gating of the resets is removed. If the source flop was scan loaded with the active value for the set or reset signals of the flip-flops it drives and its reset becomes active when scan_en goes low, the flip-flops in the fanout may or may not be reset.

Figure 10-32. Race Condition Check



If the source flop had time to reset to the inactive value before the scan_en released the set or reset signal of the flip-flops in its fanout, the flip-flops in the fanout may not be reset or set. Conversely, if the source flop did not have time to reset to its inactive value before the scan_en released the set/reset signal of the flip-flops in its fanout, the flip-flops in the fanout are reset or set.

ATPG most often recognizes this situation and X's out the state of the destination flip-flops, which causes lots of masking. To prevent the race, the set or reset signal of the source or destination flip-flops is disabled with the ltest_en signal instead of the scan_en signal based on

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

which one has less affected logic that can no longer be tested. When this happens, the tool reports the following note:

```
//  Note: The DFT_C9 DRC found the following node sourcing set reset pins
//        of scannable flip-flops:
//          rst
//        It is disabled with 'ltest_en' because it is involved in a
//        cascaded set/reset configuration that generates glitches during
//        scan test.
```

Avoid Disabling Async Reset or Set Port With scan_en

When you are performing DFT for a block that is not a wrapped core, you may want to avoid inserting a gate to disable the reset or set signal when it is a port because disabling the set or reset port is handled at the next level up. To do that, constrain the set or reset port off using the add_input_constraints command before calling the check_design_rules command. After you are done with the insertion, go back to Setup mode and delete the input constraints before calling extract_icl. This is because those constraints are not needed for ICL extraction and would end up in the ICL module if you keep them and they are verified to still be constrained when running extract_icl at the next level up.

Remember that for wrapper cores, you need to insert a gate to disable the reset or set signal even when it is a port because the tool infers a dedicated wrapper cell on those ports to isolate the reset source during internal test mode. If you are using the boundary scan chain to isolate the top level ATPG modes, you should gate the reset/set signals with scan_en.

DFT_C10

Category: Clock

Context Supported: dft

Default Handling: Error/Warning

report_drc_rules: Supported

Verifies that an injection node for an OCC or TCK MUX can properly control a clock source. The default handling of the rule violation is a warning for the logic test DFT specification, and an error for the -dft etchecker sub context.

This rule check is executed with severity warning if you issue the set_dft_specification_requirements command with the -logic_test switch as follows:

set_dft_specification_requirements -logic_test on

It is executed with a severity of error in the context dft -etchecker.

The validated DFT injection node can then be used using the “get_clock_option -dft_inject_node” command and switch to determine where to inject the TCK multiplexer—see the -tck_injection switch description to the set_dft_specification_requirements command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can upgrade or downgrade this DRC to error, warning, note, or ignore using the set_drc_handling command.

You specify the DFT injection node when you create a new clock using the "add_clocks -dft_inject_node" command and switch. To specify the injection node for an existing clock, use the "set_clock_options -dft_inject_node" command and switch. This DRC rule is performed using the dft_glb_s_se_0_te_1 simulation context.

The DFT_C10 runs to ensure that the injected DFT node meets the following requirements:

- Only one injection node should be defined for a clock source. The clock source can be an ordinary clock source (a primary pin) or a derived clock source (for example, a PLL output).
- An injection node does fanout to scan elements and memories as the corresponding clock source.
- The DFT inject node is in the controlling fanout of a defined clock source. In case of differential inputs, the DFT inject node is the node or is in the controlling fanout of the node where the clock became single ended.
- All scan elements sourced by clock source are in the fanout of the DFT inject node.
- There are no references of generated clock in the fanout of the DFT inject node.
- All branch clocks sourced by a clock source are in the fanout of the DFT inject node.
- The injection node is not specified where a constant setting is propagated or the node is open.
- An injection node has to be specified in case of a clock source that is driving a generated clock source reference and scan elements/memories.
- The path between the clock source and the injection node must be transparent to ensure that the above rules can be fully checked.

Table 10-9. Error and Warning Messages Generated by DFT_C10

Warning/Error: The backward tracing from the specified -dft_inject_node '<injection_node>' did not reach the clock '<defined_clock_source>'. The tracing stopped at node '<tracing_end_node>'. [<stop_location_reason_addemdum>] (DFT_C10-x).
Warning/Error: The backward tracing from the specified -dft_inject_node '<injection_node>' which is where the clock '<defined_clock_source>' becomes single ended at node <differential_output_node>. The tracing stopped at node '<tracing_end_node>'. (DFT_C10-x).
Warning/Error: Clock '<clock_source>' fanouts to ['<number>' scannable element[s]] [and] ['<number>' BIST'ed memory cell inputs[s]] which [is][are] not in the fanout of the specified -dft_inject_node '<injection_node>', which are 'failing_element_1' 'failing_element_2' … 'failing_element_10' … . (DFT_C10-x)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 10-9. Error and Warning Messages Generated by DFT_C10 (cont.)

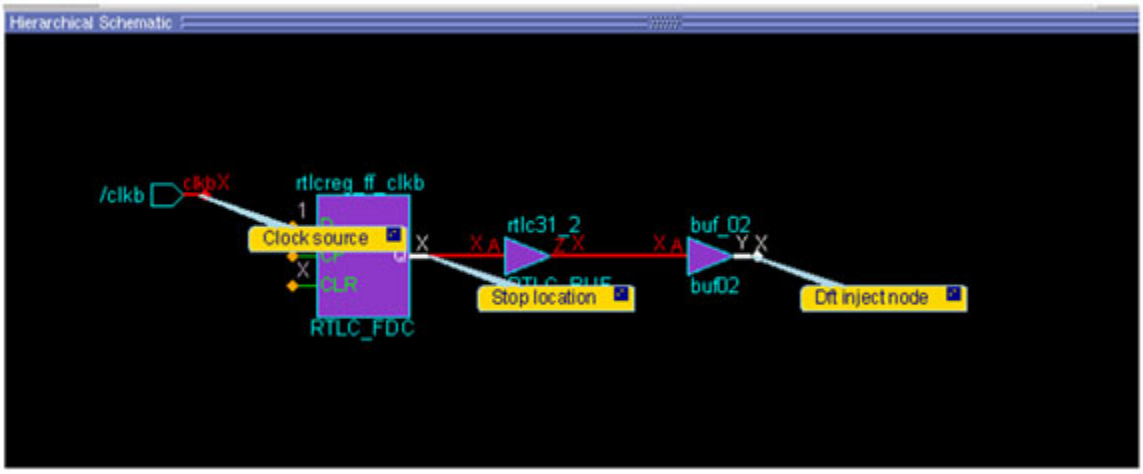
Warning/Error: The clock source '<clock_source>' is the source of the reference of the generated clock '<gen_clock_source>' and require the specification of a correct injection node. Use 'set_clock_option clk -dft_inject_node' to specify a node that matches the following criteria: 1) The DFT inject node is in the controlling fanout of '<clock_source>'. 2) All scan elements and all memories sourced by '<clock_source>' are in the fanout of the DFT inject node. 3) There are no references of generated clock in the fanout of the DFT inject node. 4) There are no branch clocks in the fanout of the DFT inject node. (DFT_C10-x)
Warning/Error: The clock source '<clock_source>' is the source of the reference of the generated clock '<gen_clock_source' and require the specification of a correct injection node. Use 'set_clock_option <clock_source> -dft_inject_node' to specify a node that matches the following criteria: 1) The DFT inject node is in the controlling fanout of '<clock_source>'. 2) All scan elements and memories sourced by '<clock_source>' are in the fanout of the DFT inject node. (DFT_C10-x).
Warning/Error: The branch clock '<clock_branch_node>' is sourced by the clock '<clock>' without going through its" "dft_inject_node '<injection_node>'." (DFT_C10-x)
Warning/Error: The specified -dft_inject_node '<injection_node>' for clock '<clock_source>' has no source. (DFT_C10-x).
Error/Warning: The clock '<clock_source>' is the source of the branch clock <clock_branch_node>' and require the specification of a correct injection node. Use 'set_clock_option <clock_source> -dft_inject_node' to specify a node that matches the following criteria: 1) The DFT inject node is in the controlling fanout of '<clock_source>'. 2) All scan elements and all memories sourced by '<clock_source>' are in the fanout of the DFT inject node. 3) There are no references of generated clock in the fanout of the DFT inject node. 4) There are no branch clocks in the fanout of the DFT inject node. (DFT_C10-x)
Error/Warning: The dft injection node '<injection_node>' is defined for the clock source '<clock_source_1>' and for the clock source '<clock_source_2>'. Only differential clock pairs can share the same injection node. (DFT_C10-x)
Warning/Error: The specified -dft_inject_node '<injection_node>' for clock '<clock_source>' has a constant value of <0/1>. (DFT_C10-x)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

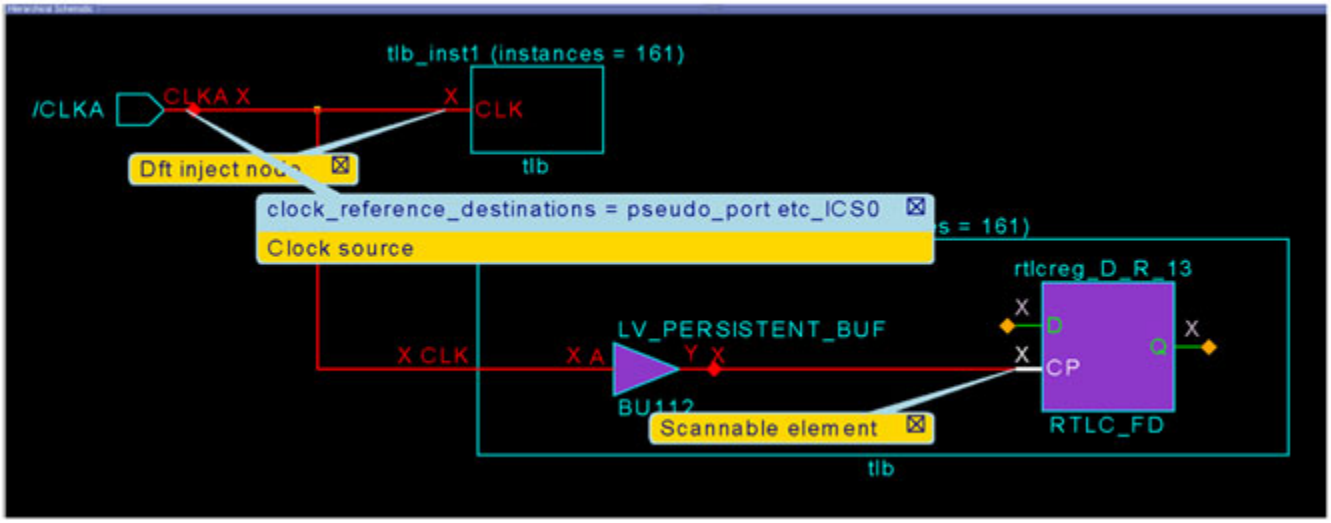
```
//  Error: The backward tracing from the specified -dft_inject_node
//         'buf_02/Y' did not reach the clock 'clkb'. The tracing stopped
//         at node 'rtlcreg_ff_clkb/Q'. (DFT_C10-6)
```



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

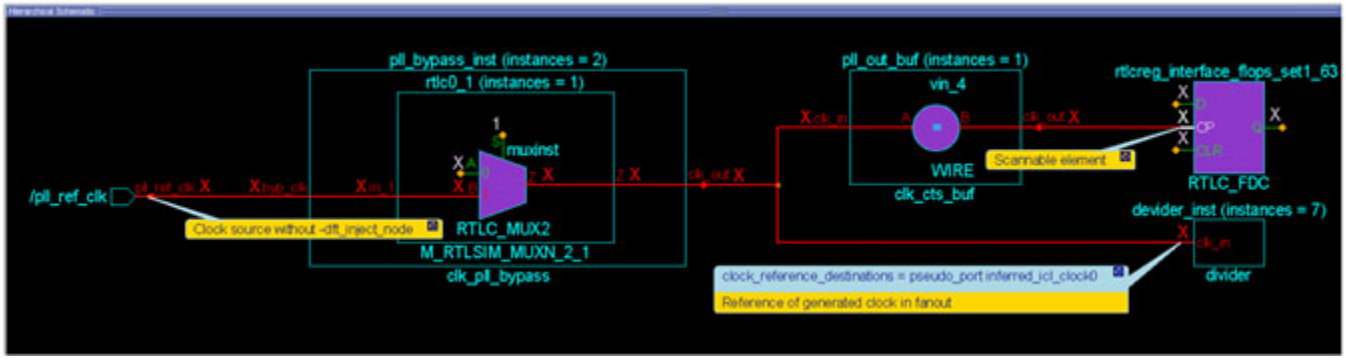
```
//  Error: The clock 'CLKA' fanouts to 156 scannable elements which are
//  not the fanout of the specified -dft_inject_node 'tlb_inst1/CLK',
//  which are:
//           'tlb_inst2/rtlcreg_D_R_13/CP'
//           'tlb_inst2/rtlcreg_D_R_12/CP'
//           'tlb_inst2/rtlcreg_D_R_11/CP'
//           'tlb_inst2/rtlcreg_D_R_10/CP'
//           'tlb_inst2/rtlcreg_D_R_9/CP'
//           'tlb_inst2/rtlcreg_D_R_8/CP'
//           'tlb_inst2/rtlcreg_D_R_7/CP'
//           'tlb_inst2/rtlcreg_D_R_6/CP'
//           'tlb_inst2/rtlcreg_D_R_5/CP'
//           'tlb_inst2/rtlcreg_D_R_4/CP'
//           ... (DFT_C10-1)
```



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

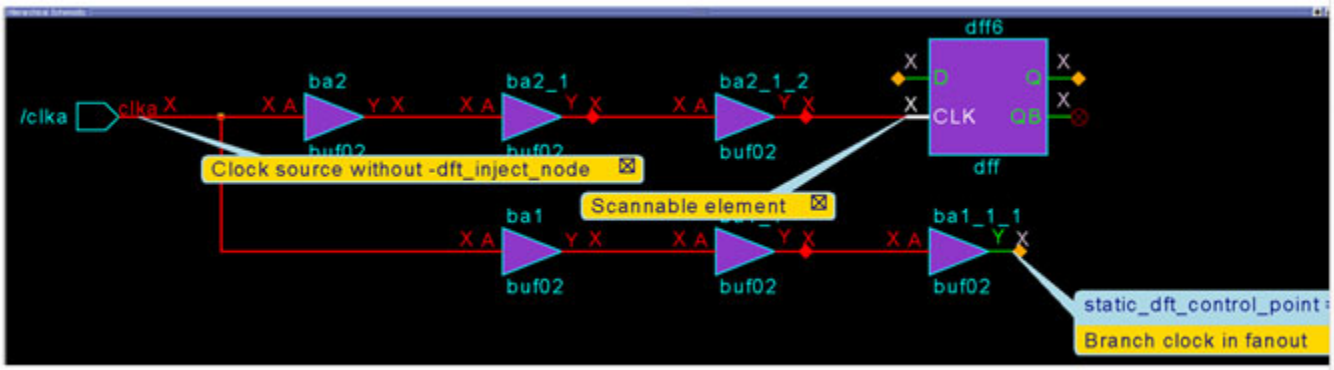
Example 3

```
//  Error: The clock 'pll_ref_clk' is the source of the reference of the
//         generated clock 'divider_inst/clk_out2' and requires the
//         specification of a correct injection node.
//         Use 'set_clock_option pll_ref_clk -dft_inject_node' to specify
//         a node that matches the following criteria:
//          1) The DFT inject node is in the controlling fanout of
//             'pll_ref_clk'.
//          2) All scan elements and all memories sourced by
//             'pll_ref_clk' are in the fanout of the DFT inject node.
//          3) There are no references of generated clock in the fanout of
//             the DFT inject node.
//          4) There are no branch clocks in the fanout of the DFT inject
//             node. (DFT_C10-1)
```



Example 4

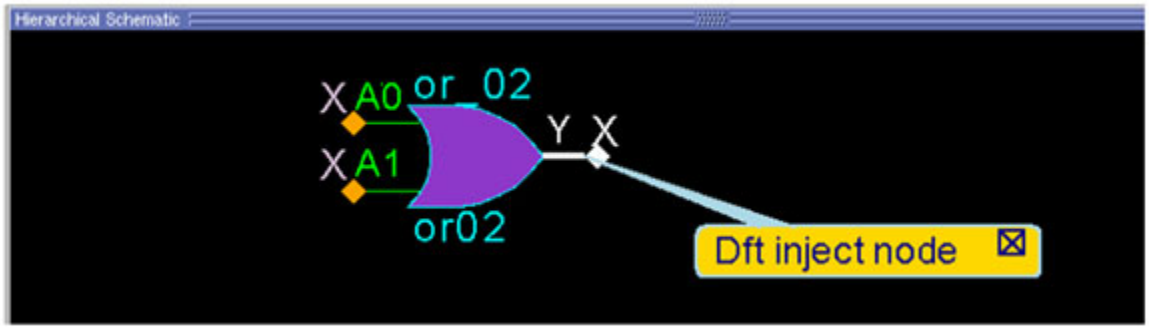
```
//  Error: The clock source 'clka' is the source of the branch clock
//         'ba1_1_1/Y' and require the specification of a correct
//         injection node.
//         Use 'set_clock_option clka -dft_inject_node' to specify a node
//         that matches the following criteria:
//          1) The DFT inject node is in the controlling fanout of 'clka'.
//          2) All scan elements and all memories sourced by 'clka' are in
//             the fanout of the DFT inject node.
//          3) There are no references of generated clock in the fanout of
//             the DFT inject node.
//          4) There are no branch clocks in the fanout of the DFT inject
//             node. (DFT_C10-1)
```



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

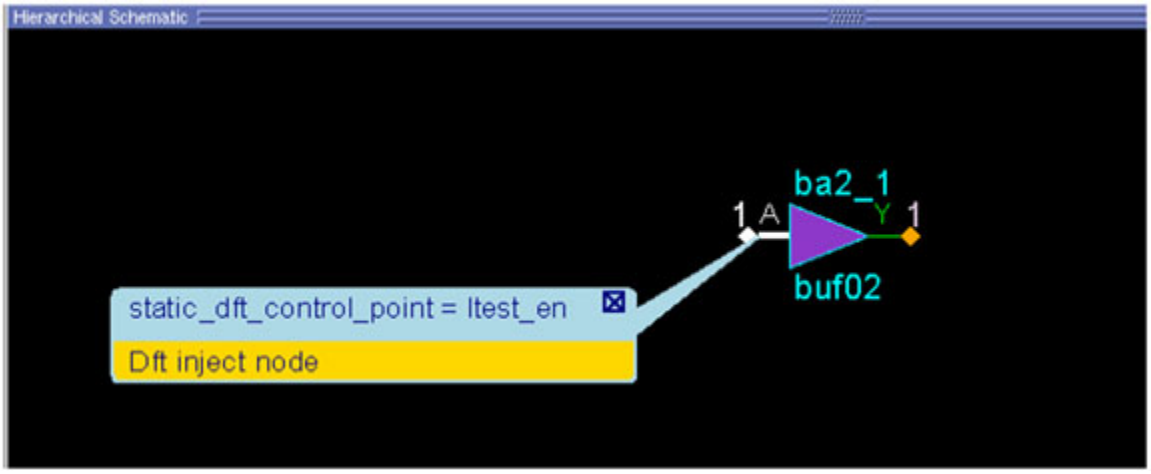
Example 5

```
//  Error: The backward tracing from the specified -dft_inject_node
//         'or_02/Y' did not reach the clock 'clk1'. The tracing stopped
//         at node 'or2/Y'. (DFT_C10-2)
```



Example 6

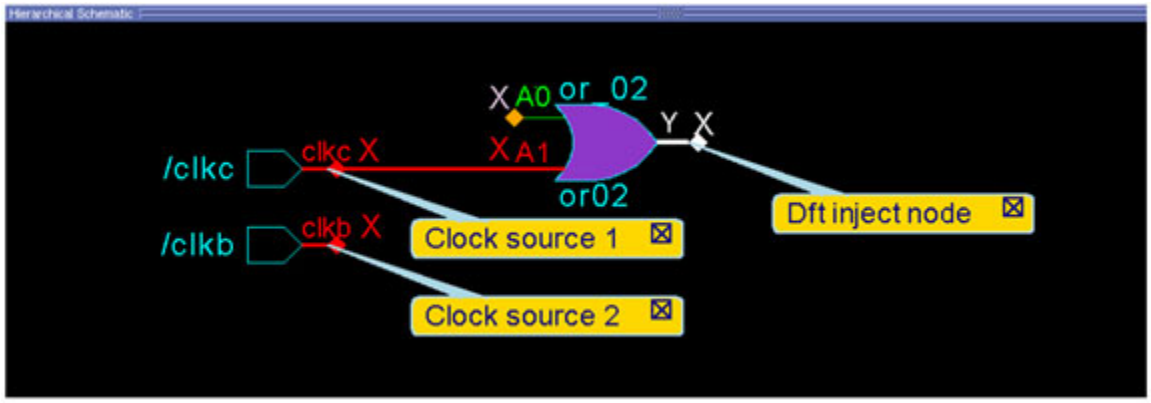
```
//  Error: The specified -dft_inject_node 'ba2_1/A' for clock 'clka' has a
//         constant value of 1. (DFT_C10-1)
```



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 7

```
//  Error: The dft injection node 'or_02/Y' is defined for the clock
//         source 'clkc' and for the clock source 'clkb'.
//         Only differential clock pairs can share the same injection
//         node.(DFT_C10-1)
```



DFT_C11

Category: Clock

Context Supported: dft

Default Handling: Warning

report_drc_rules: Supported

Identifies and verifies the clock location and polarity for TCK mode injection. The default handling of this rule violation is warning, but it can be downgraded to note.

This DRC rule identifies clocks requiring the injection of TCK to run TCK mode patterns on all memories and child blocks. It also identifies existing TCK injection that is controlled by the tck_select DFT signal. The polarity of the existing TCK injection is verified.

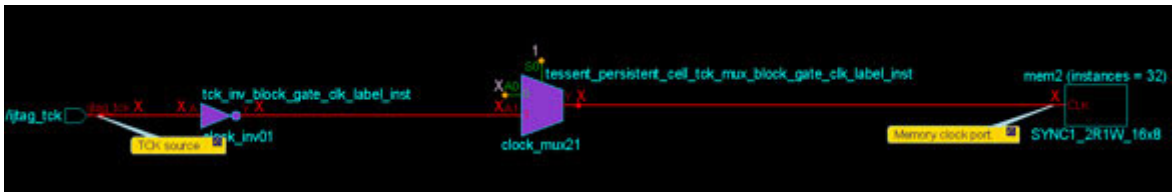
This rule check is executed only when the -tck_injection option of the set_dft_specification_requirements command is specified to “on” or “internal_sources_only”. This DRC rule is performed using the dft_glb_ns_fe_tck simulation context.

The warnings reported by this DRC rule are shown in Table 10-10 and illustrated in the figure below.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 10-10. Warning Messages Generated by DFT_C11 Checks

The TCK clock that reaches the clock pin <clock_location> for TCK mode is inverted to the source. During TCK mode tests, memory BIST instruments require to be driven by a TCK that has a consistent polarity. The BAP, controllers and memory interfaces must receive an identical TCK clock. The polarity of TCK reaching the memory itself does not have to be consistent with the memory BIST instruments.



This warning reports existing TCK injection that could be incompatible with Tessent Shell's TCK mode. It is required that the TCK clock injected on the functional clock path, driving the memory BIST controllers and memory interfaces, be of the same polarity as the IJTAG network's TCK clock driving the memory BIST BAP. Since the polarity of the network's TCK clock is not yet known, this warning is issued.

DFT_C12

Category: Clock

Context Supported: dft

Default Handling: Error

report_dft_rules: Supported

Verifies that it is possible to edit the identified TCK injection points and verifies the direction for internal pin cases. The default handling of this rule violation is error, but it can be downgraded to note.

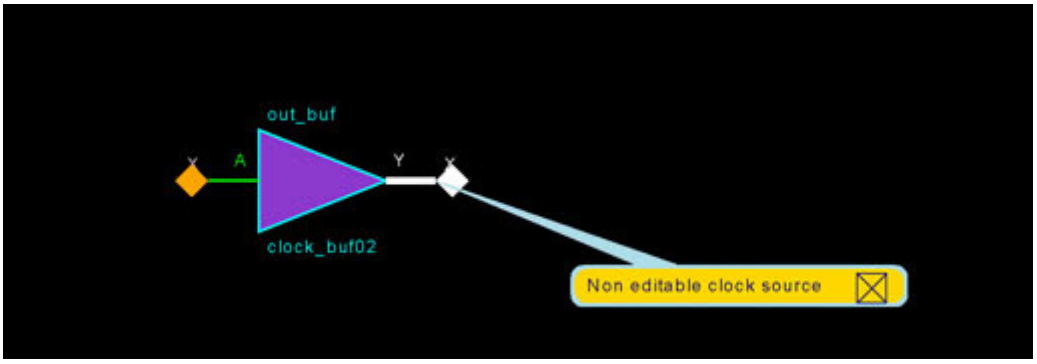
This rule check is executed only when the -tck_injection option of the set_dft_specification_requirements command is specified to "on" or "internal_sources_only". This DRC rule is performed on the data collected during DFT_C11.

Two types of errors are generated by this DRC rule:

The clock source <node_name> would require TCK injection but is not editable. Use the "-dft_inject_node" option of the add_clocks command to specify an editable location.
The clock source <clock_name> is pointing to input pin <node_name>. This is currently not supported for TCK injection. Define clocks on output pins or use the "-dft_inject_node" option of the add_clocks command to specify an output pin.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The first rule violation, illustrated in the figure below, occurs if a clock source requires the insertion of a clock mux and possibly an inverter for TCK injection, but that location is not editable.



The solution for this violation is to specify the first editable location following the clock source using the "-dft_inject_node" option of the add_clocks command. The other option is to downgrade the handling of this DRC, which removes the violating locations reported from the TCKInjectionPoints wrapper during create_dft_specification. The impact of this choice is that for TCK mode patterns, you must inject TCK on the functional clock primary input port, bypassing any PLL or divider on the clock path.

The second error type reports a current limitation with TCK injection. TCK can only be injected on clocks defined on internal output pins. The solution is to define the clock on an output pin or to define an output pin using the "-dft_inject_node" option of the add_clocks command. If you choose to downgrade the handling of this DRC, the problematic <node_name> locations are omitted from the TCKInjectionPoints wrapper during create_dft_specification. The impact of this choice is that for TCK mode patterns, you must inject TCK on the functional clock primary input port, bypassing any PLL or divider on the clock path.

DFT_C13

Category: Clock

Context Supported: dft

Default Handling: Error

report_dft_rules: Supported

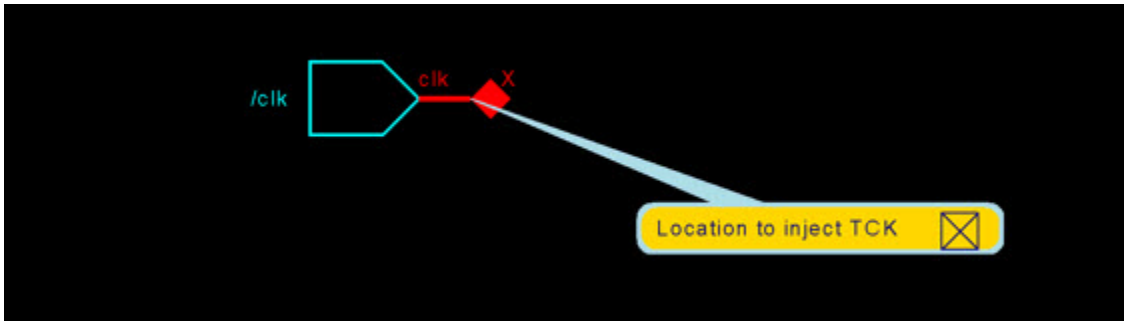
This rule verifies the existence or the scheduled addition of the required tck_select DFT signal.

This DRC rule is performed on the data collected during the DFT_C12 rule check. DFT_C13 is executed only when the -tck_injection option of the set_dft_specification_requirements command is specified to "on" or "internal_sources_only", and when clock locations have been identified for TCK injection.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following error is generated if the tck_select DFT signal is not found, and illustrated in the figure below:

```
Clocks have been identified for TCK injection but the DFT signal
tck_select does not exist. It is required to control the select of
the inserted muxes. You must add the tck_select DFT signal to continue
with the -tck_injection option of set_dft_specification_requirements.
```



The tck_select DFT signal is required to drive the select pin of the TCK injection muxes that are inserted for memory BIST TCK mode. The solution for a DFT_C13 violation is to use the add_dft_signals command to add the tck_select signal. If you choose to downgrade the handling of this DRC to ignore, no TCK injection point is identified and exported to the DftSpecification TCKInjectionPoints wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Pre-DFT Scan Rules (DFT_S Rules)

The tool uses the DFT_S rules to verify that the sub-module DFT requirements specified in *.tcd_scan* files are met.

DFT_S1 3310

DFT_S1

Category: Scan

Context Supported: dft

Default Handling: Error

report_drc_rules: Supported

Verifies that TestEn pins of sub-module instances with a *.tcd_scan* file description are set to their specified active polarity after test setup.

The default handling of this rule violation is Error. You can change handling to Warning, Note, or Ignore.

Note

Ignore this violation when a test_setup procedure supplies the value that is not yet applied during pre-DFT DRCs. If your module instances are black-boxed and you do not try to trace the chain, you do not need to drive the ltest_en pin high to run scan insertion. You do need to drive ltest_en high before tracing the chains in ATPG.

The *.tcd_scan* format describes the scan chains and the scan control signals of a scan-inserted IP module child instance. (See "Scan" on page 4794 for more details.) This enables various DRC checks from the child's boundary pins, regardless of whether the child module is loaded or black-boxed in its parent RTL design. Based on this format, pre-dft DRCs can validate the controllability of the instance clock, reset, and constant control pins.

The Scan wrapper's TestEn wrapper specifies instance pins that require a constant active value during logictest mode. This DRC rule verifies that the dft_glb_s_se_0_te_1 simulation context propagates a constant value to match the specified "active_polarity" of the pin. (See "Simulation Contexts for Pre-DFT DRCs" on page 3280 for details on the dft_glb_s_se_0_te_1 simulation context.)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For example, given the following TestEn pins description:

```
Core(child2) {
  Scan {
    is_hard_module : 1;
    pre_scan_drc_on_boundary_only : 1;
    traceable : 0;
    TestEn (TE1,TE2,TE_array) {
      active_polarity : 6'b110011;
    }
    Clock(clk) {
    }
    ...
```

A DFT_S1 violation on the TE1 pin occurs if the propagated simulation value is either logic ‘X’ or ‘0,’ and displays the following message:

```
//  Error: The 'TestEn' pin 'TE1' on tcd_scan instance 'child2_inst1' has
value 'X' instead of '1'
//     You can fix that error by either fixing your design,
//     changing your test setup, or specifying this command:
//     'add_dft_control_points [get_pins child2_inst1/TE1] -
dft_signal_source_name ltest_en' (DFT_S1-1)
```

As the example above demonstrates, a single TestEn wrapper can describe many TestEn pins. You can use Tessent Visualizer to display the location of the faulty pin.

Figure 10-33. Tessent Visualizer Showing a DFT_S1 Error Location

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Extra Rules (E Rules)

Extra rules identify potential design requirement problems.

The following subsections describe the extra rules.

Tip

In Tessent Visualizer, you can press Ctrl + R to execute a report_gates command on selected gates. For more information, see "How to Report Gate Data."

E1 3312
E2 3313
E3 3313
E4 3314
E5 3317
E6 3319
E7 3320
E8 3321
E9 3322
E10 3322
E11 3327
E12 3328
E13 3328
E14 3329
E15 3333
E16 3333

E1

Category: Extra

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

All scan cells must be LSSD scan cells that contain a master and a slave latch. They may also contain shadow latches. This rule is meant to enforce a strict LSSD architecture within a design. The application performs this check by inspecting the memory elements of all scan cells. The rule violation occurs when any memory element (that is not a latch or a scan cell) does not contain a SLAVE.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The default handling for this rule violation is ignore. Failure to satisfy this rule has no effect. The occurrence message is:

```
MASTER N (G) is not an LSSD latch. (E1-1)
```

N is the instance name of the MASTER gate, and G is the gate ID number.

The summary message is:

```
N scan cells are not LSSD. (E1)
```

N is the number of occurrences of rules violation E1.

E2

Category: Extra

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

There must be no data inversion between adjacent scan cells, the scan chain input pin (SCI) and its adjacent scan cell, and the scan chain output pin (SCO) and its adjacent scan cell. The application performs this check by inspecting the inversion data for all scan cells. The rule violation occurs when any adjacent scan cells (including SCI and SCO) have an inversion difference.

The default handling for this rule violation is ignore. Failure to satisfy this rule has no effect. The occurrence message is:

```
Scan chain has inversion between N1 (G1) and N2 (G2). (E2-1)
```

N1 is the instance name of one MASTER (or SCI) gate, G1 is its gate ID number, N2 is the instance name of the other MASTER (or SCO) gate, and G2 is its gate ID number.

The summary message is:

```
There were N scan chain inversions. (E2)
```

N is the number of occurrences of rules violation E2.

E3

Category: Extra

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

There must be no inversion between MASTER and SLAVE for any scan cell. The application performs this check by inspecting the inversion data for the memory elements of all scan cells. The rule violation occurs when the MASTER is inverted relative to its SLAVE.

The default handling for this rule violation is ignore. Failure to satisfy this rule has no effect. The occurrence message is:

```
SLAVE N (G) is inverted relative to MASTER. (E3-1)
```

N is the instance name of the SLAVE, G is its gate ID number, and E3 is the rule ID number.

The summary message is:

```
There were N SLAVEs inverted relative to MASTER. (E3)
```

N is the number of occurrences of rules violation E3.

E4

Category: Extra

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Checks for conflicting values driving the same net during evaluation of the test procedures.

Tri-state drivers must not have conflicting values when driving the same net during the application of the test procedures. The tool performs this check using the simulated values of each time period of all test procedures except the capture procedure. A rule violation occurs if any bus gate is at an X state and two or more of its inputs are not at Z.

The default handling for this rule violation is warning.

Effect on Testability

Failure to satisfy this rule does not affect coverage, but results in the risk of bus contention.

How to Debug E4 Violations

The occurrence message is:

```
Bus contention on N (G) occured at time T of P procedure. (E4-1)
```

N is the instance/net name of the bus gate, G is the gate ID number, T is the time period, and P is the procedure name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The summary message is:

```
There were N occurences of bus contention in test procedures. (E4)
```

N is the number of times an E4 violation occurred.

You can access the simulated values at the gate where the error occurred by issuing the set_gate_report command with the Drc_pattern argument, then using the report_gates command to report the gate whose ID number is displayed in the error message. If this is the only DRC violation, or if you have changed the default handling to error, you can also view simulated values of the gate of interest by issuing set_gate_report with the Error_pattern argument prior to using report_gates. Gate reporting with the proper simulation data helps you identify the conflicting inputs; by tracing back from these inputs, you can identify how to correct the problem.

If you see an "A" appended to the "E4" in the occurrence message (E4-1-**A** for example), it indicates the occurrence is due to the tool aborting the E4 check before it was completed. As these are "possible" but unproven E4s, the tool may be able to screen out some of them if you raise the abort limit. Refer to "Possible Resolutions" for more information.

How to Debug with Tessent Visualizer

To view the location of an E4 DRC violation using Tessent Visualizer, use the following command steps:

1. set_gate_level Primitive
2. open_visualizer

Choose **Open > DRC Browser** from the Tessent Visualizer menu and double-click the desired E4 DRC violation in the table (or right-click and select **Visualize DRC** from the popup window). The tool automatically changes the gate reporting to "drc_pattern load_unload" and displays the bus gate affected by that violation in the **Flat Schematic** tab.

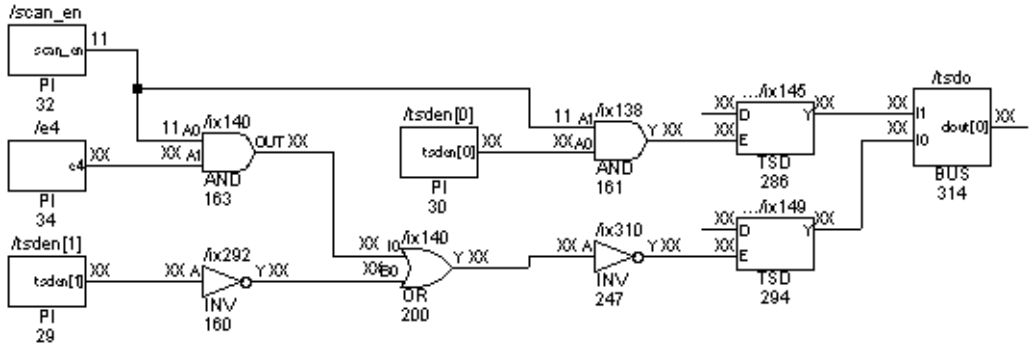
Tip

Alternatively, you can issue the analyze_drc_violation command with the *rule_id-occurrence#* argument at the tool's command line. For example, to display the first occurrence of an E4 violation: analyze_drc_violation e4-1

Once the offending bus gate is displayed, trace backward from the bus gate. Figure 10-34 shows an example Tessent Visualizer display of such a trace, where the starting point of the trace is the offending gate, /tsdo.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-34. Example E4 Violation Trace in Flat Schematic Window of Tessent Visualizer



How to Debug from the Tool Command Line

To view the location of the first occurrence of an E4 violation, use the following steps:

1. set_system_mode analysis
2. report_drc_rules E4-*occurrence#*
3. set_gate_level Primitive
4. set_gate_report Drc_pattern Load_unload
5. report_gates *gate_id*

The following shows an example of the use of this command sequence starting at step 2:

report_drc_rules e4-1

```
// Warning: Bus contention on /tsdo/dout[0] (314) occurred at time 0 of
//   load_unload procedure. (E4-1)
```

set_gate_level primitive
set_gate_report drc_pattern load_unload
report_gates 314

```
//  /tsdo (314)  BUS
//      "I0"   I  (XX)  294-/tsdo/ix149/Y
//      "I1"   I  (XX)  286-/tsdo/ix145/Y
//      "OUT"  O  (XX)  396-/dout[0]  330-
```

The Xs on the two inputs to the bus gate in this example are the source of the problem. A reasonable way to proceed is to choose one input and trace back through the gates driving it until you find the source of the Xs. For example, you might first look at where the "I0" input comes from:

report_gates 294

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  /tsdo/ix149 (294)  TSD
//      E      I  (XX)  247-/ix310/Y
//      "D"    I  (XX)  221-
//      Y      O  (XX)  314-
```

Notice the Xs on the enable (E) input. This input is of particular interest because an X here results in an X on the TSD's output regardless of the value on its D input. So tracing back from the enable would be a logical next step:

report_gates -endpoints -backward 247

```
//  ----------------------------------------------------------------------
// Begin backward trace for gate /ix310 (247).
//  ----------------------------------------------------------------------
//  /tsden[1] (29)  PI
//      tsden[1]  O  (XX)  160-/ix292/A
//  /scan_en (32)  PI
//      scan_en  O  (11)  163-/ix140/A0  161-/ix138/A1  162-/senmux/ix9/A1
//  /e4 (34)  PI
//      e4     O  (XX)  163-/ix140/A1
//  Number gates in trace = 3.
```

If you could get a 0 on the net connected to this TSD's enable, it would remove the Xs on its output.

Possible Resolutions

Cell constraints and ATPG constraints (dynamic or static) have no impact on the E4 rule. Pin constraints can have an impact because the tool may add extra cycles at the end of the test_setup procedure if the procedure does not preserve the constrained value. The constrained value can be overridden by another force value in the test procedure.

It is common during test_setup, when a design is being initialized, to have valid but momentary bus contention before the design reaches its initialized state. The E4 rule often detects these momentary occurrences of bus contention and as a result, the tool may report many E4 violations for the test_setup procedure. If you know these momentary occurrences of bus contention are not a problem and do not wish to be reminded of them, you can disable the E4 checks for the test_setup procedure by issuing the command "set_drc_handling -Skip_procedure Test_setup".

Increasing the abort limit with the set_abort_limit command can help screen out false E4 violations by enabling the tool to perform more extensive contention checking. This works for E4A violations only.

E5

Category: Extra

Contexts Supported: dft (with no sub context), dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Default Handling: Note in patterns -scan context and dft (with no sub context) context. Ignore otherwise.

report_drc_rules: Supported

When the application places constrained states on constrained pins and binary states on PIs and scan cells, X states must not propagate to an observable point. Failure to satisfy this rule results in the risk of X states propagating to an observable point. This is a serious condition for BIST circuits, may reduce the effective compression obtainable with EDT, and has no effect on uncompressed ATPG. For compressed designs, the number of E5 violations should be minimized in order to reduce the impact on compression.

Note

In dft (with no sub context) context, this rule is checked when get_dft_specification_requirements -logic_test returns “on”.

Note

Tessent TestKompress uses X-masking to handle the X states, but they typically increase the number of patterns generated, lowering the effective compression.

The application performs this check on gates that can create an X state with their inputs at binary values. It does not consider gates that do not have a path to an observable point, or that have all paths blocked by tied or constrained circuitry. The tool checks for the following conditions:

- A violation on a wired gate (WIRE) occurs if the tool can place different values on its inputs and the net resolution is set to wire.
- A violation on a BUS gate occurs if more than one of the BUS-connected tri-state drivers or switches turn on simultaneously, or all drivers turn off simultaneously and the Z state behaves as an X.
- A violation on a tri-state driver gate (TSD) or a switch gate (SW) occurs if it does not connect to a BUS gate. You can turn off the enable line, and the Z state behaves as an X.
- A violation on a TIE-X gate occurs if the gate is locally sensitizable up to the point where it has multiple fanouts (or observable points).
- A violation on a transparent latch (TLA) occurs if a single clock line is not set to its on-state, or the set and reset lines are not off.
- A violation on a ROM or RAM gate occurs if you can set a read line to off, the read_off value is X, and an output is sensitizable when the read line is off.
- A RAM/ROM violation also occurs if any memory element is uninitialized and an output is sensitizable to an observation point.
- PI pins can propagate Z to PO when Z handling is X (default). To prevent this, NO_Z constraints on PIs are considered by E5. Additionally, you cannot delete NO_Z

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

constraints added in setup mode in a non-setup mode to preserve the E5 check assumption.

The default handling for this rule violation in patterns -scan context and dft (with no sub context) context is note; in other contexts, it is ignored. When a violation occurs, you can access the tied/constrained simulated values by setting the gate reporting to “constrain_value” and using the report_gates command for the gate ID number displayed in the DRC message. The tool needs to be in a non-setup system mode. By tracing back and forward from this gate, you can identify why the violation occurred.

The occurrence message is:

```
T gate N (G) may have an observable X-state. (E5-1)
```

T is the gate type, N is the instance/net name of the gate, and G is the gate ID number.

The summary message is:

```
N gates may have an observable X-state. (E5)
```

N is the number of occurrences of rules violation E5.

E6

Category: Extra

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

When the tool places constrained states on constrained pins, the inputs of a gate must not have sensitizable connectivity to more than one memory element of a scan cell.

The tools consider a scan cell to have multiple memory elements if the cell contains, for example, a shadow element or copy element. Refer to Remote Element and Shadow Element in the *Tessent Scan and ATPG User’s Manual* for detailed information about these memory elements and their use in scan cells.

The application performs this check by tracing the forward cones of influence of all scan cell memory elements through unconstrained and untied circuitry. The rule violation occurs when any gate is in the cone of influence of more than one memory element of a single scan cell.

The default handling for this rule violation is ignore. Failure to satisfy this rule may result in some loss of test coverage, but most faults should be detectable using a skewed load test procedure.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The occurrence message is:

```
Multiple memory elements of scan cell P (C) are connected to N (G). (E6-1)
```

P is the position number of the scan cell, C is the chain name, N is the instance name of the gate, and G is its gate ID number.

The summary message is:

```
There were N scan cells with multiple memory element  connectivity to a
gate.  (E6)
```

N is the number of occurrences of rules violation E6.

E7

Category: Extra

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

External bidirectional drivers must be at the high impedance (Z) state during the application of the test procedures. You can use this rule to ensure that no bus contention can occur at bidirectional pins independent of the force values on the bidirectional pins. The application performs this check using the simulated values of each time period of all test procedures (except test_setup). The rule violation occurs if any bidirectional tri-state driver is not at a Z state. Using the -Mode option with the set_drc_handling command and the report_drc_rules command, you can check the value being forced on the bidirectional pins.

The default handling for this rule violation is ignore. If rule E4 (which ensures bus-mutual exclusivity) passes, a violation of rule E7 has no effect. Failure to satisfy this rule (normally) has no effect if there is no violation of rule E4, which ensures no bus contention actually occurs.

You can access simulated gate values using the report_gates command with the gate data set to DRC_pattern. If this is the only DRC violation or if you have changed the default handling to error, you can also view simulated values using report_gates with gate reporting set to error_pattern. Gate reporting with the proper simulation data helps you identify the bidirectional pin that failed, and by tracing connectivity from this point, you can identify how to correct the problem.

Note

You should set the gate data prior to any violation analysis, to ensure the gate data reported is for the correct violation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The occurrence message is:

```
BIDI pin P not set to input mode at time T of P procedure.  (E7-1)
```

P is the pin name of the bidirectional pin, T is the time period, and P is the procedure name.

The summary message is:

```
There were N occurrences of BIDIs not set to input mode during scanning.
(E7)
```

N is the number of occurrences of rules violation E7.

E8

Category: Extra

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

All scan cell MASTER elements of a scan chain must use a single shift clock. If the scan cells contain slaves, all slaves of all scan cells of a scan chain must also use a single shift clock. You can use this rule to ensure that the tester does not cause clock skew problems during the loading and unloading of the scan chains. The application performs this check by inspecting the memory elements for all scan cells of a chain. The rule violation occurs when a chain uses multiple clocks to shift master or slave data.

If multiple shift clocks pulse in the shift procedure and they are blocked from reaching the scan chain by the effects of pin constraints, you must specify the set_drc_handling command with the Atpg_analysis option to avoid an error. This checking considers only pin constraints that have not been overridden by force statements during the shift and load_unload procedures.

The default handling for this rule violation is ignore. Failure to satisfy this rule has no ATPG effect. The occurrence message is:

```
Multiple clocks were used to shift T of scan chain C. (E8-1)
```

T is the type of scan cell memory element (MASTERs or SLAVEs), and C is the chain name.

The summary message is:

```
There were N occurrences of multiply clocked scan chains.  (E8)
```

N is the number of occurrences of rules violation E8.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

If you set the DRC E8 handling to anything other than Ignore, the report_scan_chains command additionally displays the clock of each scan cell. This enhanced report is only available in non-Setup mode.

E9

Category: Extra

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

The drivers of wire gates must not be capable of driving opposing binary values. The application performs this check by attempting to satisfy the placement of opposing binary values on all combinations of the two drivers of a wire gate. The rule violation occurs at a wire gate if it is possible to satisfy those conditions for at least one combination of drivers. When a violation occurs, the tool identifies the failing wire gate and the drivers capable of being placed at opposing values.

This rule ensures that there is no possible contention (for the good machine) on wire gates. The tool does not perform this rule check on wire gates whose behavior you changed to AND or OR using the set_net_resolution command. Also, the tool does not consider pin constraints and equivalences with this check.

The default handling for a violation of this rule is set to ignore. A violation of this rule indicates the possibility that patterns exist that have contention on wire gates. The occurrence message is:

```
WIRE gate N (G) has possible contention on drivers G1 and G2. (E9-1)
```

N is the gate name of the wire gate, G is its gate ID number, and G1 and G2 are the gate ID numbers of the driver gates.

The summary message is:

```
There were N WIRE gates which may have possible contention.  (E9)
```

N is the number of occurrences of rules violation E9.

E10

Category: Extra

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Checks for bus contention mutual-exclusivity during measure_po events in test procedures.

Tri-state drivers must not have conflicting values when driving the same net during the measurement of primary output (PO) values (measure_po events in test procedures). This rule differs from the E4 rule in that it is not checked during test procedure events other than measure_po events. The tool performs this check by analyzing each dominant strong bus to determine if it can cause contention. The analysis places each bus in one of the following categories:

- **Pass** — Test generation analysis determines a contention condition cannot occur.
- **Fail** — Test generation analysis identifies a possible contention condition.
- **Abort** — Test generation analysis terminated while attempting to determine if a contention condition could occur.
- **Bidi** — Test generation determines that the bidirectional pin (which can only have a single tri-state driver) can potentially create a contention condition.

Buses in either the fail or abort categories violate this rule. Buses in the bidi category require checking during post-DRC ATPG processes because they do not exhibit natural mutual-exclusivity behavior and can potentially cause contention. You instruct the tool to perform this additional analysis by using the Atpg_analysis option with the set_drc_handling command. Buses in the pass category require no further checking.

Refer to "Bus Mutual Exclusivity Analysis" in the *Tessent Scan and ATPG User's Manual* for background information about the tool's bus mutual exclusivity checking. For more information about the measure_po statement, refer to the "Test Procedure File" in the *Tessent Shell User's Manual*.

Effect on Testability

Failure to satisfy this rule results in the risk of bus contention during measurement of PO values (measure_po events in test procedures).

How to Debug E10 Violations

The occurrence message is:

```
Bus gate N (G) has possible contention on drivers G1 and G2. (E10-1)
```

N is the gate name of the bus gate, G is the gate ID number, and G1 and G2 are the gate ID numbers of the driver gates.

The summary message is:

```
There were N bus gates which may have possible contention (E10)
```

N indicates the number of bus gates failing the E10 rule.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If you see an “A” appended to the “E10” in the occurrence message (E10-1-**A** for example), it indicates the occurrence is due to the tool aborting the E10 check before it was completed. As these are “possible” but unproven E10s, the tool may be able to screen out some of them if you raise the abort limit. Refer to “Possible Resolutions.”

How to Debug with Tessent Visualizer

To view the location of an E10 DRC violation using Tessent Visualizer, use the following command steps:

1. set_gate_level Primitive
2. open_visualizer

Choose **Open > DRC Browser** from the Tessent Visualizer menu and double-click the desired E10 violation in the table (or right-click and select **Visualizer DRC** from the popup window). The tool automatically changes the gate reporting to “parallel_pattern 0” and displays the bus gate affected by that violation in the **Flat Schematic** tab of Tessent Visualizer. Think of “parallel_pattern 0” as an example of one possible pattern (created for DRC purposes only) that, if included in a test pattern set, would cause contention.

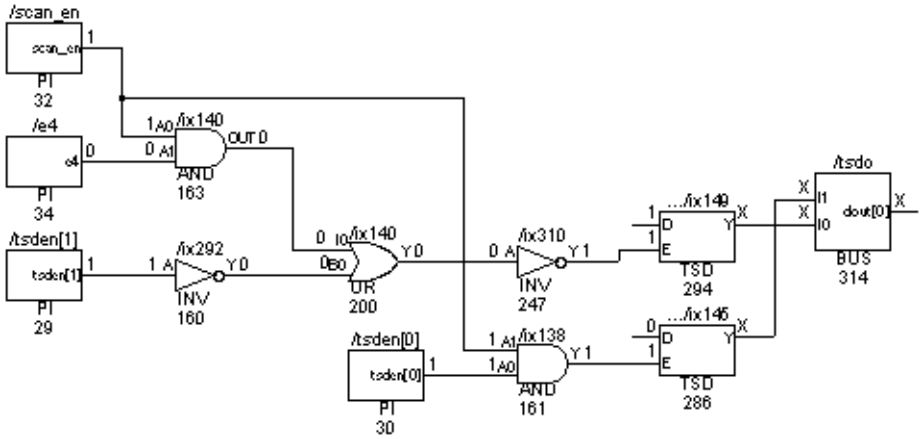
Tip

Alternatively, you can issue the analyze_drc_violation command with the *rule_id-occurrence#* argument at the tool’s command line. For example, to display the first occurrence of an E10 violation:

```
analyze_drc_violation e10-1
```

Once the offending bus gate is displayed, trace backward from the bus gate. The following figure shows an example Tessent Visualizer display of such a trace, where the starting point of the trace is the offending gate, /tsdo. Notice that TSDs /ix145 and /ix149 have conflicting values and both are enabled.

Figure 10-35. Example E10 Violation Trace in Flat Schematic Window of Tessent Visualizer



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You could use Tessent Visualizer to trace back on the enable line of each TSD to the primary inputs and see that both tsden[1] and /scan_en are 1, causing the conflict.

How to Debug from the Tool Command Line

To view the location of the an occurrence of an E10 DRC violation from the tool command line, you can use the following command steps:

1. set_drc_handling E10 Warning
2. set_system_mode analysis
3. report_drc_rules E10-*occurrence#*
4. analyze_bus *gate_id*
5. set_gate_report Parallel_pattern *pattern_number*
6. report_gates *gate_id*

The following show an example of the use of this command sequence starting at step 3:

report_drc_rules e10-1

```
//  Warning: BUS gate /tsdo/dout[0] (314) has possible contention on
//           drivers 294 and 286. (E10-1)
```

analyze_bus 314

```
//  ATPG bus checking performed on /tsdo/dout[0] (314) for 2 bussed TSDs.
//  Failure occurred while checking TSDs 294 and 286 (data in
//  parallel_pattern 0).
```

set_gate_report parallel_pattern 0
report_gates 294 286

```
//  /tsdo/ix149 (294)  TSD
//      E      I  (1)  247-/ix310/Y
//      "D"    I  (1)  221-
//      Y      O  (X)  314-
//  /tsdo/ix145 (286)  TSD
//      E      I  (1)  161-/ix138/Y
//      "D"    I  (0)  213-
//      Y      O  (X)  314-
```

The conflicting values on the input ("D") of each TSD in this example while both are enabled (1 on the "E" input) is the source of the problem. Assuming the two TSDs should not be enabled simultaneously, a reasonable way to proceed is to choose one enable input and trace back through the gates driving it until you find the source of the 1. For example, you might first look at where the /tsdo/ix149 (294) input comes from:

report_gates -endpoints -backward 294

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  ----------------------------------------------------------------------
//  Begin backward trace for gate /tsdo/ix149 (294).
//  ----------------------------------------------------------------------
//  /tsden[1] (29)  PI
//      tsden[1]  O  (1)  160-/ix292/A
//  /scan_en (32)  PI
//      scan_en  O  (1)  163-/ix140/A0  161-/ix138/A1  162-/senmux/ix9/A1
//  /e4 (34)  PI
//      e4     O  (0)  163-/ix140/A1
//  /dataot/reg_q_0_ (373)  DFF
//      "S"    I  (X)  43-
//      R      I  (X)  157-
//      CLK    I  (X)  313-/dataot/ix104/Y
//      "D0"   I  (X)  255-
//      "OUT"  O  (1 [X])  127-  128-
//      MASTER  cell_id=8  chain=chain3  group=grp1  invert_data=TFFT
//  Number gates in trace = 4.
```

This shows that /tsden[1] is a 1. A similar backward trace from instance 286 to /tsden[0] shows that it too is a 1. If you could get a 0 on one of the /tsden inputs, it would remove the contention.

Possible Resolutions

Be aware you may not need to care about E10 violations in certain cases. If you are not concerned about contention after the capture clock is pulsed and you are satisfied with test coverage, you do not need to do anything. On the other hand, if the tool is rejecting many patterns due to contention during simulation and as a result, run time is excessive and coverage low, you may want to do something. By default, the tool checks for contention prior to the capture clock pulse in the capture cycle. If you are concerned about contention after the capture clock is pulsed, you need to direct the tool to check for this type of contention. The key point to remember is you only need to do something if you are not happy with what the tool does by default or the run time or coverage is unacceptable, in which case the following commands are helpful:

Note

These command suggestions do not make E10 rule violations go away, but help you generate contention-free patterns when E10 violations are present.

- add_input_constraints — Adds a pin constraint to a primary input pin. This command is useful if you determine from your troubleshooting and knowledge of the design that forcing a particular primary input pin to a specific value would resolve a contention issue. Typically, the need for such a constraint would have been anticipated during the design phase and perhaps simply overlooked when setting up for ATPG.
- add_atpg_functions and add_atpg_constraints — These commands enable you to create user-defined ATPG function constraints. This can be useful if you know where in the design constraints can be applied to remove contention. Constraints you apply save time the tool would have to spend learning those locations through its own analysis.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Following is an example of the use of these commands to specify constraints:

```
add_atpg_functions func1 select1 tsden[0] ~tsden[1]
add_atpg_constraints 1 func1
```

- set_drc_handling E10 Atpg_analysis — Directs the tool to perform additional analysis toward the end of DRC to determine if buses that the initial E10 DRC placed in the bidi category can cause contention. Increases the tool's DRC effort.
- set_drc_handling E10 -Mode Sequential — This command considers the inputs to a single level of sequential cells behaving as "staging" latches in the enable lines of tri-state drivers. All of the latches found in a back trace must share the same clock. There must also be only a single clocked data port on each cell, and both set and reset inputs must be tied (not pin constrained) to the inactive state. This check ensures that there is no connectivity from the cells in the input cone of the sequential cells and enable of the tri-state devices except through the sequential cells. Using this option increases run time.
- set_abort_limit — Changes the abort limit. Increasing the abort limit can help screen out false E10 violations by enabling the tool to perform more extensive contention checking. This works for E10-*occurrence#*-A violations only.

E11

Category: Extra

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

This rule checks for the ability of a bus gate to attain a Z state. This check analyzes each dominant strong bus to determine if conditions can place a Z value on the bus gate.

As a result, the analysis places each bus in one of the following categories:

- **Pass** — Test generation analysis determines that no condition could place a Z value on the bus gate.
- **Fail** — Test generation analysis determines that conditions could place a Z value on the bus gate.
- **Abort** — Test generation analysis terminated while attempting to determine if conditions could place a Z value on the bus gate.
- **Bidi** — The bus is a bidirectional bus.

Buses in both the fail and abort categories violate this rule.

The default settings for this rule are ignore, noverbose, and atpg_analysis. You can change the handling with the set_drc_handling command. For more information on ATPG analysis, refer to "How to Turn on ATPG Analysis" on page 3170.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The occurrence message is:

```
BUS gate N (G) is capable of attaining a Z state. (E11-1-A)
```

N is the gate name of the bus gate, and G is its gate ID number. The -A following the violation ID number indicates the check aborted.

The summary message is:

```
There were N BUS gates capable of attaining a Z state. (E11)
```

N indicates the number of buses failing the E11 rule.

E12

Category: Extra

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

This rule determines if the test procedures violate any ATPG constraints. The default handling for this rule is ignore. You can change the handling with the set_drc_handling command.

For more information on ATPG analysis, refer to "How to Turn on ATPG Analysis" on page 3170.

The occurrence message is:

```
ATPG constraint violation on N (G) occurred at time T of P procedure.
(E12-1-A)
```

N is the gate name, G is its gate ID number, T is the simulated time period, and P is the procedure name. The -A following the violation ID number indicates the check aborted.

The summary message is:

```
There were N occurences of ATPG constraint violations in test procedures.
(E12)
```

N indicates the number of E12 rule violations.

E13

Category: Extra

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_drc_rules: Supported

This rule determines if it is possible to satisfy both ATPG constraints and bus contention prevention (for buses that fail rule E10). The default settings for this rule are ignore and noverbose.

The occurrence message is:

```
Contention prevention/ATPG constraints satisfiability check failed. (E13-
1)
```

This rule does not issue a summary message. You can change the handling with the set_drc_handling command.

E14

Category: Extra

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Default Handling: Error

report_drc_rules: Supported

The expected simulation values listed in the test_setup procedure must match the actual simulated values.

E14 ensures the test_setup procedure is correct.

This DRC simulates the test_setup procedure when setup mode is exited and compares the simulated values with the expected values listed in the test_setup procedure.

Exception

Not applicable.

Messaging

If the test_setup procedure file fails E14, the following message displays:

```
//  Error: Mismatch on <signal_name> at time <n>, cycle <cycle>
//        of procedure <proc>. Expected <val>, simulated <val>. (E14-#)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When the E14 violation originates from an iRead, it reports the original iRead target, the iCall that called the iProc containing the iRead, the chain flip-flop name if appropriate core data exists, and the iReadVar if applicable. The following message displays:

```
//  Error: Mismatch on <signal_name> at time <n>, cycle <cycle> of
//         procedure <proc>. Expected <val>, simulated <val>.
//         This compare event originates from
//         an iRead command targeting <iRead_target> in iCall <iCall_name>.
//         Use the command 'report_procedures -expand_iprocs' to help debug
//         failures within iProcs. (E14-1.
//         Please make sure that the ICL verification patterns passed
//         simulation with the current netlist.
```

If an ICL ScanRegister has a capture source with known values (that is, if the bits of the CaptureSource property consist of constant values, references to primary inputs, or references to the parallel outputs of other ScanRegister bits with known content) and there is no explicit iRead targeting the Scan Register, then Tessent Shell creates an implied iRead for each of those ScanRegister bits with known capture source. The implied iRead commands are created only for the scan loads in which this ScanRegister is active. If the E14 violation originates from such an implied iRead, one of the following messages displays:

```
//  Error: Mismatch on <signal_name> at time <n>, cycle <cycle> of
//         procedure <proc>. Expected <val>, simulated <val>.
//         This compare event originates from an implied iRead command
//         targeting <register> (<register> captures the constant value <val>)
//         in iCall <iCall_name>.
//         Use the command 'report_procedures -expand_iprocs' to help debug
//         failures within iProcs. (E14-1)

//  Error: Mismatch on <signal_name> at time <n>, cycle <cycle> of
//         procedure <proc>. Expected <val>, simulated <val>.
//         This compare event originates from an implied iRead command
//         targeting <register> (<register> captures the value from the
//         scan register <other_register>) in iCall <iCall_name>.
//         Use the command 'report_procedures -expand_iprocs' to help debug
//         failures within iProcs. (E14-1)
//         Please make sure that the ICL verification patterns passed
//         simulation with the current netlist.

//  Error: Mismatch on <signal_name> at time <n>, cycle <cycle> of
//         procedure <proc>. Expected <val>, simulated <val>.
//         This compare event originates from an implied iRead command
//         targeting <register> (<register> captures the value from the
//         primary input <primary_input>) in iCall <iCall_name>.
//         Use the command 'report_procedures -expand_iprocs' to help debug
//         failures within iProcs. (E14-1)
```

Tessent Shell also verifies the iWrite commands in the test_setup and test_end procedures, if possible. During the processing of an iWrite command that targets an ICL pin, Tessent Shell tries to map the ICL pin to a design pin using the “tessent_design_instance” attributes of the relevant ICL instances. The iWrite verification of a particular ICL pin is only done if all “tessent_design_instance” attributes on the ICL instance path from the top level to the targeted ICL pin are set. As the tessent_design_instance attribute usually contains the rtl name of the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

instance, Tessent Shell also uses the rtl-to-gate mapping mechanism to find the design pin in the gate level netlist. (Refer to the add_rtl_to_gates_mapping command for more information about this mechanism.) If this mapping fails, the following warning appears:

```
//  Warning: In the test_setup iCall <iCall_name>, there is an iWrite command
//           targeting the ICL object <icl_object>, which has been resolved
//           to the driving ICL pin <icl_pin>. The effect of this iWrite
//           command cannot be verified during the E14 check. The rtl path
//           <rtl_path> of the associated design pin, which has been composed
//           from all 'tessent_design_instance' attributes on the ICL instance
//           path to the ICL pin, cannot be mapped to an existing design pin
//           in the netlist.
//           Either the netlist is incomplete or the mapping between rtl
//           description and gatelevel netlist failed. If this problem
//           originates from a mapping failure, use the command
//           'add_rtl_to_gates_mapping -instance_name_map_list' to
//           specify the appropriate instance mapping.
```

If the design pin can be found and it is tied off to a value that conflicts with the iWrite command, the following warning appears:

```
//  Warning: In the test_setup iCall <iCall_name>, there is an iWrite command
//           targeting the ICL object <icl_object>, which has been resolved
//           to the driving ICL pin <icl_pin>. The effect of this iWrite
//           command cannot be verified during the E14 check, because the
//           associated design pin <design_pin> is tied to <val>, while the
//           iWrite command tries to apply the value <other_val> to this pin.
```

If the design pin is an internal input with a constraint that conflicts with the iWrite command, the following warning appears:

```
//  Warning: In the test_setup iCall <iCall_name>, there is an iWrite command
//           targeting the ICL object <icl_object>, which has been resolved
//           to the driving ICL pin <icl_pin>. The effect of this iWrite
//           command cannot be verified during the E14 check, because the
//           associated design pin <design_pin> is constrained to <val>, while
//           the iWrite command tries to apply the value <other_val> to this pin.
```

If the design pin is the ist_enable pin of the Tessent MissionMode in-system test (IST) controller, the tool performs the following actions:

- Locates each ist_enable pin and checks the last simulation value.
- Adds the pin to the suspect list when its last simulation value is not 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Displays a message like the following when it finds a suspect:

```
//  Error: Mismatch on <signal_name> at time <n>, cycle <cycle> of
//         procedure <proc>. Expected <val>, simulated <val>.
//         This compare event originates from an implied iRead command
//         targeting <pin> (<pin> captures the constant value '1').
//         Use the 'report_procedures -expand_iprocs' command to help debug
//         failures within iProcs. Please make sure that the ICL verification
//         patterns passed simulation with the current netlist. (E14-1)
//         Possible root cause: IST controller is not transparent, and it
//         disrupts the IJTAG network initialization. Please make sure that
//         the following pin is set to 0:
//             <ist_enable_pin> (E14-1)
```

When the E14 violation originates from the verification of an iWrite command, it reports the original iWrite target and the iCall that called the iProc containing the iWrite. The following message displays:

```
//  Error: Mismatch on <signal_name> at time <n>, cycle <cycle> of
//         procedure <proc>. Expected <val>, simulated <val>.
//         This compare event originates from the verification of an
//         iWrite to <iWrite_target> during test
//         setup in iCall <iCall_name>.
//         Use the command 'report_procedures -expand_iprocs' to help debug
//         failures within iProcs. (E14-1)
```

The summary message for E14:

```
E14:  #fails=1 handling=warning (Miscompare on test_setup/test_end values)
```

Troubleshooting

1. Run the "set_gate_report drc test_setup" command before the "check_design_rules" command to ensure test_setup data is available for debugging.
2. Use Tessent Visualizer to compare the DRC simulated values from the test_setup procedure with the simulated values for associated nodes in the design.
3. Correct the expect statements in the test_setup procedure to match the DRC simulation values.
4. For iMerges, both the write_procfile command and the report_procedures commands have an optional switch (-EXpand_iprocs) to enable the expansion of the iCalls within a given procedure. If you use an iCall within an iMerge, that entire iMerge is expanded.
5. If an E14 violation occurs and the design has a BISR controller, verify that the fuse box interface has a valid ATPG model or that an input constraint is specified on the fuseValue output. Refer to the section "Overview of Multiple Load ATPG on Memories for Wrapped Cores With Built-in Self Repair" in the *Tessent Shell User's Manual* for details.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

E15

Category: Extra

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Default Handling: Warning

report_drc_rules: Supported

This DRC checks if the PDL-based test setup is stable until the end of the test_setup procedure.

If the test_setup procedure contains iCall statements or iCalls added by means of the set_test_setup_icall command, and if there are iWrite commands in the iProcs referenced by the iCalls, then Tessent Shell creates the required patterns to ensure that the iWrite values are present at the specified locations in the design. The tool identifies the values which can be expected to persist until the end of the test_setup procedure, only considering the ICL and PDL input. Finally, E15 checks if these relevant iWrite values match the results of the test_setup simulation in the stable_after_setup simulation context. As the DRC compares the results of the IJTAG-based value tracking with the results of the design-based test_setup simulation, a failure indicates either that the IJTAG-based test setup has been corrupted by other (non-IJTAG related) actions in the test_setup procedure, or that there are inconsistencies between the design and the ICL description of the IJTAG network.

Like the iWrite verification of the E14 DRC, the E15 DRC also requires a successful mapping between ICL pins and design pins. The same conditions (presence of the "tessent_design_instance" attributes and so on) apply. See the description of E14 for details.

Messaging

If the test_setup procedure file fails E15, the following message displays:

```
// Error: Mismatch on pin '<pin>' at the end of the test_setup procedure.
//        Expected '<val>', simulated '<val>'.
//        During test_setup, the value '<val>' has been applied to ICL element <icl elem>
//        by means of an iWrite command in 'iCall <icall>'.
//        This value is still expected to be present at this pin at the end
//        of test_setup as it has not been overwritten by subsequent iWrite commands
//        or other PDL retargeting activities. (E15-#)
```

E16

Category: Extra

Contexts Supported: all contexts

Default Handling: Error

report_drc_rules: Supported

During test_setup or test_end simulation, if an IJTAG instance required for iRead/iWrite does not exist in the design, or if the instance exists in the design but the network components

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

necessary to access the instance do not exist, it may cause mismatches in test_setup/test_end simulation (E14) or false violations of other DRCs. The E16 DRC is performed during system mode transition before test_setup simulation to flag those missing but required instances.

If an instance flagged by this rule falls in any of the following categories, an E16 violation is reported with corresponding error messages and advice on how to fix it. Other potential reasons for an incomplete instance are currently not considered.

- The instance is defined as a blackbox, or is part of an instance that is defined as a blackbox.
- The instance is defined as interface-only, or is part of an instance that is defined as interface-only.
- The instance is an RTL module but is not synthesized, or is part of an instance that is an RTL module but is not synthesized.

Note that E16 only processes the design instances that can successfully be derived from the involved ICL instances using the hierarchical_design_instance attribute. This attribute is not always available. For details, see hierarchical_design_instance.

Messaging

For a blackbox that has no definition, the following message is issued per module, not per instance:

```
// Error: Design instance 'inst7_pm8x50' is part of the IJTAG network
//        and is active during IJTAG test_setup procedure retargeting,
//        but its module 'empty_mod' has no definition.
//        This may cause mismatches in test_setup simulation (E14
//        violations) or false violations of other DRCs.
//        Please read the definition of this module. (E16-1)
```

For a blackbox that has a definition and is explicitly declared as a blackbox using the "add_black_boxes -module" command, the following message is issued per module, not per instance:

```
// Error: Design instance 'inst7_pm8x50' is part of the IJTAG network and
//        is active during IJTAG test_setup procedure retargeting,
//        but its module 'bb_mod' is the target of the
//        'add_black_boxes -modules' command.
//        This may cause mismatches in test_setup simulation (E14
//        violations) or false violations of other DRCs.
//        Please remove the corresponding 'add_black_boxes -modules'
//        command. (E16-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For a blackbox that was created by the “add_black_boxes -instance” command, the following message is issued per instance:

```
// Error: Design instance ‘inst7_pm8x50/usb_1’ is part of the IJTAG
//        network and is active during IJTAG test_setup procedure
//        retargeting, but it is the target of the
//        ‘add_black_boxes -instances’ command.
//        This may cause mismatches in test_setup simulation (E14
//        violations) or false violations of other DRCs.
//        Please remove the corresponding ‘add_black_boxes -instances’
//        command. (E16-1)
```

For an interface-only case, the following message is issued per module, not per instance:

```
// Error: Design instance ‘inst7_pm8x50/usb_1’ is part of the IJTAG
//        network and is active during IJTAG test_setup procedure
//        retargeting, but its module ‘inf_mod’ is an interface-only
//        module. This may cause mismatches in test_setup simulation (E14
//        violations) or false violations of other DRCs.
//        Please read the full design view for this module. (E16-1)
```

For an unsynthesized RTL module (dft context with no sub-context), the following message is issued per RTL module, not per instance:

```
// Error: Design instance ‘inst7_pm8x50/usb_1’ is part of the IJTAG
//        network and is active during IJTAG test_setup procedure
//        retargeting, but its RTL module ‘rtl_mod’ is not synthesized.
//        This may cause mismatches in test_setup simulation (E14
//        violations) or false violations of other DRCs.
//        Please invoke
//        ‘set_dft_specification_requirements -logic_test on’ command
//        which would force the synthesis of the whole design.
//        Alternatively, you can set the synthesize_before_analysis
//        attribute to ‘true’ on the module to force it to be synthesized.
//        (E16-1)
```

The summary message is:

```
E16:  #fails=3 handling=Error (missing instances for IJTAG network
simulation in test_setup/test_end procedures)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

EDT Finder Rules (F Rules)

F rules verify the EDT logic connectivity and operation prior to scan chain tracing.

F3 3336
F4 3337
F5 3337
F6 3338
F7 3340
F8 3340
F9 3346
F10 3347
F11 3349
F12 3350
F13 3350
F14 3351
F15 3352
F16 3352
F17 3353
F18 3353
F19 3354
F20 3355
F21 3357
F22 3357

F3

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

The design must contain at least one state element able to change states during shift. F3 verifies that at least one state element in the design is clocked and not constrained to a fixed value.

You cannot change the handling with the set_drc_handling command.

The error message is:

```
At least one state element should be able to change its state during
shift. None were found. (F3)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

F4

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

EDT requires a design containing at least one decompressor. F4 verifies there is a decompressor in the design.

You cannot change the handling with the set_drc_handling command.

The error message is:

```
No decompressors found. (F4)
```

F5

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

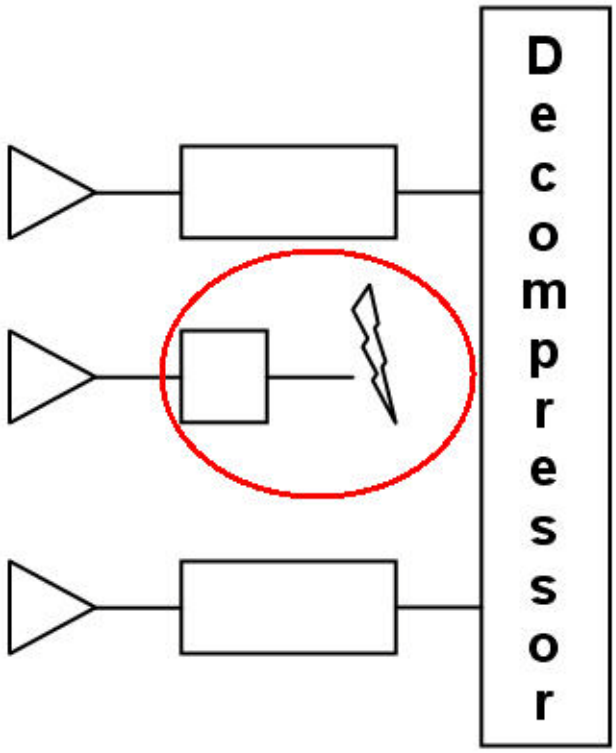
report_drc_rules: Supported

All defined channel inputs must drive a decompressor. F5 verifies that all defined input channels are driving an identified decompressor by tracing the path from the channel input to the decompressor.

Figure 10-36 shows an error condition.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-36. Blocked Decompressor Input



The error message is:

```
Channel input P (N) [of EDT block E] is not driving a decompressor.
Either no structural connection exists, or the path is blocked.
Trace stopped at state element(s) S (G). (F5-1)
```

Where P is the channel index and N the pin name, E is the name of an EDT block, S is an instance name and G is the corresponding gate ID number. Multiple state elements may be listed in case of fanouts.

Troubleshooting

Check the procedure file for incorrect or missing shift procedure settings.

F6

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Warning

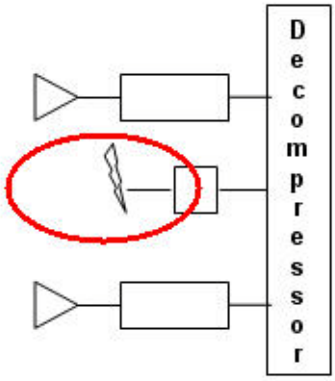
report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Channel input pins that control the decompressors must be defined. F6 traces each decompressor input pipeline to verify that decompressors are controlled by channel input pins, and verifies that the state elements in the channel input pipelines are properly clocked during shift.

Figure 10-37 shows an error condition.

Figure 10-37. Blocked Channel Input to Decompressor



This check is performed even when F5 violations exist.

If the decompressor is driven by one or more defined channel inputs, each blocked input pipeline is reported as follows:

```
The input pipeline of decompressor D [of EDT block E] is broken between
decompressor state element L (G1) and the channel input after tracing N
state elements. The last state element reached is: M (G2). (F6-1)
```

If all input pipelines of a decompressor are blocked the following message is reported once per decompressor.

```
All input pipelines of decompressor D are blocked. One of the input
pipelines is broken between decompressor state element L (G1) and the
channel input after tracing N state elements. The last state element
reached is: M (G2). (F6-1)
```

If the clock input of a channel input pipeline is not set active during shift, the following message is reported:

```
The input pipeline of decompressor D [of EDT block E] is broken between
decompressor state element L (G1) and the channel input after tracing N
state elements. The clock inputs of state element M are never set active
during the shift procedure (F6-1)
```

Where D is the internal identification number of the decompressor, E is the name of an EDT block, L is the instance name of the decompressor state element and G1 is the corresponding gate ID number, N is the number of found pipeline stages, M is the instance name of the last traced state element and G2 is its gate ID number.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

F7

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Decompressors must be controlled by defined channel inputs. F7 traces all inputs to the decompressor to verify that they are connected to channel input pins.

If the decompressor is driven by at least one defined channel input, each rule violation is reported as follows:

```
The pin P is not a channel input but drives decompressor state element L
(G) of decompressor D [of EDT block E]. (F7-1)
```

If the decompressor is only driven by pins not defined as channel inputs, the following error message is reported once per decompressor:

```
None of the primary input pins driving decompressor D is defined as a
channel input. One of the traced pins is P. (F7-1)
```

Where P is a pin name, D is the internal identification number of the decompressor, E is the name of an EDT block, L is the instance name of the decompressor state element and G is the corresponding gate ID number.

Note

In the EDT mapping flow, the reported pins are channel inputs mapped to the top level of the design.

F8

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

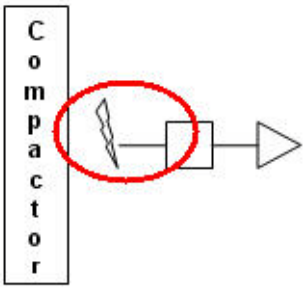
report_drc_rules: Supported

Verifies that all channel outputs are driven by a compactor. F8 traces all channel outputs to verify that they are driven by a compactor, and verifies that the state elements in the channel output pipelines are properly clocked during shift.

Figure 10-38 shows an error condition.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-38. Output Channel and Compactor Connection Blocked



The error message is:

```
Channel output P (N) [of EDT block E] is not driven by a compactor.
Either no structural connection exists, or the path is blocked. The last
state element reached while tracing back is: M (G). The following gate
drives the channel output and blocks the path: M (G) (F8-1)
```

If the clock input of a channel output pipeline is not set active during shift, the following message is reported:

```
Channel output P (N) [of EDT block E] is not driven by a compactor. Either
no structural connection exists, or the path is blocked. The clock inputs
of state element M (G) are never set active during the shift procedure
(F8-1)
```

If a primary input was reached the message is:

```
The primary input pin PI was reached while tracing back from channel
output P (N) [of EDT block E]. (F8-1)
```

If an unclocked element is driving the channel output, the message is:

```
N state elements drive the channel output but could not be identified as
mask hold registers (because they are not driven by mask shift registers)
nor as part of scan chains (because the clock inputs are never set active
during the shift procedure). One of those is: M1 (G1).
...
N is the number of unclocked state element driving the channel output,
M1 is the instance name of one of those, G1 is its gate ID number.
```

If the tracing problem is caused by a defined value on a channel output, the message is:

```
The channel output P (N) [of EDT block E] cannot be traced back because
it is constrained to V. An unconstrained simulation value of X is
expected. (F8-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the tracing problem is caused by wrong values in mask hold registers, the message is:

```
Wrong values detected in mask hold registers after disabling EDT X masking
logic and simulating load_unload until first clock event in apply shift.
One of the failing state elements is: M (G). Simulated value: V1.
Expected value: V2.(F8-1)
```

Where P is the channel index and N is the pin name, E is the name of the EDT block, M is the instance name of the last traced state element, G is its gate ID number, PI is the name of the reached primary input pin, and V, V1, V2 are gate values.

How to Debug Using Tessent Visualizer

An F8 error can occur when the ATPG model for the pad is incorrect or the pad is incorrectly sensitized. When running in EDT, the F8 failure report may show:

```
//  Error: Channel output 1 ('edt_channels_out1') is not driven by a compactor.
//         Either no structural connection exists, or the path is blocked.
//         No state elements reached while tracing back. (F8-1)
```

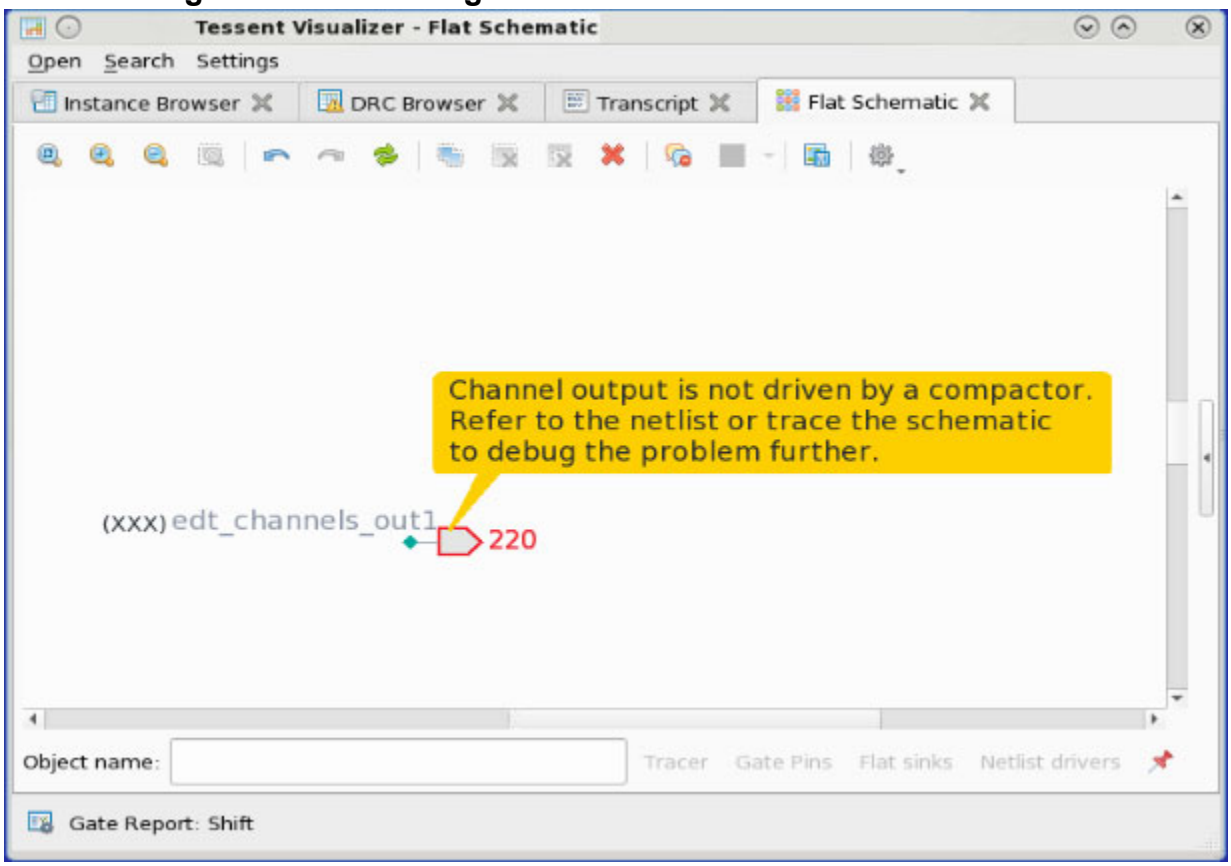
To view the location of the F8 error, use the following command:

analyze_drc_violation F8-1

This command invokes Tessent Visualizer and opens the Flat Schematic window as shown in Figure 10-39.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

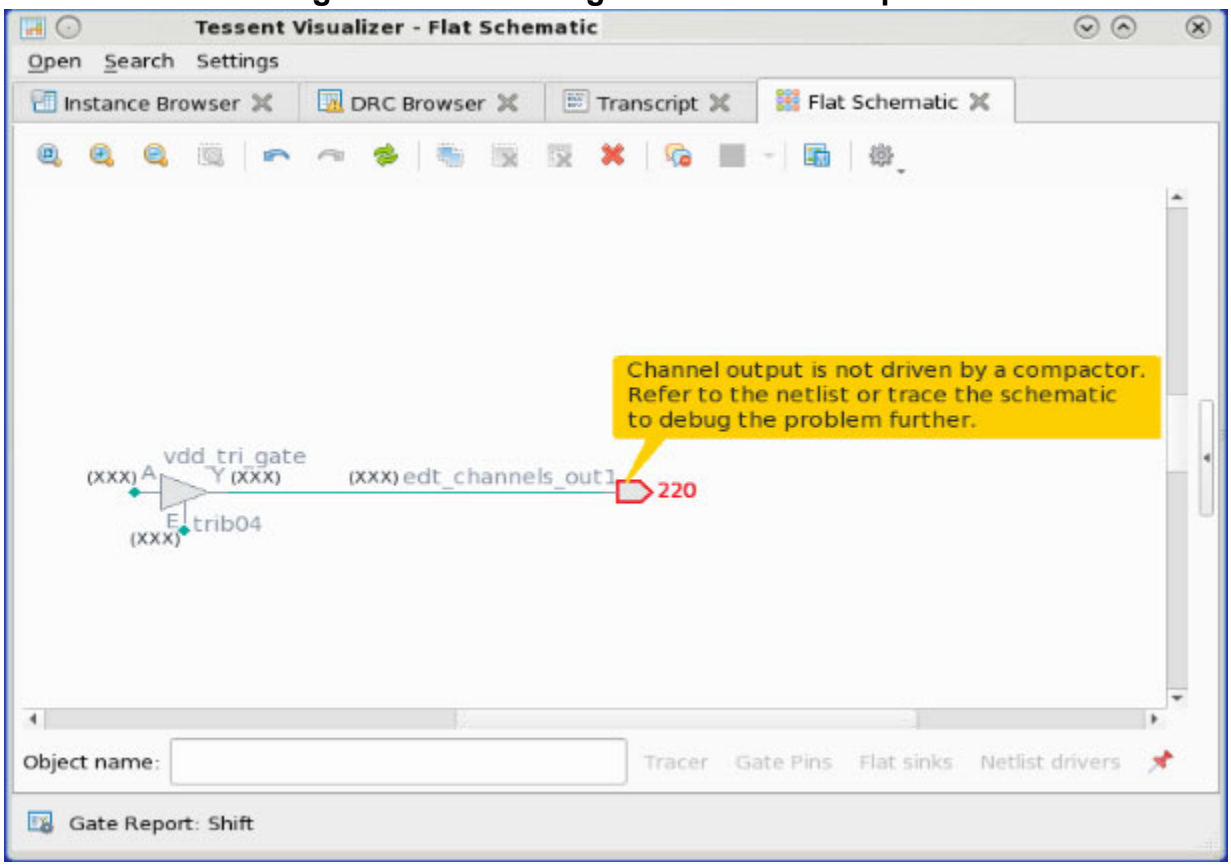
Figure 10-39. Viewing F8 DRC Violation in Tessent Visualizer



To determine why the data is X, you trace backward from the output channel following the X path. Figure 10-40 shows that edt_channels_out1 is driven by a tri-state pad.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

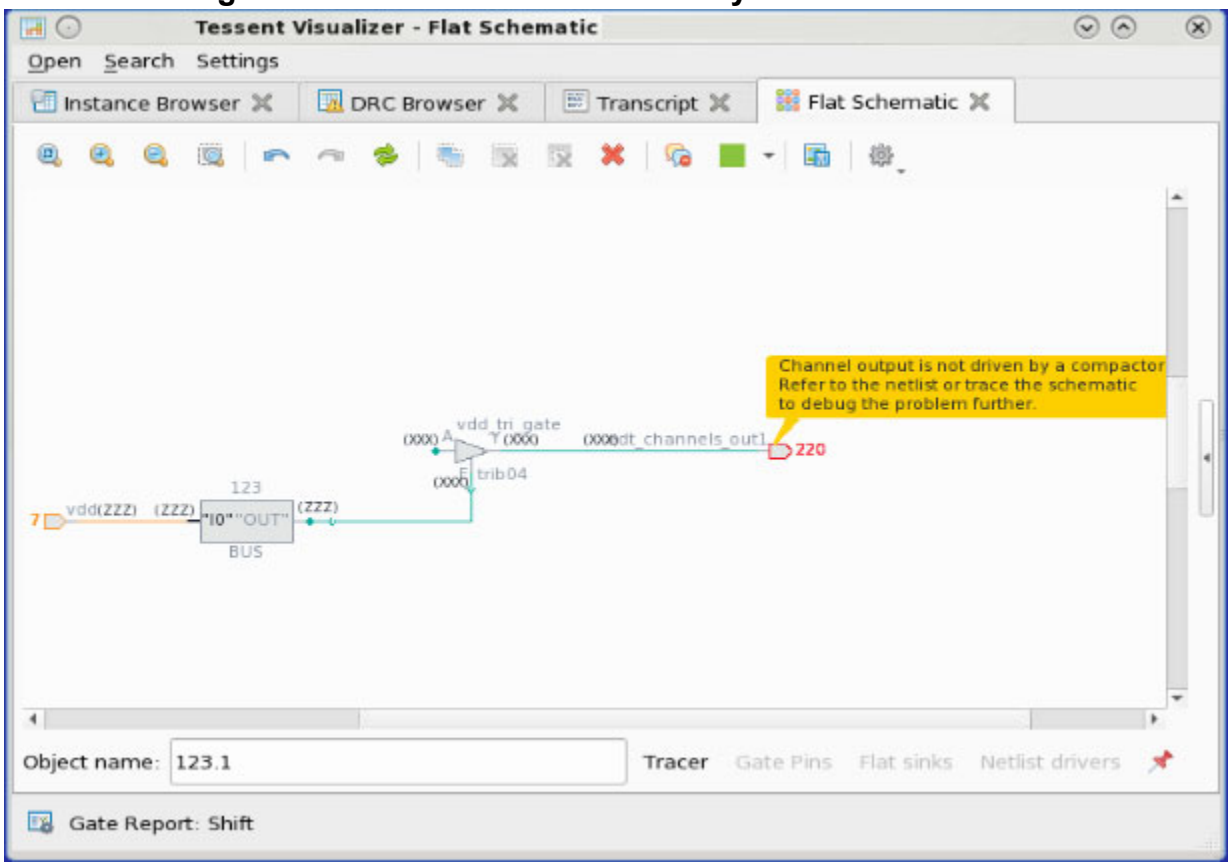
Figure 10-40. Tracing Back on the F8 X path



Notice that the enable pin of the tri-state device is driven X. As you continue to trace back from the enable pin, Figure 10-41 shows that it is driven by the VDD port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-41. Tri-State Driven to X by connection to VDD



Use this command to solve the problem:

add_input_constraint VDD -CT1

Another possible F8 DRC violation displays:

```
//  Error: Channel output 1 ('edt_channels_out1') cannot be traced back
// because it is constrained to 0.
// An unconstrained simulation value (X) is expected. (F8-1)
```

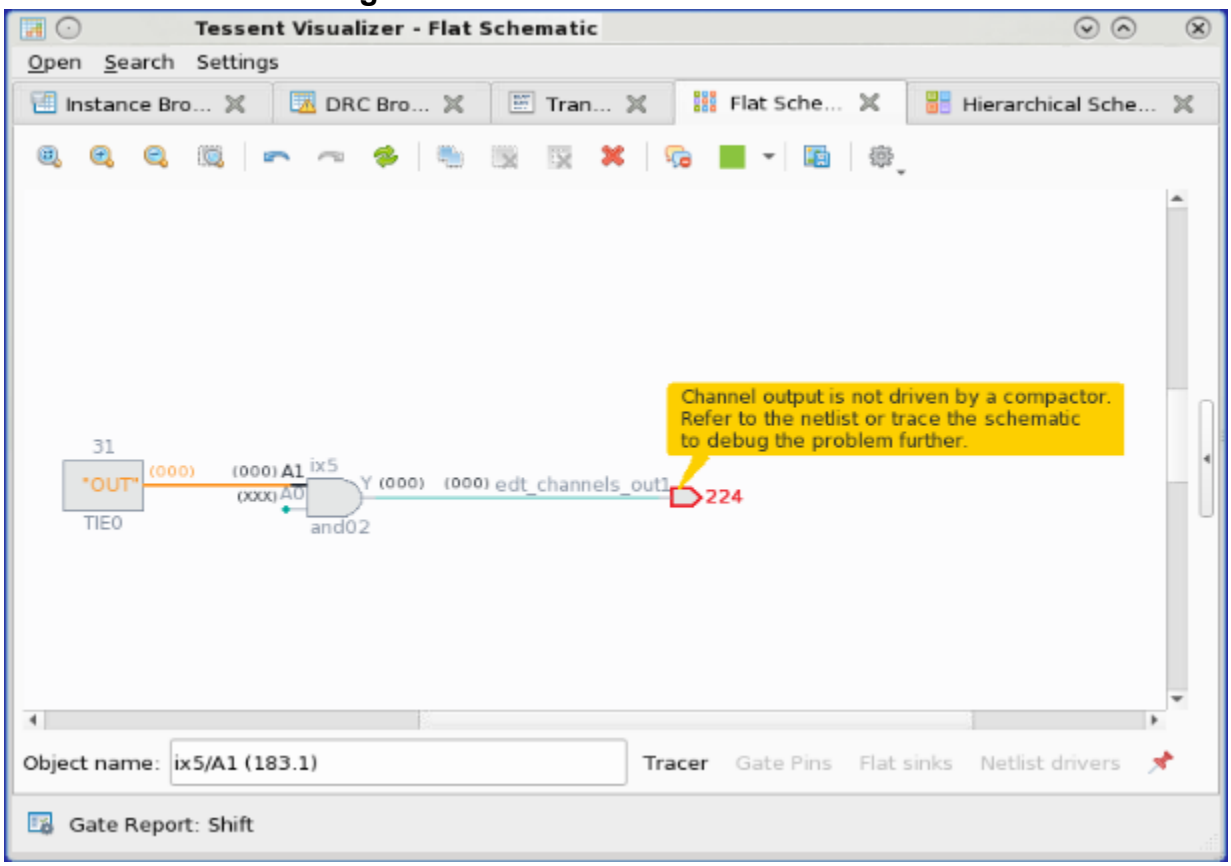
You can use Tessent Visualizer to debug it.

analyze_drc_violation F8-1

To determine why the data is 0, trace back from the edt_channels_out1 pin. In this case, you can see that the output is driven by an AND gate with one pin tied to 0, as shown in Figure 10-42.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-42. F8 DRC Violation Cause



To resolve this problem, you must fix the design to ensure that the AND gate driving the output channel does not have one input tied to 0.

F9

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

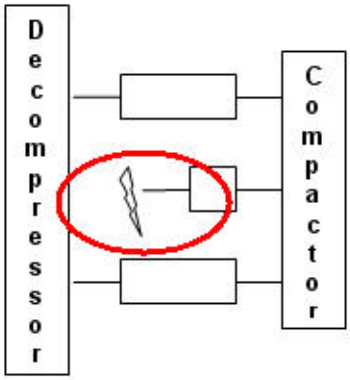
report_drc_rules: Supported

Verifies that all compressed scan chains are connected to a decompressor. F9 traces all channel outputs to verify they connect to a decompressor.

Figure 10-43 shows an error condition.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-43. Blocked Scan Chain Path



The error message is:

```
Compressed scan chain driving channel output P (N) [of EDT block E]
blocked after tracing C cells. The last gate reached while
tracing back is: M (G).  (F9-1)
```

Where P is the channel index and N is the pin name, E is the name of an EDT block, C is the number of scan cells traced in the scan chain, M is the instance name of the last traced gate and G is its gate ID number.

F10

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

This DRC verifies that the EDT logic found matches the EDT logic expected by the tool.

The EDT logic verification steps include, for example, checking the number of found scan chains compared to the number expected, comparing the identified decompressor input taps with the expected ones, and checking if the channel pins driving/connected to a decompressor are consistent with the specifications.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

These are examples of the error messages for this case:

```
The size I1 of decompressor D [of EDT block E] does not match the expected
size I2. (F10-1)

The input taps of decompressor D [of EDT block E] do not match the ones
expected by the tool. For details about the found decompressor use
'report_edt_finder -Decompressors -Id D -Verbose'. (F10-2)

The expected decompressor taps of scan chain S [of EDT block E] do not
match the taps of any found scan chain. (F10-3)

The input taps of channel input N (P) do not match the ones expected by the
tool. Verify that the channel inputs are defined in the correct order.
(F10-4)

The found number of scan chains (8) driven by the decompressor does not
match the expected number (7). (F10-1)

The scan chain 'chain3' could not be found because it does not drive a
channel output. Check if its scan output 'ff6/Q' (268) is properly
connected to the compactor. (F10-2)

Decompressor 3 is driven by a channel input defined for EDT blocks 'blk1'
and ‘blk2' ('core2_5_edt_channels_in1'), but is connected to a channel
output defined for EDT block 'blk3' ('top_out_7'). Please double-check the
EDT channel pin specifications. (F10-1)
```

Where <*D*> is the name of the decompressor, <*E*> is the name of an EDT block, <*S*> is the name of the scan chain, <*N*> is the index number of the channel input, and <*P*> is the pin name.

This DRC also verifies that the status of the low-power hardware is consistent with the user specification:

- If the tool identifies low-power hardware but it is not enabled, the tool issues an error.
- If the tool is unable to identify any low-power hardware but it is enabled, the tool also issues an error.

```
Low power hardware was found but should be disabled according to
user-specified settings.
(F10-1)

The defined low power hardware of EDT block 'blk2' could not be found.
Verify that the hardware is enabled in the test procedures. (F10-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

This DRC also identifies channel pin definition mismatches. These are examples of the error messages for the case:

```
// Error: Decompressor 2 is connected to channel outputs defined for EDT
//     blocks 'm2_4x32' ('m2_edt_channels_out3', 'm2_edt_channels_out4')
//     and 'm3_50x187' ('m2_edt_channels_out1', 'm2_edt_channels_out2'),
//     It is driven by channel inputs defined for EDT block 'm2_4x32'.
//     Please double-check the EDT channel pin specifications. (F10-1)

// Error: Decompressor 2 is driven by channel inputs defined for EDT
//     blocks 'm2_4x32' ('m2_edt_channels_in2', 'm2_edt_channels_in3',
//     'm2_edt_channels_in4') and 'm3_50x187' ('m2_edt_channels_in1').
//     It is connected to channel outputs defined for EDT block 'm2_4x32'.
//     Please double-check the EDT channel pin specifications. (F10-1)

// Error: Decompressor 3 is driven by channel inputs defined for EDT block
//     'm3_50x187', but is connected to channel outputs defined for EDT
//     block 'm2_4x32'.
//     One of the channel inputs is 'm3_pdata[0]' and one of the channel
//     outputs is 'm3_paddr[0]'.
//     Please double-check the EDT channel pin specifications. (F10-1)

// Error: Decompressor 3 is driven by channel inputs defined for EDT
//     blocks 'm2_4x32' ('m3_pdata[0]')
//     and 'm3_50x187' ('m3_pdata[1]', 'm3_pdata[2]', 'm3_pdata[3]').
//     It is connected to channel outputs defined for EDT blocks
//     'm1_28x16' ('m3_paddr[2]', 'm3_paddr[3]') and 'm3_50x187'
//     ('m3_paddr[0]', 'm3_paddr[1]').
//    Please double-check the EDT channel pin specifications. (F10-3)

//  Error: Decompressor 9 is connected to channel outputs defined for EDT
//     block 'blk1' ('edt_channel_out[8]', 'edt_channel_out[18]',
//     'edt_channel_out[20]'), but is not driven by all the channel inputs
//     defined for this EDT block.
//     The following channel input specified for the EDT block
//     'blk1' is not driving decompressor 9: 'edt_channel_in[38]'.
//     Please double-check the EDT channel pin specifications. (F10-1)
```

F11

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Ignore

report_drc_rules: Supported

Verifies that all decompressors connect to at least one scan chain and that the path from the decompressor to the scan chains is not blocked due to wrong or missing shift procedure settings.

The default handling for this DRC is “ignore”.

No correction is required if the decompressor is not needed for testing.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The occurrence message is:

```
Decompressor D has been found but is not used in the current mode. It
contains S state elements, one of which is N (G). (F11-1)
```

Where D is the internal identification number of the decompressor and S is the number of its state elements, N is the instance name of a decompressor state element and G is the corresponding gate ID number.

F12

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Warning

report_drc_rules: Supported

Verifies that all uncompressed scan chains are defined with the add_scan_chains command.

The warning message is:

```
An uncompressed scan chain has been found that was not defined with the
"add scan chain" command. Its scan input is P1, scan output is P2, and it
contains S state elements. (F12-1)
```

Where P1 is a primary input pin, P2 is a primary output pin and S is the number of state elements in the scan chain.

F13

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Verifies that the decompressor, Xpress mask shift register state elements, and low-power shift registers are reset correctly prior to shift.

The error message in the case of the decompressor is:

```
The decompressor [of EDT block E] is not reset correctly prior to shift.
Simulated value: V1. Expected value: V2.
One of the failing decompressor state elements is: M (G). (F13-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In the case of an Xpress compactor, the state elements of the mask shift registers are checked in the same way. The error message is:

```
The mask shift register driven by channel input P (N) [of EDT block E]
is not reset correctly prior to shift.
Simulated value: V1. Expected value: V2.
One of the failing mask shift register state elements is: M (G). (F13-1)
```

In the case of a low-power shift register, the error message is:

```
The low power shift register driven by channel input P (N)
[of EDT block E]is not reset correctly prior to shift.
Simulated value: V1. Expected value: V2.
One of the failing low power shift register state elements is: M (G).
(F13-1)
```

In these messages, P is the channel index and N the pin name, E is the name of the EDT block, M is the name of the failing state element, and G is the gate ID number, and V1 and V2 are gate values.

F14

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Verifies that the state elements of the mask shift registers and the low-power shift registers hold their value during capture.

Only one violation per EDT block is reported.

The error message for mask shift registers (Xpress compactor) is:

```
The mask shift register driven by channel input P (N) [of EDT block E]
does not hold its value during capture. One of the failing mask/power
shift register state elements is: M (G). (F14-1)
```

The error message for mask shift registers (basic compactor) is:

```
The mask shift register [of EDT block E] does not hold its value during
capture. One of the failing mask shift register state elements is: M (G).
(F14-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The error message for a low-power shift register is:

```
The low power shift register driven by channel input P (N)
[of EDT block E] does not hold its value during capture. One of the
failing low power shift register state elements is: M (G). (F14-1)
```

In these messages, P is the channel index and N the pin name, E is the name of the EDT block, M is the name of the failing state element and G its gate ID number.

F15

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Verifies that input pipeline state elements are initialized correctly. F15 verifies that any state elements that exist between a channel input and the EDT logic are initialized to the right value so that 0s get propagated into the EDT logic during the first shift.

Only one violation per EDT block is reported.

You cannot change the handling with the set_drc_handling command.

The error message is:

```
State elements in the input pipeline driven by channel input P (N)
[of EDT block E] are initialized to wrong values after test_setup and
load_unload. One of the failing input pipeline state elements is: M (G).
Simulated value: V1. Expected value: V2. (F15-1)
```

Where P is the channel index and N the pin name, E is the name of the EDT block, M is the name of the failing state element, G is the gate ID number, and V1 and V2 are gate values.

F16

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Input pipeline state elements must preserve their value during capture and load_unload. F16 verifies that any state elements between a channel input and the EDT logic preserve the value shifted into them during capture and load_unload procedure.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For additional information on pipelining, see "Input Channel Pipelines Must Hold Their Value During Capture" in the *Tessent TestKompress User's Manual*.

Only one violation per EDT block is reported.

The error message is:

```
State elements in the input pipeline driven by channel input P (N)
[of EDT block E] do not preserve their value during capture and
load_unload. One of the failing state elements is: M (G). (F16-1)
```

Where P is the channel index and N the pin name, E is the name of the EDT block, M is the name of the failing state element, and G is the gate ID number.

F17

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Verifies that the state elements of the mask hold registers and the low-power hold registers are clocked during load_unload.

Only one violation per EDT block is reported.

The error message reported when mask hold registers violate this rule is:

```
Mask hold register state elements [of EDT block E] are not clocked during
load_unload. One of the failing mask hold register state elements is:
M (G). (F17-1)
```

The error message reported when a low-power hold register violates this rule is:

```
Low power hold register state elements [of EDT block E] are not clocked
during load_unload. One of the failing low power hold register state
elements is: M (G).
(F17-1)
```

In these messages, E is the name of the EDT block, M is the name of the failing state element of the low-power hold register and G is its gate ID number.

F18

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_drc_rules: Supported

Verifies that the state elements of the mask hold registers load their values from the state elements of the mask shift registers before shift in the load_unload procedure; verifies that the state elements of the low-power hold registers load values from the low-power shift registers before shift in the load_unload procedure

Only one violation per EDT block is reported.

The error message reported for mask hold registers is:

```
Mask hold register state elements [of EDT block E] do not load values from
mask shift registers before shift in the load_unload procedure.
One of the failing mask hold register state elements is: M1 (G1).
The driving mask shift register state element is: M2 (G2).
Simulated value: V1. Expected value: V2.
```

The error message reported when a low-power hold register violates this rule is:

```
Low power hold register state elements [of EDT block E] do not load values
from low power shift registers before shift in the load_unload procedure.
One of the failing low power hold register state elements is: M1 (G1).
The driving low power shift register state element is: M2 (G2).
Simulated value: V1. Expected value: V2. (F18-1)
```

In these messages, E is the name of the EDT block, M1 is the name of the failing state element, G1 is its gate ID number, M2 is the name of the driving state element of the low-power hold register and G2 its gate ID number, and V1 and V2 are gate values.

F19

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Verifies that the state elements of the mask hold registers and low-power hold registers preserve their values during shift.

Only one violation per EDT block is reported.

The error message reported for mask hold registers is:

```
Mask hold register state elements [of EDT block E] do not preserve their
values during shift. One of the failing mask hold register state elements
is: M (G).(F19-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The error message reported for low-power hold registers is:

```
Low power hold register state elements [of EDT block E] do not preserve
their values during shift. One of the failing low power hold register
state elements is: M (G). (F19-1)
```

In these messages, E is the name of an EDT block, M is the name of the failing state element of the low-power hold register and G is its gate ID number.

F20

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

The F20 design rule check verifies the masking modes of a compactor.

F20 programs the compactor to the various masking modes by directly loading the mask hold registers and tracing backward from the channel outputs (or from the first pipeline stage in case of a pipelined channel output) to verify that only the expected scan chains are driving the channel outputs. F20 also checks for masked chains with incorrect unload specification.

If the F20 DRC has verified the masking modes (that is, if the check is not set to “ignore”), the tool skips the corresponding K22 DRCs.

Mask-all Masking Mode

If the compactor supports the mask-all masking mode, F20 verifies that all scan chains are masked in that mode; that is, the channel output receives a “0” (or a “1” in the case of an inversion). In the case of a violation, the error message is:

```
Mask-all masking mode: The channel output P (N) [of EDT block E] receives
an X although all scan chains driving it are expected to be masked. The
last reached gate while tracing back is: M (G) [which is the scan output
of scan chain 'S']. (F20-1)
```

Where P is the channel index and N is the pin name, E is the name of an EDT block, M is the instance name of the last traced gate and G is its gate ID number, and S is the name of the found scan chain. Note, the string “[which is the scan output of scan chain ‘S’]” is only reported if M is the scan output of a scan chain.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

1-Hot Masking Mode

F20 verifies that only the expected scan chain drives the expected channel output. All other scan chains are masked. If the expected channel output is not driven by the expected scan chain the error message is:

```
1-hot masking mode for scan chain 'S1':
The channel output P (N) [of EDT block E] is not driven by gate M1 (G1)
which is the scan output of scan chain 'S1'.
The last reached gate while tracing back is:
M2 (G2) [which is the scan output of scan chain 'S2']. (F20-1)
```

Where P is the channel index and N is the pin name, E is the name of an EDT block, S1 is the name of the expected scan chain, M1 is the instance name of the scan output gate of scan chain S1 and G1 is its gate ID number, M2 is the instance name of the last traced gate and G2 is its gate ID number, and S2 is the name of the found scan chain. Note, the string "[which is the scan output of scan chain 'S2']" is only reported if M is the scan output of a scan chain.

F20 checks for masked chains with incorrect unload specification and produces an error message similar to the following:

```
//  Finding internal scan chains.
//  Error: The found number of scan chains (2) driven by the decompressor
//  does not match the expected number (1). (F10-1)
//  Error: 1-hot masking mode for scan chain 'chain1':
//      The channel output 1 (/edt_channels_out1) is driven by gate
//      '/skeleton_design_top_i/chain1_cell0/Q' (12) which is the scan
//      output of chain 'chain1' but the scan chain is defined to be
//      masked and to unload a constant 0.
//      The last gate reached while tracing back is:
//      '/skeleton_design_top_i/chain1_cell0/Q' (12) which is the scan
//      output of chain 'chain1' (F20-1)

//  Error: XOR masking mode 2: The channel output 1 (/edt_channels_out1)
//  is not driven by the expected scan chains.
//    The following unexpected driver has been found:
//    '/skeleton_design_top_i/chain1_cell0/Q' (12) which is the scan
//    output of chain 'chain1' (F20-2)
```

Xpress Masking Mode

If the compactor supports the Xpress masking modes, F20 verifies that only the expected scan chains drive the expected channel outputs. All other scan chains are masked. If the expected channel output is not driven by the expected scan chains, the error message is:

```
XOR masking mode K: The channel output P (N) [of EDT block E] is not driven
by the following scan chains:
S1, S2, …, SN as expected.
The last reached gate while tracing back is:
M (G) [which is the scan output of scan chain 'S']. (F20-1)
```

Where K is the index of the masking mode, P is the channel index and N is the pin name, E is the name of an EDT block, S1, S2, …, SN are the names of the expected scan chains, M is the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

instance name of the last traced gate and G is its gate ID number, and S is the name of the found scan chain. Note, the string [which is the scan output of scan chain 'S']" is only reported if M is the scan output of a scan chain.

F21

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

The F21 design rule check verifies that all scan chains driving a compactor have the expected latency; that is, the expected number of pipeline stages between the scan output and the channel output.

In some cases, EDT Finder learns and adjusts the number of compactor and output channel pipeline stages to match what was found in the design. If it succeeds in learning and adjusting the number of pipeline stages, the F21 DRC is not issued.

EDT Finder only adjusts the number of compactor pipeline stages if the number of compactor pipeline stages in the EDT IP was not passed from the IP Creation phase. Normally this information is passed through and any discrepancy results in a DRC violation.

In the case of a violation, the error message is:

```
Incorrect number of pipeline stages between the scan chain 'S'
and the channel output P ('N') [of EDT block 'E']: identified=N1
expected=N2.
The scan output of scan chain 'S' is M (G)
The following pipeline stages have been identified:
P1 (G1)
P2 (G2)
…
PN (GN) (F21-1)
```

Where S is the name of the scan chain, P is the channel index and N is the pin name, E is the name of an EDT block, N1 and N2 are the number of identified and expected pipeline stages, M is the scan output of the scan chain and G is its gate ID number, and P1, P2, …, PN are the identified pipeline stages and G1, G2, …, GN their gate ID numbers.

F22

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Warning

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_drc_rules: Supported

When two adjacent memory elements (source and sink) in the scan path are clocked by different shift clocks, the sink must not capture data from the source at the same time that the source changes its value. Failure to satisfy this rule can result in unwanted shoot-through during scan shifting when clock skew exists between the different shift clocks.

This DRC verifies this rule for EDT-related state elements. This DRC complements the T24 DRC that verifies this rule for the scan chains. Together, these two DRCs cover the complete scan path.

In the case of a violation, the error message is:

```
Two adjacent pipeline cells of channel input P (N)[of EDT block E]
are clocked on the same edge by different clocks.

N1 (G1), driven by clock N3 (G3), and
N2 (G2), driven by clock N4 (G4)

Consider adding a lockup cell between them, or modifying the clock timing.
(F22-1)
```

The first sentence of the error message depends on the location of the violation. Other possibilities are:

```
Two adjacent cells of the pipeline of channel input P (N) and the
decompressor [of EDT block E]…

Two adjacent cells of the decompressor [of EDT block E] and scan chain
'C1' …

Two adjacent cells of scan chain 'C1' and the compactor [of EDT block E] …

Two adjacent cells of scan chain 'C1' and the pipeline of channel output P
(N) [of EDT block E]…

Two adjacent cells of the compactor [of EDT block E]…

Two adjacent cells of the compactor and the pipeline of channel output P
(N) [of EDT block E]…

Two adjacent pipeline cells of channel output P (N) [of EDT block E]…
```

N1 and N2 are the instance names of the adjacent elements clocked by different clocks, N3 and N4 are the instance names of the clocks, G1-G4 are the corresponding gate IDs, P is the channel index and N the pin name, E is the name of an EDT block, and C1 is the name of a scan chain.

Note

If the F22 DRC has verified this rule (that is, the set_drc_handling command is not set to "ignore" F22 violations) the K21 DRC is skipped. This only applies to the pattern generation phase and not the IP creation phase.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

How to Debug Using Tessent Visualizer

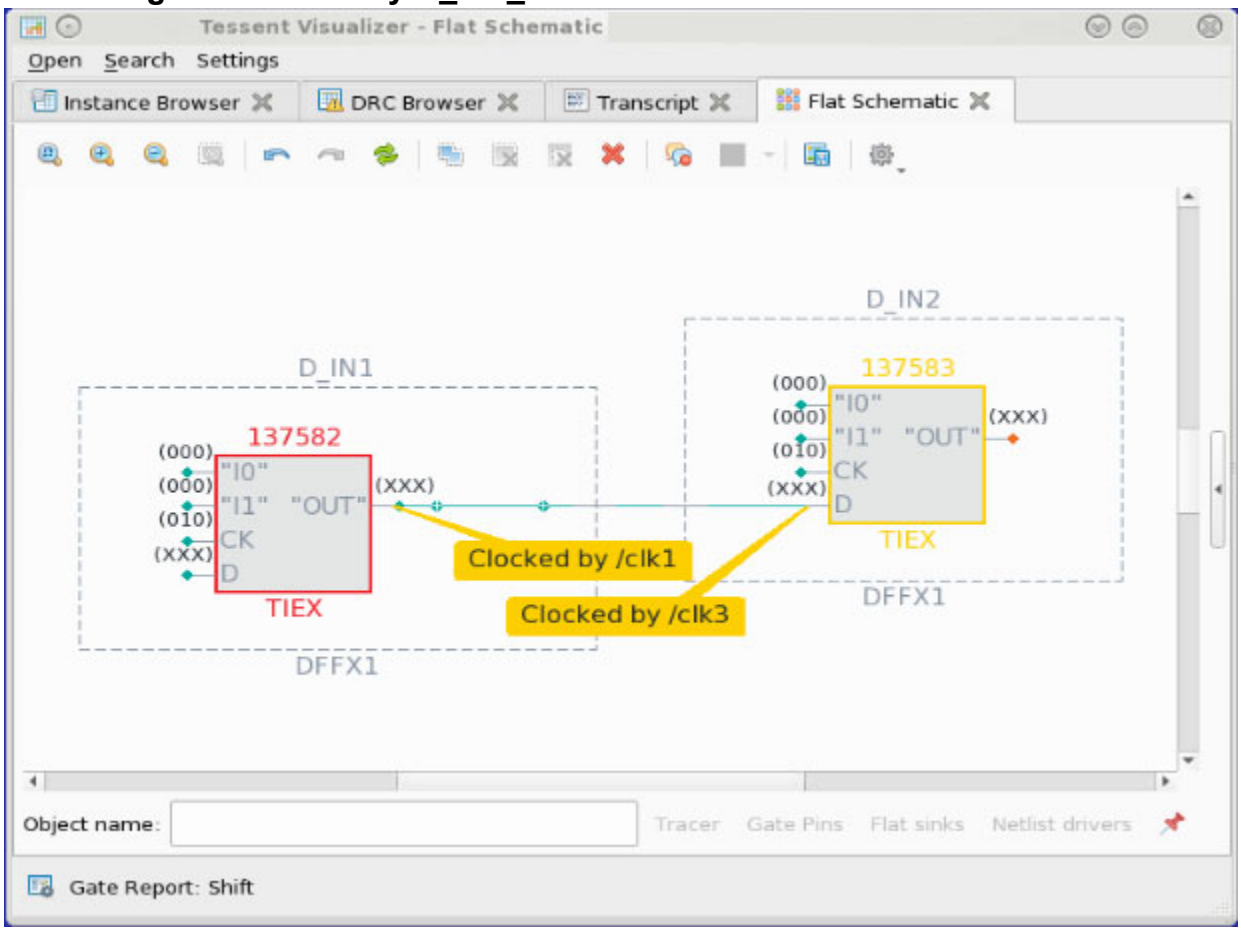
To view the location of a F22 DRC violation using Tessent Visualizer, use the following command:

analyze_drc_violation F22-1

This command invokes Tessent Visualizer and opens the Flat Schematic window.

The Flat Schematic window shows the connected path between the two adjacent cells and displays in the callouts the two different clocks that pulse on the same edge. The gate report is set to the shift procedure so you can see that both cells capture on the same edge.

Figure 10-44. analyze_drc_violation F22 in Flat Schematic Window



This type of violation typically occurs because during IP creation the clock of the first scan cell in the chain is leading edge (010), but after synthesis the first cell is actually a trailing edge flop (101). When you are using the skeleton flow, it is important that the edge of the first and last cell is defined correctly or the tool may end up with an extra or missing lockup cell.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Solution

You can correct this violation by inserting a lockup cell between the adjacent memory elements reported in the violation message, or by regenerating the EDT IP using the correct edges for the first cell.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Flattening Rules (FN, FP, and FG Rules)

The flattening rules check the hierarchical design's suitability for flattening.

The flattening rule violations and their identification literals are divided into the following groups:

- **Net Rules** — FN1 through FN9
- **Pin Rules** — FP1 through FP13.
- **Gate Rules** — FG3, FG7 and FG8.

You can change the handling of the net, pin, and gate flattening rules with the set_flattener_rule_handling command.

FN1 3362
FN2 3362
FN3 3362
FN4 3362
FN5 3363
FN6 3363
FN7 3363
FN8 3363
FN9 3364
FP1. 3364
FP2. 3364
FP3. 3364
FP4. 3364
FP5. 3365
FP6. 3365
FP7. 3365
FP8. 3365
FP9. 3366
FP10. 3366
FP11. 3366
FP12. 3366
FP13. 3366
FG3 3367
FG7 3367

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

FG8 3367

FN1

Category: Flattened Net

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule determines if a module net is floating. By default, the tool ignores this rule in the RTL context.

You can overwrite this default handling by using the set_flattener_rule_handling command.

FN2

Category: Flattened Net

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule determines if a module net has a driver and a constant value property. The constant value property is ignored.

FN3

Category: Flattened Net

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning for non-RTL contexts, Ignore for RTL contexts. (You can overwrite this default handling by using the set_flattener_rule_handling command.)

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule determines if an instance net is floating (has no driver/source) but the net does have a drain/sink. Note that if the net also has no drain/sink, the net incurs an FN4 violation rather than FN3.

FN4

Category: Flattened Net

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This rule determines if an instance net is not used (has no driver/source, and also has no drain/sink). Note that if the net has a drain/sink but no driver, the net incurs an FN3 violation rather than FN4.

FN5

Category: Flattened Net

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule identifies a wired net with multiple drivers.

FN6

Category: Flattened Net

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule is violated when a net declared to be WAND (Wired-AND) or WOR (Wired-OR) has a Z driver, and therefore cannot be simulated as Boolean AND or OR functions, respectively. The net becomes a BUS, and the net wire type declaration is ignored.

FN7

Category: Flattened Net

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule determines if two connected nets have inconsistent net attributes. Both inconsistent attributes are ignored.

FN8

Category: Flattened Net

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule identifies parallel wired behavior.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

FN9

Category: Flattened Net

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule determines if the bus net has multiple bus keepers. The effects of the bus keepers are additive.

FP1

Category: Flattened Pin

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule determines if the circuit has no primary inputs.

FP2

Category: Flattened Pin

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule determines if the circuit has no primary outputs.

FP3

Category: Flattened Pin

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule determines if a primary input drives logic gates and switch gates.

FP4

Category: Flattened Pin

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_drc_rules: Not Supported. Use report_flattener_rules.
This rule determines if a pin was moved.

FP5

Category: Flattened Pin
Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis
Default Handling: Warning
report_drc_rules: Not Supported. Use report_flattener_rules.
This rule determines if a pin was deleted by merging.

FP6

Category: Flattened Pin
Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis
Default Handling: Warning
report_drc_rules: Not Supported. Use report_flattener_rules.
This rule identifies merged wired in/out pins.

FP7

Category: Flattened Pin
Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis
Default Handling: Warning
report_drc_rules: Not Supported. Use report_flattener_rules.
This rule identifies merged wired input and output pins.

FP8

Category: Flattened Pin
Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis
Default Handling: Warning
report_drc_rules: Not Supported. Use report_flattener_rules.
This rule determines if a module boundary pin has no name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

FP9

Category: Flattened Pin

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule determines if an inout pin is used as output only.

FP10

Category: Flattened Pin

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Not Supported. Use report_flattener_rules

This rule determines if an output pin is used as an inout pin.

FP11

Category: Flattened Pin

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule determines if an input pin is used as an inout pin.

FP12

Category: Flattened Pin

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule determines if an output pin has no fanout.

FP13

Category: Flattened Pin

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This rule determines if an input pin is floating. By default, the tool ignores this rule in the RTL context.

You can overwrite this default handling by using the set_flattener_rule_handling command.

FG3

Category: Flattened Gate

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule determines if the bus keeper has no functional impact.

FG7

Category: Flattened Gate

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule determines if the netlist module has a primitive that is not supported. If you reduce the handling to warning or note, this primitive is treated as undefined.

FG8

Category: Flattened Gate

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported. Use report_flattener_rules.

This rule determines if the library model has a primitive that is not supported. If you reduce the handling to warning or note, this primitive is treated as undefined.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

General Rules (G Rules)

G rules are general rules that identify inconsistencies in scan data and other definitions. G rules are the first rules checked during the DRC process and all violations generate error conditions that must be corrected. You cannot change the handling of these rules.

G1 3368
G2 3368
G3 3369
G4 3369
G5 3369
G6 3370
G7 3370
G8 3370
G9 3371
G10 3371
G11 3372
G12 3372

G1

Category: General

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Each defined scan chain group, except for “dummy”, must contain at least one scan chain. You can correct this error condition by either adding a scan chain to the group or by deleting the scan chain group.

The error message is:

```
No scan chains have been defined for group N. (G1-1)
```

N is the name of the scan chain group, and G1-1 indicates the rule and violation ID numbers.

G2

Category: General

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_drc_rules: Not Supported

If you define scan chains and do not use the “dummy” scan chain option, you must define at least one clock. You can correct this error condition by either defining a clock that controls the defined scan chains or by deleting all scan chain groups.

The error message is:

```
Scan chains exist but no synchronous clocks have been defined. (G2-1)
```

G3

Category: General

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

If the circuit has no memory elements, you cannot define clocks. You can correct this error condition by deleting all clocks.

The error message is:

```
Clocks are defined but no memory elements exist in the circuit. (G3-1)
```

G4

Category: General

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

If the circuit has no memory elements, you cannot define scan chain groups. You can correct this error condition by deleting all scan chain groups.

The error message is:

```
Scan groups are defined but no memory elements exist in the circuit.(G4-1)
```

G5

Category: General

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

If there are no RAMs in the circuit, you cannot define write control lines. You can correct this error condition by deleting all write control lines.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The error message is:

```
Write controls are defined but no RAMs exist in the circuit. (G5-1)
```

G6

Category: General

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

If you define a linear feedback shift register (LFSR), you cannot use the “dummy” scan chain option. You can correct this error condition by either deleting all LFSRs or deleting the dummy scan chain group.

The error message is:

```
Cannot use dummy scan chain with BIST LFSRs. (G6-1)
```

G7

Category: General

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

The RAM/ROM instance name given on a preceding read_modelfile command must contain a single RAM or ROM gate. You can correct this error condition by using the correct RAM or ROM instance name for the read_modelfile command. Or, you can do nothing and re-invoke the rules checker, in which case, the tool does not use a modelfile for the intended RAM or ROM.

The error message is:

```
Cannot use RAM/ROM modelfile M for invalid instance N. (G7-1)
```

M is the modelfile name, N is the instance name, and G7-1 indicates the rule and violation ID numbers.

G8

Category: General

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

All ROM gates must have a defined initialization file, unless you use the random initialization option. You can correct this error condition by specifying a modelfile in the model cell library, using the read_modelfile command to specify a modelfile, using random initialization, or changing the model cell library to treat the ROM gate as undefined.

The error message is:

```
ROM initialization file not defined for N (G). (G8-1)
```

N is the instance name of the ROM, G is the gate ID number, and G8-1 indicates the rule and violation ID numbers.

G9

Category: General

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

For all constrained scan cells identified by chain and position, the scan chain must be a valid scan chain, the position must be less than the length of the chain, and the scan cell must not be the same as another constrained scan cell. You can correct this error by identifying and correcting all invalid scan cell constraints.

The error message is:

```
Invalid cell constraint position P for chain C. (G9-1)
```

P is the cell position number (0-based, where 0 is the scan cell closest to the scan-out pin), C is the scan chain name, and G9-1 indicates the rule and violation ID numbers.

G10

Category: General

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

For all constrained scan cells identified by pin pathname, the pin must be a valid output pin of a cell, the pin must connect to a scan memory element through a path that only contains buffers and inverters, and the scan cell must not be the same as another constrained scan cell. To correct this error, you must identify and correct all invalid scan cell constraints.

The error message is:

```
Invalid cell constraint pin name P. (G10-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

P is the pin pathname of an output pin of a cell, and G10-1 indicates the rule and violation ID numbers.

G11

Category: General

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

If you define a “dummy” scan chain group with a test procedure file, you cannot define any scan chains. The purpose of the dummy scan group is to provide the ability to use a test_setup procedure when no scan cells exist. To correct this error (if scan cells do exist), you should place the test_setup procedure in the test procedure file for a defined scan chain group.

The error message is:

```
Scan chains may not be defined when using dummy scan group procedure file.
(G11-1)
```

G12

Category: General

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

General rule 12 reports add_cone_blocks command pin-pathname violations because their checking is delayed until DRC is performed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL Extraction Rules (I Rules)

Tessent Shell currently performs limited ICL Extraction rule checks on the design.

Table 10-11 lists the ICL-specific DRC rules that check the mandatory preconditions the design must have in order to extract the ICL top level from the netlist.

Table 10-11. ICL Extraction DRCs

DRC	Description
I1	Performs consistency checks on each design module that was mapped to an ICL module during the execution of the set_current_design command. Also verifies that the direction of each port on the mapped ICL module is consistent with the direction of the corresponding port in the design module.
I2	Verifies that a connection can be identified from an ICL-attributed pin to another ICL-attributed pin or to a top-level port. Also checks if the simulated value for every ICL-attributed instance input pin is compatible with the tied value of 1 or 0 specified in the ICL.
I3	Verifies that the port functions of an identified connection between an ICL-attributed port on an ICL-attributed module match the port function connection rules.
I4	Checks if an inferred port function and polarity for the primary input/output port is consistent for all found connections.
I5	Checks the consistency of specified pin constraints on ICL-attributed design instances.
I6	Checks all latches that have been previously identified during forward tracing of design instance pins with a scan out ICL attribute, and verifies that the backward tracing of the enable port of those latches reaches an identified primary port with TCK ICL attribute. In addition, all these latches have to be enabled by a low signal on the TCK port.
I7	Checks if the clock properties of design ports specified by the add_clocks command are consistent with the port functions and function_modifier attributes of the related extracted ICL ports.

The ICL-specific DRC rules described in this section are based on the IEEE 1687-2014 (IJTAG) standard. These checks are only performed in the Tessent Shell patterns -ijtag context—see "ICL Extraction" in the *Tessent IJTAG User's Manual*.

Refer to "Design Rule Checks" in the *Tessent IJTAG User's Manual* for complete information.

I1

Category: ICL Extraction

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Performs consistency checks on each design module that was mapped to an ICL module during the execution of the set_current_design command.

Specifically, the tool verifies that all ports on the mapped ICL module are present in the corresponding design module. If an ICL module port is not found in the corresponding design module, the tool issues the following error messages:

```
The port 'P' exists in the description of the ICL module 'M1' but not on
the matching design module 'M2'.
```

Where:

'P' is the name of the ICL module port, 'M1' is the name of the ICL module, and 'M2' is the name of the design module matching the ICL module 'M1'.

The tool also verifies that the direction of each port on the mapped ICL module is consistent with the direction of the corresponding port in the design module. If a design module port has a conflicting direction, the tool issues the following error messages:

```
DataInPort 'PI' exists in the description of the ICL module 'M1'. However,
 the corresponding port in the design module 'M2' is not of a
compatible direction. An ICL port of type DataInPort must map to a design
port that is either an input or bidirectional port.

DataOutPort 'PO' exists in the description of the ICL module 'M1'.
 However, the corresponding port in the design module 'M2' is not of a
 compatible direction. An ICL port of type DataOutPort must map to a
 design port that is either an output or bidirectional port.
```

Where:

```
'PI' and 'PO' are the names of the ICL module ports, 'M1' is the name of
 the ICL module, and 'M2' is the name of the design module matching the ICL
 module 'M1'.
```

Note

Visualization for the I1 DRCs is not possible due to the nature of such errors.

Note

Logic synthesis can flatten or bit-blast ports, causing post-synthesis names to differ from those in the initial RTL. This process is automatic, and typically does not require user intervention. For more information, refer to "Automatic Design Mapping" in the *Tessent Shell User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

I2

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that a connection can be identified from an ICL-attributed pin to another ICL-attributed pin or to a top-level port.

This DRC violation typically occurs when a pin is unconnected. Specifically, this rule checks that every ICL-attributed instance input pin is connected to either an ICL-attributed instance output pin or a primary input port, and every ICL-attributed instance output pin is connected to either an ICL-attributed instance input pin or a primary output port.

The tool also checks if the simulated value for every ICL-attributed instance input pin is compatible with the tied value of 1 or 0 specified in the ICL using the allowed_tied_high or allowed_tied_low value for the connection_rule_option attribute, and reports an I2 violation if there is a mismatch. If the simulated value is X and the connection_rule_option is not allowed_no_source then the failure to drive the input pin is reported.

These checks are performed for all ICL input ports and for ICL output ports with the port functions DataOutPort and ScanOutPort.

If a connection cannot be identified, the tool issues the following error message:

```
The DataOutPort 'Q' on ICL-attributed design instance 'I', module 'M' does
 not trace to any primary or ICL module port.
Either no structural connection exists, or the path is blocked.
```

If the trace stopped at only one gate, which is always the case in the backward trace, the tool issues one of the following error messages which includes the gate information:

```
The DataOutPort 'Q' on ICL-attributed design instance 'I', module 'M' does
 not trace to any primary or ICL module port.
Either no structural connection exists, or the path is blocked.
The backward trace stopped at gate '/I/Q'.

The DataOutPort 'Q' on ICL-attributed design instance 'I', module 'M' does
 not trace to any primary or ICL module port.
Either no structural connection exists, or the path is blocked.
The forward trace stopped at gate '/I/Q'.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If there is a mismatch between the simulated value and the tied value, the tool issues one of the following error messages:

```
Mismatch between permissible tied value (1) declared by attribute
 'connection_rule_option' for DataInPort 'P' on ICL-attributed design
 instance 'I', module 'M' and the observed simulated value 0.

Mismatch between permissible tied value (0) declared by attribute
 'connection_rule_option' for DataInPort 'P' on ICL-attributed design
 instance 'I', module 'M' and the observed simulated value 1.
```

Where:

'P' is the name of the DataInPort, 'Q' is the name of the DataOutPort, 'I' is the instance name of the ICL-attributed start instance, 'M' is the module name of the ICL-attributed start instance. Tessent Visualizer can then highlight the starting instance 'I' and pin 'P' and the last gates that was reached during the tracing process.

I3

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the port functions of an identified connection between an ICL-attributed port on an ICL-attributed module match the port function connection rules.

If the port functions do not match, the tool issues the following error message:

```
The connection between ScanOutPort 'SO' on ICL-attributed design instance
 'I1', module 'M1' and ClockPort 'SI' on ICL-attributed design instance
 'I2', module 'M2' violates the ICL port function connection rules.

An instance ClockPort is only allowed to be driven by the following
 sources: ClockPort on the parent module or ToClockPort reference from an
 instance within the parent module. Where recognized values of the
 function_modifier attribute are used to alter ICL port semantics they
 must be applied consistently.
```

Where:

'I1' and 'I2' are the instance names of the ICL-attributed instances, 'M1' and 'M2' are the module names of the ICL-attributed instances, 'SO' is the name of the ScanOutPort on ICL-attributed design instance I1, 'SI' is the name of the ClockPort on ICL-attributed design instance 'I2'.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The port functions of the top-level module ports are inferred from the identified connections according to Table 10-12 and are attached as port function attributes on the top level module ports.

Table 10-12. Port Connections

Start ICL-attributed Instance Port Function	Inferred Top Module Port Function
scan_in	scan_in
shift_en	shift_en
capture_en	capture_en
update_en	update_en
data_in	data_in
select	select
address	data_in
write_en	data_in
read_en	data_in
scan_out	scan_out
data_out	data_out
to_shift_en	to_shift_en
to_capture_en	to_capture_en
to_update_en	to_update_en
to_select	to_select
to_ir_select	to_select
to_reset	to_reset
to_clock	to_clock

I4

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

This rule checks if an inferred port function and polarity for the primary input/output port is consistent for all found connections.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For example, a top-level module input port cannot be both a scan_in signal and a tck clock; it must be one or the other. If the inferred port function of port ‘SI’ is ‘scan_in’ due to an identified connection to port ‘P1’ and the inferred port function of port ‘SI’ is also ‘TCK’ due to another identified connection to port ‘P1’, the tool issues this error message.

```
Inconsistent port functions found for primary port /TCK.
Port function ScanInPort inferred due to a connection started at port 'SI'
 on ICL-attributed design instance 'I1', module 'M1' is not consistent
 with port function TCKPort inferred due to a connection started at port
 'TCK' on ICL-attributed design instance 'I2', module 'M2'.
```

Where:

‘SI’ is the ICL-attributed start port of the first connection, ‘I1’ is the instance name of the ICL-attributed start instance of the first connection, ‘M1’ is the module name of the ICL-attributed start instance of the first connection, ‘TCK’ is the inferred port function of the primary port due to the second connection, ‘I2’ is the instance name of the ICL-attributed start instance of the second connection, and ‘M2’ is the module name of the ICL-attributed start instance of the second connection.

If the polarity inferred due to the connection from the primary input ‘P’ to the module port ‘P1’ is different from the polarity inferred due to the connection from that primary input to another module port ‘P2’, the following error message is issued:

```
Inconsistent polarity for reset port functions found for primary port P.
Polarity ‘Low’/’High’ inferred due to a connection started at port ‘P1’ on
 ICL-attributed design instance ‘I1’, module ‘M1’ is not consistent with
 the polarity ‘High’/’Low’ inferred due to a connection started at port
 ‘P2’ on ICL-attributed design instance ‘I2’, module ‘M2’.
```

Where:

‘P’ is the primary port name, ‘P1’ is the ICL-attributed start port of the first connection, ‘I1’ is the instance name of the ICL-attributed start design instance of the first connection, ‘M1’ is the module name of ‘I1’, ‘P2’ is the ICL-attributed start port of the second connection, ‘I2’ is the instance name of the ICL-attributed start design instance of the second connection, ‘M2’ is the module name of ‘I2’.

I5

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error

report_drc_rules: Supported

Checks the consistency of specified pin constraints on ICL-attributed design instances.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This DRC checks that the input ports specified in the forced_high_input_port_list are actually simulated to 1 in the stable_after_setup simulation conditions. It also checks that the input ports specified in the forced_low_input_port_list are actually simulated to 0 in the stable_after_setup simulation conditions. You typically need to adjust your test_setup procedure to correct this situation.

The error message is:

```
Input constraint verification failed for input pin ‘P’ on ICL-attributed
design instance ‘I’, module ‘M’.

The corresponding design module port was specified with the attribute ‘A’
in file ‘F’ on or near line ‘L’.

Expected V1, simulated V2.
```

Where:

‘P’ is the name of the failing pin, ‘I’ is the instance name of the ICL-attributed design instance of ‘P’, ‘M’ is the module name of ‘I’, ‘A’ is the input constraint attribute name (either forced_high_input_port_list or forced_low_input_port_list) defined for ‘P’ in the ICL file ‘F’ on line number ‘L’. ‘V1’ and ‘V2’ are the simulation values on the gate corresponding to the failing pin in the flat netlist.

I6

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks all latches that have been previously identified during forward tracing of design instance pins with a scan out ICL attribute, and verifies that the backward tracing of the enable port of those latches reaches an identified primary port with TCK ICL attribute. In addition, all these latches have to be enabled by a low signal on the TCK port.

If no primary input port has been identified as TCK port, the tool issues the following error message:

```
Enable check of latch instance (LI) failed, driven by output pin (P)
on ICL-attributed design instance (I) of module (M) because no extracted
primary port with TCK attribute has been identified. (I6-1)
```

Where:

(LI) is the instance name of a latch found during forward tracing of the pin with a scan out functionality, (P) is the name of the start point of the forward tracing, (I) is the instance name of the ICL-attributed design instance of (P), and (M) is the module name of (I).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the enable port of a latch does not have a fixed simulation value and the backward tracing from the enable port does not end at the primary input port that has been identified before as TCK port, the tool issues the following error message:

```
Enable check of latch instance (LI) failed, driven by output pin (P) on
ICL-attributed design instance (I) of module (M) because it requires an
inverted connection to the extracted primary port (P2) with TCK
attribute. Either no structural connection exists, or the path is blocked.
The backward trace stopped at gate (G). (I6-2)
```

Where:

(LI) is the instance name of a latch found during forward tracing of the pin with a scan out ICL attribute, (P) is the name of the start point of the forward tracing, (I) is the instance name of the ICL-attributed design instance of (P), (M) is the module name of (I), (P2) is the name of the primary input identified as TCK port during extraction and (G) is the name and ID of the last traced gate.

Tessent Visualizer can then highlight the starting instance (LI), the identified primary port TCK (P2), the scan out pin (P) and the last gates that were reached during the backward tracing process of the latch enable port.

If the enable port of a latch has a fixed simulated value, the following error message is issued:

```
Enable check of latch instance (LI) failed, driven by output pin (P) on
 ICL-attributed design instance (I) of module (M) because it requires an
 inverted connection to the extracted primary port (P2) with TCK
 attribute. The path is blocked by a signal setting. (I6-3)
```

Due to the blocking signal settings Tessent Visualizer highlights the gates that were reached during a structural backward tracing process of the latch enable port, the starting instance (LI), the identified primary port TCK (P2) and the scan out pin.

Where:

(P) is the name of the start point of the forward tracing, (I) is the instance name of the ICL-attributed design instance of (P), and (M) is the module name of (I),

If the enable port of a latch does not have a fixed simulated value and the backward tracing ends at the primary input port that has been identified before as TCK port, but a low signal simulation does not enable the latch, the error message is:

```
Enable check of latch instance (LI) failed, driven by output pin (P) on
 ICL-attributed design instance (I) of module (M) because a low signal at
 the extracted primary pin (P2) with TCK attribute will not enable the
 latch. (I6-4)
```

Tessent Visualizer can then highlight the starting instance (LI), the identified primary port TCK (P2), the scan out pin (P) and the path from the enable port of the latch to the (P2) port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Where:

(P) is the name of the start point of the forward tracing, (I) is the instance name of the ICL-attributed design instance of (P), (M) is the module name of (I),

I7

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

This DRC only applies to design ports which have been subject to the add_clocks command. These ports are referred to as clocks. This rule checks if the properties of the clocks are consistent with the port functions and function_modifier attributes of the related extracted ICL ports.

The following is checked in detail:

- If a clock is mapped to an extracted ICL port, then this ICL port must be one of the following:
  - ClockPort
  - TCKPort
  - CaptureEnPort with attribute function_modifier = “CaptureShiftClock”
  - CaptureEnPort with attribute function_modifier = “CaptureShiftClockInv”
- If a clock is mapped to an extracted ICL port of type TCKPort, then it must not be an asynchronous clock.

The following error message is issued in case of a violation against the first requirement:

```
//  Error: The DataInPort 'dat' on ICL-attributed design instance 'inst',
//  module 'block' is connected to the primary input port 'DAT',
//  which is specified as a clock.
//  Top module clock ports are allowed to be connected only to
//  module ports defined as ClockPort, TCKPort or module ports
//  defined as CaptureEnPort when qualified with the ICL attribute
//  function_modifier = "CaptureShiftClock" or
//  function_modifier = "CaptureShiftClockInv". (I7-1)
```

The following error message is issued in case of a violation against the second requirement:

```
//  Error: The ClockPort 'clk' on ICL-attributed design instance 'inst',
//  module 'block' is connected to the primary input port 'CLK',
//  which is not specified as a pulse-always clock. (I7-3)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following error message is issued in case of a violation against the third requirement:

```
//  Error: The TCKPort 'tck' on ICL-attributed design instance 'inst',
//  module 'block' is connected to the primary input port 'TCK',
//  which is specified as an asynchronous clock. (I7-4)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL Semantic Rules (ICL Rules)

The tool performs ICL semantic rule checks upon reading (using the read_icl command) an ICL file into the tool. These checks provide a more extensive check of the ICL upon reading the file or files into Tessent IJTAG. The ICL must conform to all of the semantic rules set out for the ICL in the IEEE 1687-2014 (IJTAG) standard.

The semantic rule checks are as follows:

- **Parser** — When you read in an ICL module, the tool parses the file with a series of checks to detect for errors. The tool terminates the ICL parsing checks if any error is detected.
- **Module** — Once the complete module is available, the tool performs an extensive set of semantic checks using the totality of information contained exclusively within the module.
- **Hierarchical Module** — When the set_current_design command is run the entire ICL module instance hierarchy is traversed and each module is rechecked, taking note of any instance parameter overrides on the module instances. Also, at this time those semantic rules that check the relationships between modules, such as module instance port binding semantics, are validated. If any of these checks fail then the ICL hierarchy is not well formed and cannot be used, the set_current_design command fails.
- **ICL Primitive Model** — A further set of checks are performed, using the more detailed ICL primitive model, these are performed at the primitive model bit level and generally check interconnection semantics locally, within a single module, or on the flattened ICL primitive representation, at the global level. Again, if any of these checks fail then the ICL hierarchy is not well formed and cannot be used, the set_current_design command fails.

Note

Many of the ICL Rules have multiple messages associated with them as some rules have broad classes of failure modes

Note

The following ICL Rules are obsolete and have been removed from the documentation: ICL24, ICL43, ICL46, ICL47, ICL69, ICL72, ICL73, ICL74 and ICL131.

ICL1 3387
ICL2 3388
ICL3 3388
ICL4 3389
ICL5 3389

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL6 3389
ICL7 3389
ICL8 3390
ICL9 3390
ICL10 3390
ICL11 3391
ICL12 3391
ICL13 3391
ICL14 3391
ICL15 3392
ICL16 3392
ICL17 3392
ICL18 3392
ICL19 3393
ICL20 3393
ICL21 3393
ICL22 3394
ICL23 3394
ICL24 3394
ICL25 3394
ICL26 3395
ICL27 3395
ICL28 3395
ICL29 3395
ICL30 3396
ICL31 3396
ICL32 3396
ICL33 3396
ICL34 3397
ICL35 3397
ICL36 3397
ICL37 3398
ICL38 3398
ICL39 3398
ICL40 3399

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL41 3399
ICL42 3399
ICL44 3399
ICL45 3400
ICL48 3400
ICL49 3400
ICL50 3401
ICL51 3401
ICL52 3402
ICL53 3402
ICL54 3402
ICL55 3402
ICL56 3403
ICL57 3403
ICL58 3404
ICL59 3404
ICL60 3405
ICL61 3405
ICL62 3405
ICL63 3405
ICL64 3406
ICL65 3406
ICL66 3407
ICL67 3407
ICL68 3407
ICL70 3408
ICL71 3408
ICL75 3411
ICL76 3411
ICL77 3412
ICL78 3412
ICL79 3413
ICL80 3413
ICL81 3417
ICL82 3418

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL83 3418
ICL84 3418
ICL85 3419
ICL86 3419
ICL87 3419
ICL88 3419
ICL89 3420
ICL90 3421
ICL91 3421
ICL92 3422
ICL93 3422
ICL94 3422
ICL95 3422
ICL96 3423
ICL97 3423
ICL98 3423
ICL99 3424
ICL100 3424
ICL101 3424
ICL102 3425
ICL103 3425
ICL104 3425
ICL105 3426
ICL106 3427
ICL107 3427
ICL108 3427
ICL109 3428
ICL110 3428
ICL111 3428
ICL112 3428
ICL113 3429
ICL114 3429
ICL115 3429
ICL116 3430
ICL117 3430

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL118 3430
ICL119 3431
ICL120 3431
ICL121 3431
ICL122 3432
ICL123 3432
ICL124 3432
ICL125 3433
ICL126 3433
ICL127 3434
ICL128 3434
ICL129 3434
ICL130 3435
ICL132 3435
ICL133 3436
ICL134 3436
ICL135 3437
ICL136 3438
ICL137 3439
ICL138 3439
ICL139 3440
ICL140 3440
ICL141 3442
ICL142 3443
ICL143 3443
ICL144 3443

ICL1

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source files have no parser syntax errors.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

A failure of this semantic rule occurs if the input ICL source is not in compliance with the formal grammar for ICL. The accompanying message shows where the tool was when it encountered the incorrect syntax:

```
// Error: Incorrect character sequence. Encountered on or near line 5
// of file source.icl:
//                }
//                ^
//          ICL1
```

Note that the initial location of the incorrect syntax may actually be found prior to the location reported.

ICL2

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file has no un-terminated multi-line comments.

ICL3

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module contains non-overlapping names within the single naming region used for several types of design objects.

This rule checks the following objects:

- Ports
- Registers
- One-hot groups
- Multiplexers
- Alias description
- Logic Signals

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL4

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that an ICL module does not contain a duplicate attribute name for a single attributed declaration scope, such as a module or a register.

ICL5

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that an ICL module has no duplicate RefEnum usage in those contexts where a binding to an enumeration can be made. ICL allows only a single enumeration set to be referenced.

ICL6

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file does not contain incorrectly cased ICL keywords. ICL keywords are case sensitive.

ICL uses mixed case keywords, that must be typed exactly as specified. This check reports a failure when the reading of an ICL source file encounters a context that is expecting one of these keywords but sees an incorrectly capitalized version of a valid keyword.

ICL7

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that an identifier does not exceed the maximum character length.

The maximum length for an identifier is 1024 characters.

ICL8

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file has at least one module.

ICL9

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file has no duplicate ICL module names.

All of the modules within a single ICL source file must be unique. A source file is allowed to provide a new definition of a module that has previously been read from another ICL source file; in this case the previous definition of the module is replaced with the new module definition, if the ICL source file is read without error.

ICL10

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that an ICL module has no duplicate ICL parameter names within the definition of a single module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL11

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that an ICL module has no duplicate ScanInterface names within the definition of a single module.

ICL12

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file has no duplicate property specifications.

For example, a DataOutPort declaration may provide a specification of the source value for the port using the Source keyword. This specification may only be made once within the declaration of the port.

ICL13

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that an ActivePolarity specification of a ResetPort or ToResetPort has one of the values 0 or 1.

ICL14

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_drc_rules: Supported

Checks that the ICL source file has no duplicate instance names within a module.

ICL15

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the ICL source file contains no directly self-referential instances.

A module is not allowed to instantiate itself. Circular instance hierarchies are also not allowed, and are caught by a different rule check.

ICL16

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that each module instance within a one-hot data group declares the address value for that instance, using the AddressValue specification.

ICL17

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that within a module definition, contained in an ICL source file, each module instance does not have multiple overrides for the same instance parameter name.

ICL18

Category: ICL Semantics

Rule Check Type: Parser

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the specification of an address value for a module instance only occurs within module instances that are defined within a one-hot data group. Module instances that occur outside of a one hot data group, in the main body of a module, cannot have address value declarations, using the AddressValue specification.

ICL19

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Each scan register declaration must define the source of the scan input, using the ScanInSource specification.

ICL20

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file contains only supported ICL Constructs.

Tessent IJTAG does not yet provide support for the full ICL syntax. This rule fails when the ICL reader encounters a piece of ICL syntax that is not currently supported.

ICL21

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the data register declaration within a one hot data group provides a definition of the address value for that data register, using the AddressValue specification.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL22

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a data register declaration that contains no write data source, which implies that the register is read only, also does not provide a write enable source. A read only data register does not require and must not have a write enable control.

ICL23

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a data register with an address value definition does not also have a write enable definition. Where a data register is defined within a one-hot data group and has an address value, the address and explicit write enable control is used to control write accesses to the data register. Such an addressed data register does not require and must not define a write enable source.

ICL24

Reserved for future use.

ICL25

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that an addressed data register does not have a write data source.

An addressed data register implicitly obtains the write data source from the data inputs of the module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL26

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module definition, within an ICL source file, contains no duplicate enumeration declarations. Within an ICL module each enumeration value set must be given a unique name. The same enumeration element name can be reused, possibly with different value bindings, across multiple enumeration value sets.

ICL27

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file does not contain an enumerations using the Enum specification with a duplicate Enum item.

ICL28

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file does not contain an unterminated string constant.

ICL29

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Verifies that the iApplyEndState value, defined within an alias declaration, does not contain unknown bits.

ICL30

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks for inverted string constants.

Within a constant value expression the inversion operator (~) can only be applied to numeric constant values.

ICL31

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the constant value concatenation used to specify the value of a parameter definition in a module is comprised entirely of constant value elements that are of compatible type, either all string values or all sized numeric values.

ICL32

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a parameter definition within a module declaration does not contain a concatenation of values with an unsized numeric constant value.

ICL33

Category: ICL Semantics

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that there are no self referential parameter definitions.

A parameter definition must be made in terms of explicit immediate constant values or references to other parameter definitions. When the parameter definition hierarchy is resolved it cannot contain cyclical parameter references.

ICL34

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the size preamble for a sized number is a non-zero unsigned decimal number, where the size value is a deferred constant value or a parameter reference whose value is not verified until ICL51 is activated.

ICL35

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that only module parameters (not local parameters) are overridden in a module instance.

ICL36

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that any parameter present in an instance matches the definition in the instantiated module so that the parameter can be substituted wherever it is found in the module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Additionally, the parameter reference included in a parameter override element must match both the name and type of a parameter definition within the module being instantiated.

ICL37

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

Checks that a module definition contains at least one port definition. A module without any ports is not valid within ICL.

However, Tessent Shell does not require ICL modules with InSystemTest (IST) functionality to have any port definitions. These modules are not IEEE1687-2014 compliant and might be rejected by third-party tools.

ICL38

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies several conditions related to ports in the design.

Specifically, this rule verifies the following:

- There can be at most one WriteEnPort per module.
- There can be at most one ReadEnPort per module.

ICL39

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that all range specifications of an ICL port have the same index direction.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Within a module, the same base name can be used when defining different vector port elements. However, all of the vector port definitions sharing the same base name must use a consistent index direction (either ascending or descending).

ICL40

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module has at most one definition of a port using a scalar only name. If multiple bits are required, they must be vector bits.

ICL41

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the range specifications of an ICL port do not overlap.

Within a module, the same base name can be used when defining different vector port elements. However, all of the vector port definitions sharing the same base name must have mutually disjoint index ranges.

ICL42

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that elements referenced within a ScanInterface are ports.

ICL44

Category: ICL Semantics

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that a scan input port is present in the module if the module declares either a scan register or a scan output port.

ICL45

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that a module with a ToIRSelectPort also has a TMSPort.

ICL48

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies the input port bindings on an instance within a module. Each bit of each input port binding must correspond to an input port bit definition on the instantiated module. Where an input port binding has a vector range constraint the underlying module port must have a port definition with a compatible definition.

ICL49

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that all module input ports that require input port bindings on an instance of a module have the bindings specified.

Table 10-13 lists the port types that require input binding.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 10-13. Port Types and Required Input Binding

Port Type
DataInPort [1]
AddressPort [1]
WriteEnPort [1]
ReadEnPort[1]
ScanInPort

Notes:

1. When the instance does not have the AddressValue property

ICL50

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that each bit of the module instance input port bindings within a module is specified only once. It is not valid to specify more than one input source binding for any bit of a module input port in a module instance.

ICL51

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the width of a sized constant value is greater than 0 (zero).

This check is similar to ICL34, however here the rule also considers the actual resolved values of any parameter references.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL52

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the size of a constant value expression or a reference to a named element within the module is compatible with the size expected at the location that the expression or reference is used.

For example a data register source value must be no wider than the width of the data register itself. This check takes into account unsized expression concatenation filling and truncation behaviors when considering the size compatibility.

ICL53

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the name referenced from a parameter reference corresponds to the name of a parameter or a local parameter within the same module.

ICL54

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the type of a parameter being dereferenced matches the type expected by the context of the point of dereference. For example, within a range constraint of a vector signal “sig[$p]” the type of the parameter p must be an unsigned integer.

ICL55

Category: ICL Semantics

Rule Check Type: Module

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the parameter reference included in a parameter override element matches both the name and type of a parameter definition within the module being instantiated.

ICL56

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that any instance output port reference matches an output port on an instance within the module.

ICL57

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that references to other ICL elements can be resolved.

Many ICL building blocks (ports, registers, muxes and so on) refer to other ICL elements as their sources. References to sources can appear as part of the specification of the following ICL constructs:

- “Source” specifications of output ports
- “Enable” specifications of output ports
- “InputPort” specifications of instances
- Selections of multiplexers
- “SelectedBy” specifications of multiplexers
- “ScanInSource” specifications of ScanRegisters
- “CaptureSource” specifications of ScanRegisters
- “WriteDataSource” specifications of DataRegisters

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- “WriteEnSource” specifications of DataRegisters
- Operands of LogicSignal specifications
- “Port” specifications of OneHotDataGroups and OneHotScanGroups
- “Port” specifications of ScanInterfaces
- Right-hand side specifications of Aliases

ICL57 verifies that the referenced sources are defined in the ICL. This check comprises four aspects:

1. The used identifier must denote a port of the same module, an ICL building block within the same module or an output port of an instance within the same module.
2. An index/range specification can only be used if the referenced element also has an index/range defined.
3. An index/range specification must not be outside of the index/range definition of the referenced element.
4. The referenced element must not be a so-called hierarchical alias. Hierarchical aliases are for use with PDL, only.

ICL58

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that for a known size target, only one unsized number is included with a group of sized numbers. For an unknown size target, no unsized number can be included in concatenation.

ICL59

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that a differential inverse clock is only an inverse of a primary clock. Additionally, the clock port name, an input port connection of a clock port name is required and must be a

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

concat_clock_signal with a matching width and, if differential, a corresponding differential source.

ICL60

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that all instances are instances of modules for which a module definition has been read. It is not valid to use an ICL module hierarchy that contains references to modules that have not been defined.

ICL61

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that attribute values are either strings or sized binary values or unsigned integers.

ICL62

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that when the iApplyEndState option is present, each bit of the resolved alias is a suitable target for an iWrite, specifically a DataInPort or a register bit.

ICL63

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies for a resolved alias with iApplyEndState value that includes register bits that each such bit has a reset value, and that the reset value matches the iApplyEndState value.

ICL64

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that connected ResetPorts and ToResetPorts have consistent ActivePolarity specifications.

ICL65

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the source and the sink of an ICL connection are compatible.

Many ICL building blocks (ports, registers, muxes and so on) refer to other ICL elements as their sources. References to sources can appear as part of the specification of the following ICL constructs:

- “Source” specifications of output ports
- “Enable” specifications of output ports
- “InputPort” specifications of instances
- Selections of multiplexers
- “SelectedBy” specifications of multiplexers
- “ScanInSource” specifications of ScanRegisters
- “CaptureSource” specifications of ScanRegisters
- “WriteDataSource” specifications of DataRegisters
- “WriteEnSource” specifications of DataRegisters

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Operands of LogicSignal specifications
- “Port” specifications of OneHotDataGroups and OneHotScanGroups

The IEEE1687 standard contains many rules for what can be used and what cannot be used as a source for the aforementioned ICL constructs. ICL65 verifies that the specifications in the ICL comply with the standard. In general, “data-like” sinks can only be driven by “data-like” sources, “scan-like” sinks can only be driven by “scan-like” sources, “clock-like” sinks can only be driven by “clock-like” sources and so on and so forth. But there are also some exceptions in which conversion from one signal type into another is possible. Examples: A clock signal can be derived from a TCK port, a reset signal can be derived from a TRST port. The DataMux construct can be used to intercept signals of types other than “data”.

ICL66

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that no component has itself as a member of its fan-in.

ICL67

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a ToIRSelectPort or IRSelectPort only connects to the select input of a 2:1 ScanMux.

ICL68

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Warning

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Verifies that within each module_def that all scan signals have a path from at least one scanInPort_name and to at least one scanOutPort_name.

ICL70

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that a scan interface is composed of the correct subset of signals along with control signals related by direction.

Specifically:

- Scan Client Interface — {ScanInPort, ScanOutPort, SelectPort, CaptureEnPort, ShiftEnPort, UpdateEnPort}
- Scan Host Interface — {ScanInPort, ScanOutPort, ToSelectPort, ToCaptureEnPort, ToShiftEnPort, ToUpdateEnPort}
- TAP Client Interface — {ScanInPort, ScanOutPort, TMSPort, TRSTPort}
- TAP Host Interface — {ScanInPort, ScanOutPort, ToTMSPort, ToTRSTPort}

Scan client, scan host, TAP client and TAP host ports may not be mixed.

ICL71

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

Verifies that the anonymous ScanInterface can be correctly assembled in an ICL module with a ScanInPort and/or a ScanOutPort but without explicit ScanInterface specifications. This check applies only to the ICL module associated with the current design.

If an ICL module has a ScanInPort and/or a ScanOutPort but does not have an explicit ScanInterface specification, then Tessent Shell creates the anonymous ScanInterface from the existing ScanInPort, ScanOutPort and the ScanInterface control ports of the ICL module. ICL71 checks if this ScanInterface creation is possible as well as unique.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Each ICL71 message starts with the following lines:

```
//  Error: ICL module 'test' contains ports of type 'ScanInPort' and ports
//  of type 'ScanOutPort', but no ScanInterface specification. However,
//  the combination of specified scan ports and ScanInterface control
//  ports does not allow an unambiguous assembling of the anonymous
//  ScanInterface.
//  The module has the following problematic ports:
```

followed by a list of affected ports and explanations of what is wrong (called “port_list” in the following). Each ICL71 message ends with the source location of the affected module:

```
//  Found on or near line 1 of file '../data/icl/test1.icl'. (ICL71-1)
```

The anonymous ScanInterface cannot have more than one ScanInPort or more than one ScanOutPort. The existence of more than one ScanInPort or more than one ScanOutPort in a module without explicit ScanInterface specification makes the anonymous ScanInterface ambiguous. In this case, port_list consists of the following:

```
//  3 ScanInPorts (si1, si2, si3), explicit ScanInterface specification
//  required.
//  2 ScanOutPorts (so1, so2), explicit ScanInterface specification
//  required.
```

The next step looks for ScanInterface control ports indicating ScanInterfaces of conflicting types. Each ScanInterface can be one of the following:

- client ScanInterface (which has one port with function ShiftEnPort and/or one or many ports with function SelectPort)
- host ScanInterface (which has one port with function ToShiftEnPort and/or one or many ports with function ToSelectPort)
- client-TAP ScanInterface (which has one port with function TMSPort)
- host-TAP ScanInteface (which has one port with function ToTMSPort)

Ports with function SelectPort can also be present in an ICL module with a ScanInterface other than “client”, and ports with function ToSelectPort can also be present in an ICL module with a ScanInterface other than “host”. Therefore, SelectPorts and ToSelectPorts are not part of the first step of the analysis of conflicting ScanInterface types. Only ports of type ShiftEnPort, ToShiftEnPort, TMSPort and ToTMSPort are considered during this analysis. If there is more than one port function out of ShiftEnPort, ToShiftEnPort, TMSPort and ToTMSPort present in the ICL module, the port_list shows the related ports:

```
//  1 ShiftEnPort (se), indicating the ScanInterface type 'client',
//  1 TMSPort (tms), indicating the ScanInterface type 'client-TAP'.
```

If there is no port with function ShiftEnPort, ToShiftEnPort, TMSPort, ToTMSPort, then the ScanInterface type must be derived from the SelectPorts and ToSelectPorts. In this situation, ICL71 checks also conflicts between ports with function SelectPort and ports with function

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ToSelectPort. If there is more than one port function out of SelectPort and ToSelectPort present in the ICL module, the port_list shows the following:

```
//  1 SelectPort (sel), indicating the ScanInterface type 'client',
//  1 ToSelectPort (tsel), indicating the ScanInterface type 'host'.
```

If there is no port with function ShiftEnPort, ToShiftEnPort, TMSPort, ToTMSPort, SelectPort or ToSelectPort, such that the ScanInterface creation is invalid whatever ScanInterface type is chosen, port_list contains the following:

```
//  none of the mandatory ScanInterface control ports.
```

The absence of a ScanInPort or a ScanOutPort also reduces the possible choices for the ScanInterface type. Client and client-TAP ScanInterfaces must have a ScanInPort as well as a ScanOutPort, while host and host-TAP ScanInterfaces need only one of them (but may have both). If there is no ScanInPort or no ScanOutPort, but the existing ScanInterface control ports only allow ScanInterfaces of type “client” or “client-TAP” instead of “host” or “host-TAP”, then the port_list contains the following hints:

```
//  one ScanInPort, no ScanOutPort. The anonymous ScanInterface must be of
//  type 'host' or 'host-TAP', none of the mandatory control ports for
//  ScanInterfaces of type 'host' or 'host-TAP', 1 TMSPort (tms),
//  indicating the ScanInterface type 'client-TAP'.
```

If there is exactly one port function out of ShiftEnPort, ToShiftEnPort, TMSPort, and ToTMSPort present in the ICL module, then ICL71 still needs to check that there is exactly one port with this function. Otherwise the ScanInterface would be ambiguous. If there is more than one port of type ShiftEnPort, port_list consists of the following:

```
//  2 ShiftEnPorts (se1, se2), the tool cannot build an anonymous client
//  ScanInterface without a unique ShiftEnPort or a unique SelectPort.
```

In case of more than one port of type ToShiftEnPort:

```
//  2 ToShiftEnPorts (tse1, tse2), the tool cannot build an anonymous host
//  ScanInterface without a unique ToShiftEnPort or a unique ToSelectPort.
```

In case of more than one port of type TMSPort:

```
//  2 TMSPorts (tms1, tms2), the tool cannot build an anonymous client-TAP
//  ScanInterface without a unique TMSPort.
```

In case of more than one port of type ToTMSPort:

```
//  2 ToTMSPorts (ttms1, ttms2), the tool cannot build an anonymous
//  host-TAP ScanInterface without a unique ToTMSPort.
```

If there are no ports with function ShiftEnPort, ToShiftEnPort, TMSPort, and ToTMSPort, the ICL module must contain at least ports with function SelectPort or ToSelectPort to enable the creation of a valid anonymous ScanInterface. The tool does not create anonymous

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ScanInterfaces with more than one SelectPort or ToSelectPort even though the standard permits this. If there are several ports with function SelectPort, port_list is as follows:

```
//  2 SelectPorts (sel1, sel2), the tool cannot build an anonymous client
//  ScanInterface without a unique ShiftEnPort or a unique SelectPort.
```

In case of more than one port of type ToSelectPort:

```
//  2 ToSelectPorts  (tsel1, tsel2), the tool cannot build an anonymous
//  host ScanInterface without a unique ToShiftEnPort or a unique
//  ToSelectPort.
```

The summary message for ICL71:

```
ICL71: #fails=1 handling=error (assembling of anonymous ScanInterface failed)
```

ICL75

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the instance output referenced from the Port specification of a one-hot scan group or one-hot data group is a port with an enable control or that is a port that is driven by a one-hot scan group or one-hot data group, respectively.

The net effect of this check is to ensure that any port reference is a reference to an element that has a tri-state driver, either directly (because the port is tri-state enabled) or indirectly (because it references a port that is driven by a one-hot group).

ICL76

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies rules about ResetValues for data storage elements.

Specifically:

- Data storage elements controlling scan mux must have ResetValue.
- Data storage elements enabling ScanOutPort must have ResetValue.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL77

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

Verifies that a ScanMux or OneHotScanGroup does not deselect a controlling ScanRegister.

Reconfigurable Scan Networks may run into deadlocks if scan registers, which control the scan path configuration, take themselves out of the active path. In such a situation, the scan path through the register cannot be activated without altering the values in the register, and the values in the register cannot be altered without activating the scan path through the register. In order to avoid such situations, ICL77 checks if there is a register which controls a ScanMux or a OneHotScanGroup and which is only accessible through a subset of the inputs of the ScanMux/ OneHotScanGroup.

However, such a configuration can be intended, therefore the severity of that rule can be lowered to “Warning”. If a scan path deadlock has happened, it can usually be resolved by a Global Reset or a Local Reset, unless the ResetValue specifications are chosen such that they establish a deadlock from the very beginning.

The ICL77 check does not deal with scan paths which would never be selected by the retargeter anyway, such as scan paths that are unconnected or tied off. It still may be possible to configure the scan network explicitly (by means of iWrite and iApply), such that a deadlock happens, for example, by selecting such an unconnected or tied scan path. ICL77 mainly aims at the prevention of accidental deadlocks which the retargeter could run into.

ICL78

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

Verifies that a ScanRegister connected to an IRSelect ScanMux is active in either DR scan loads or IR scan loads, but not in both.

The tool considers the two inputs of the IRSelect ScanMux up until the point at which they re-converge. ScanRegisters in one path cannot be in the other path.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This restriction does not apply to ScanRegisters that are driven by an IR_DR ScanInterface. Define an IR_DR ScanInterface using the function_modifier attribute set to “ir_dr” on the control ports of the ScanInterface. Such ScanRegisters are active in both IR and DR scan loads.

ICL79

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

When the iApplyEndState option is present, then multiple iApplyEndState values must not specify incompatible values for any target. If there are multiple iApplyEndState values for a single target bit then they must all be of the same value.

The ultimate source of the iApplyEndState value must come directly from a data register, scan register or primary input, there can be no other intervening logic in that path.

ICL80

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

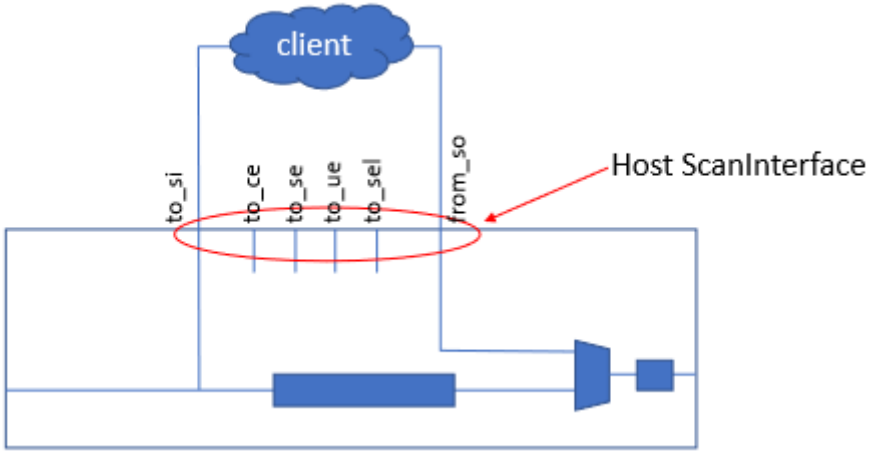
Verifies that no component has itself as a member of its fanin, that is, checks that there are no circular paths in the elaborated ICL.

This rule check considers paths across hierarchy boundaries, unlike ICL66, which only checks circular paths within one ICL module. It also checks scan paths and clock paths independently of the other elements in the ICL.

The check for a circular scan path includes special handling for a ScanInterface of type “host” or “host-TAP”, that have ScanInterfaces with port members for control signals such as ToShiftEnPort, ToSelectPort, ToTMSPort. Such a ScanInterface provides scan data to a subordinate instrument outside of the module of which the ScanInterface is a part, and receives scan data from that instrument. Therefore, the tool checks the scan data transmission path on these host ScanInterfaces from the ScanOutPort to the ScanInPort, as shown in the following figure.

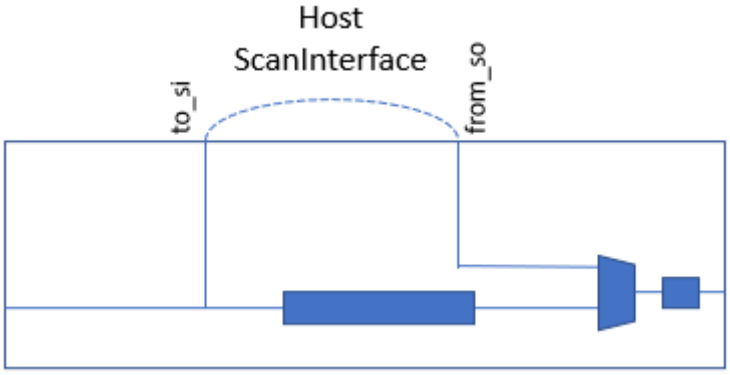
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-45. Scan Data Path to an External Subordinate Instrument



To check the correct usage of host ScanInterfaces in the current design, Tessent IJTAG configures a virtual scan path from the top module host ScanInterface's ScanOutPort to the ScanInterface's ScanInPort (if both ports exist in the ScanInterface), as shown in the following figure.

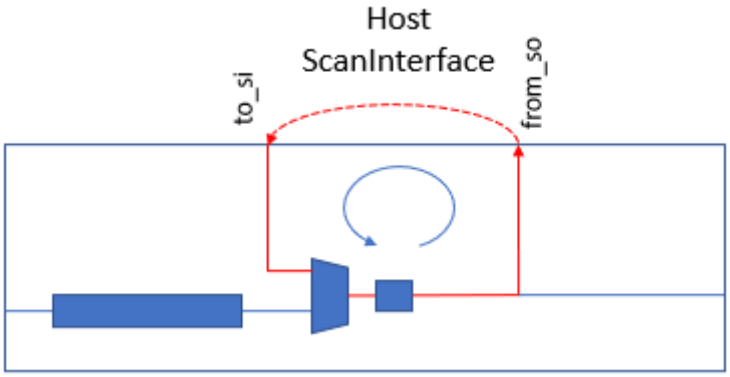
Figure 10-46. Host ScanInterface Virtual Scan Path



With this virtual connection, the ICL80 DRC also catches invalid host ScanInterface usages like the one shown in the following figure, where an invalid host ScanInterface creates a circular scan path. This enables the tool to catch issues with host ScanInterfaces early, before you instantiate the current design into a larger one in a bottom-up flow.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-47. Host ScanInterface Circular Scan Path



Whenever an ICL80 violation is related to such a virtual scan path, the error message contains an additional paragraph explaining the role of host ScanInterfaces.

Note

You can analyze ICL80 violations with Tessent Visualizer using the analyze_drc_violation command.

Examples

Example 1

The following ICL module contains a circular path from the ScanMux “sibmux” to the ScanRegister “sibreg” and back to the ScanMux “sibmux”.

```
Module chip {
  TCKPort tck;
  ScanInPort si;
  ScanOutPort so { Source sibreg; }
  CaptureEnPort ce;
  UpdateEnPort ue;
  ShiftEnPort se;
  SelectPort sel;
  ResetPort rst;

  ScanRegister sibreg {
    ScanInSource sibmux;
    CaptureSource sibreg;
    ResetValue 1'b0;
  }
  ScanMux sibmux SelectedBy sibreg {
    1'b0 : tdr[0];
    1'b1 : sibreg;
  }
  ScanRegister tdr[7:0] {
    ScanInSource si;
    CaptureSource tdr;
    ResetValue 8'b0;
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The ICL80 rule check reports the following violation:

```
//  Error: Flat ICL model contains the circular path:
//           Node 1: sibmux
//           Node 2: sibreg
//           Node 1: sibmux
//
//   Below are the locations in the ICL files where the aforementioned nodes were read from:
//           Node 1: defined in module 'chip' on or near line 17 of file 'data/chip.icl'
//           Node 2: defined in module 'chip' on or near line 12 of file 'data/chip.icl'
//   Use the 'analyze_drc_violation ICL80-1' command to debug this DRC violation in
//   Tessent Visualizer. (ICL80-1)
```

Example 2

The following ICL module contains an invalid host ScanInterface such as the one shown in Figure 10-47.

```
Module chip2 {
  TCKPort tck;
  ScanInPort si;
  ScanOutPort so; { Source sibreg; }
  CaptureEnPort ce;
  UpdateEnPort ue;
  ShiftEnPort se;
  SelectPort sel;
  ResetPort rst;

  ToTCKPort to_tck;
  ScanOutPort to_si {Source sibreg;} //error here, should be si or tdr[0]
  ScanInPort from_so;
  ToCaptureEnPort to_ce;
  ToUpdateEnPort to_ue;
  ToShiftEnPort to_se;
  ToSelectPort to_sel;
  ToResetPort to_rst;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
  ScanInterface client {
    Port si; Port so; Port ce; Port se;
    Port ue; Port sel; Port rst;
  }
  ScanInterface host {
    Port to_si; Port from_so; Port to_ce; Port to_se;
    Port to_ue; Port to_sel; Port to_rst;
  }
  ScanRegister sibreg {
    ScanInSource sibmux;
    CaptureSource sibreg;
    ResetValue 1'b0;
  }
  ScanMux sibmux SelectedBy sibreg {
    1'b0 : tdr[0];
    1'b1 : from_so;
  }
  ScanRegister tdr[7:0] {
    ScanInSource si;
    CaptureSource tdr;
    ResetValue 8'b0;
  }
}
```

The ICL80 rule check reports the following violation:

```
//  Error: Flat ICL model contains the circular path:
//         Node 1: sibmux
//         Node 2: from_so
//         Node 3: host (ScanInterface)
//         Node 4: to_si
//         Node 5: sibreg
//         Node 1: sibmux
//
//    Below are the locations in the ICL files where the aforementioned nodes were read from:
//         Node 1: defined in module 'chip2' on or near line 34 of file 'data/chip.icl'
//         Node 2: defined in module 'chip2' on or near line 14 of file 'data/chip.icl'
//         Node 3: defined in module 'chip2' on or near line 25 of file 'data/chip.icl'
//         Node 4: defined in module 'chip2' on or near line 13 of file 'data/chip.icl'
//         Node 5: defined in module 'chip2' on or near line 29 of file 'data/chip.icl'
//
//    During the identification of this circular path, the tool assumed that the reported
//    host ScanInterface will access an instrument outside of the module which the
//    ScanInterface is part of. Consequently, the tool assumed that there will be at least
//    one scan path from the ScanInterface's ScanOutPort to the ScanIinterface's ScanInPort.
//    The union of this virtual scan path (outside of the current design) and the actual scan
//    network formed the basis for triggering the circular path DRC.
//    Use the 'analyze_drc_violation ICL80-1' command to debug this DRC violation in Tessent
//    Visualizer. (ICL80-1)
```

ICL81

Category: ICL Semantics

Rule Check Type: ICL primitive model

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

The eventual driving source of a port of type {ReadEnablePort, WriteEnablePort or AddressPort} can only be a primary input or a scan register.

ICL82

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

The value associated with a clock frequency adjustment, either multiplication or division, must be a positive integer (greater than 0).

ICL83

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

The names associated with a hierarchical alias definition must all be resolvable (they must exist) to entities of an appropriate type and refer to things that have definitions compatible to their usage within the hierarchical alias definition. For example, if the reference in the alias definition includes a range constraint, then the resolved entity must be a vector object with compatible range definition.

ICL84

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that there are no unknown bits in a constant value expression and that the evaluation of the expression does not cause an integer overflow or a division by zero.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL85

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks if there is a LogicSignal expression containing an operand with unknown bits.

ICL86

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks if there is a LogicSignal expression containing operands with incompatible widths. For example,1'b0 | 2'b11.

ICL87

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks for problems with enumeration references. For example, an enum value is not defined or a register has no enum references, but uses it.

ICL88

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that there are no circular instance hierarchies in the ICL design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If a circular loop is found, the error message also indicates the location of the loop.

```
//  Error: Loop in module instantiation tree:
//            Module 'm1' has instance 'i2' of module 'm2'.
//            Module 'm3' has instance 'i1' of module 'm1'.
//            Module 'm2' has instance 'i3' of module 'm3'.
//         ICL hierarchies with circular instance loops are not supported.
Found on or near line 12 of file '../data/icl/test.icl'.
//          ICL88
```

This is a module level hierarchical check that is performed only after you run the set_current_design command.

ICL89

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

ICL connectivity extraction can be directed by the value of certain special attributes associated with the ICL ports for which a connection is being extracted, such as the 'connection_rule_option' attribute.

For more information about directing ICL extraction with ICL attributes see "Attributes of the ICL Extraction Flow" semantic in the *Tessent IJTAG User's Manual*. This semantic rule checks that the ICL port connection rule attribute has a value that is compatible with the port direction and fails if this is not the case.

The tool issues an error message if the attribute 'connection_rule_option' has a value of 'allowed_no_destination' for an input port, or one of the following values for an output port: 'allowed_no_source', 'allowed_tied', 'allowed_tied_low', 'allowed_tied_high'.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For example:

```
//  Error: ICL module 'block1' declares the port 'broken_i1' with
 attribute 'connection_rule_option' that has the invalid resolved value
 'allowed_no_destination'. For an input port the value must be one of:
 'allowed_no_source', 'allowed_tied', 'allowed_tied_high' or
 'allowed_tied_low'.
//  Error: ICL module 'block1' declares the port 'broken_o1' with
 attribute 'connection_rule_option' that has the invalid resolved value
 'allowed_no_source'. For an output port only the value
 'allowed_no_destination' is permitted.
//  Error: ICL module 'block1' declares the port 'broken_o2' with
 attribute 'connection_rule_option' that has the invalid resolved value
 'allowed_tied'. For an output port only the value
 'allowed_no_destination' is permitted.
//  Error: ICL module 'block1' declares the port 'broken_o3' with
 attribute 'connection_rule_option' that has the invalid resolved value
 'allowed_tied_low'. For an output port only the value
 'allowed_no_destination' is permitted.
//  Error: ICL module 'block1' declares the port 'broken_o4' with
 attribute 'connection_rule_option' that has the invalid resolved value
 'allowed_tied_high'. For an output port only the value
 'allowed_no_destination' is permitted.
```

ICL90

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks if there is a problem with the tessent_scan_register attribute.

A scan register with the tessent_scan_register attribute cannot specify a capture source, reset value, or default load. If a scan register with the tessent_scan_register attribute specifies a range constraint, it must be in descending order with a right index of 0. A scan register with the tessent_scan_register attribute cannot be the driver of a data out port or a logic signal.

ICL91

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks if there is a problem with the tessent_signature attribute.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the module contains a tessent_instrument_type attribute which defines the module to be an instrument of type Mentor, the tessent_signature attribute must be present and must contain a valid signature.

ICL92

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module definition, within an ICL source file, contains no duplicate access link name declarations. Within an ICL module each access link must be given a unique name.

ICL93

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module definition, within an ICL source file, contains no duplicate access link instruction declarations. Within an ICL module each access link instruction must be unique.

ICL94

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module definition, within an ICL source file, contains only one specified BSDL entity in the access link. Within an ICL module a BSDL entity can be specified only once in an access link.

ICL95

Category: ICL Semantics

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module definition, within an ICL source file, contains no duplicate access link opcode scan path declarations. Within an ICL module an opcode scan path can be specified only once in an access link.

ICL96

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module definition, within an ICL source file, contains no duplicate access link opcode active signal set declarations. Within an ICL module an opcode active signal set can be specified only once in an access link.

ICL97

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module definition, within an ICL source file, contains no access link declarations if the module does not specify the top module. Only the top ICL module can have access link declarations.

ICL98

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that a module definition, within an ICL source file, contains access link declarations that are associated with valid paths.

ICL99

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that mux selection conditions are not made redundant by previous selections in the same mux.

ICL100

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Checks that the ICL source file has no duplicate chain names within a scan interface.

ICL101

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that a ScanInterface contains ports that allow its unique classification into one of the following four ScanInterface types: client, host, client-TAP or host-TAP.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL102

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies rules about addressable data registers.

Specifically:

- Modules containing readable addressable data registers must have a ReadEnPort.
- Modules containing writable addressable data registers must have a WriteEnPort.

ICL103

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

Verifies that each ScanInPort and each ScanOutPort of the top module is referenced explicitly in at least one ScanInterface specification, if the module contains more than one ScanInPort and/or more than one ScanOutPort. This check applies only to the ICL module associated with the current design.

ICL104

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Each ‘scan client’ ScanInterface statement in a blackbox module must include a valid DefaultLoadValue definition, which reflects the length of the scan chain behind each ScanInterface after a global reset.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL105

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the ports in a ScanInterface conform to certain rules.

Specifically:

A Scan Client Interface shall have the following:

- one or many pairs of ports with function ScanInPort and ScanOutPort
- one port with function ShiftEnPort and/or one or many ports with function SelectPort
- zero or one port with function CaptureEnPort, without function_modifier attribute
- zero or one port with function UpdateEnPort, without function_modifier attribute
- optional port(s) with function CaptureEnPort and distinct function_modifier attributes
- optional port(s) with function UpdateEnPort and distinct function_modifier attributes

A Scan Host Interface shall have the following:

- one port with function ScanInPort and/or one port with function ScanOutPort
- one port with function ToShiftEnPort and/or one or many ports with function ToSelectPort
- zero or one port with function ToCaptureEnPort, without function_modifier attribute
- zero or one port with function ToUpdateEnPort, without function_modifier attribute
- optional port(s) with function ToCaptureEnPort and distinct function_modifier attributes
- optional port(s) with function ToUpdateEnPort and distinct function_modifier attributes

A TAP Client Interface shall have the following:

- one or many pairs of ports with function ScanInPort and ScanOutPort
- one port with function TMSPort
- (optional port with function TRSTPort)

A TAP Host Interface shall have the following:

- one port with function ScanInPort and/or one port with function ScanOutPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- one port with function ToTMSPort
- (optional port with function ToTRSTPort)

Note

ScanInterfaces with multiple CaptureEnPorts, UpdateEnPorts, ToCaptureEnPorts or ToUpdateEnPorts are not standard-compliant. Third-party IJTAG tools might reject the processing of ICL files with such ScanInterface specifications.

ICL106

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that a chain definition of an ICL module scan interface only contains ports of the type ScanInPort and ScanOutPorts.

ICL107

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that only scan client and TAP client interfaces can have scan chain definitions. Scan host and TAP host interfaces cannot have scan chain definitions.

ICL108

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that if a client scan interface has more than one ScanInPort/ScanOutPort pair, then each pair shall be declared in a scan chain definition.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL109

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that if a scan chain definition is present, there are no ScanInPort, ScanOutPort or DefaultLoadValue statements outside the scan chain definition.

ICL110

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that for every scan chain definition of an ICL module's scan interface, a ScanInPort/ ScanOutPort pair is defined. The ports of this chain cannot be used in any other scan chain definition in the same scan interface.

ICL111

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies rules about data storage elements.

Specifically:

- Data storage elements controlling SelectPorts of ScanInterfaces must have ResetValue.
- Data storage elements controlling ToSelectPorts of ScanInterfaces must have ResetValue.

ICL112

Category: ICL Semantics

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that no binary digits of value 1 are truncated from a sized constant number.

If the number of binary digits of a number's binary representation is larger than the number of binary digits defined by the size constant, then the left-most digits are truncated until it matches the size constant. Truncating a digit of value 1 generates an error.

ICL113

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies the mutual exclusivity of certain items in a ToClockPort specification.

The use of "Period" is mutually exclusive with the usage of "DifferentialInvOf", "Source", "FreqMultiplier" and "FreqDivider".

The use of "DifferentialInvOf" is mutually exclusive with the usage of "Period", "Source", "FreqMultiplier" and "FreqDivider".

ICL114

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that ports of type AddressPort, ReadEnPort and WriteEnPort are not explicitly connected in an instantiation with an AddressValue specification.

ICL115

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that specialized DataMuxes used to gate or suppress control signals have the right inputs.

- If an ICL DataMux suppresses a CaptureEn signal or an UpdateEn Signal, this mux must have exactly one input of this particular signal type (CaptureEn or UpdateEn). The other multiplexer inputs must be constant zero values.
- If an ICL DataMux gates a reset signal, a TRST signal, or a TMS signal, this mux must have exactly one input of this particular signal type (reset, TRST, or TMS). The other multiplexer inputs must be signals of the type data signal.

Refer to "ICL129" on page 3434 for additional details on using the DataMux construct to intercept reset signals, TRST signals, or TMS signals.

ICL116

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the function_modifier attribute is used consistently, if an ICL DataMux is used for the gating of an UpdateEn signal, a CaptureEn signal, a Reset signal, a TMS signal or a TRST signal.

ICL117

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that there is no DataRegister involved in the scan path configuration. There must not be any DataRegister in the input cone of the select input of a ScanMux or in the input cone of the EnableSource of a ScanOutPort.

ICL118

Category: ICL Semantics

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that there is no scan path from the ScanInPort of a Host ScanInterface to the ScanOutPort.

A host ScanInterface with both ScanInPort and ScanOutPort provides scan data through its ScanOutPort to the instruments under its control, and receives scan data through its ScanInPort from the instruments under its control. Therefore there cannot be a scan path between the ScanInPort and the ScanOutPort of a host ScanInterface inside the module containing the ScanInterface.

ICL119

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the instruments which are under control of an ICL host ScanInterface do not bypass the ScanInPort or the ScanOutPort of that ScanInterface.

If a host ScanInterface has both a ScanInPort as well as a ScanOutPort, then the instruments under its control must also use both of them. You cannot connect the instruments to the ScanInPort only and let the ScanOutPort be unused, nor can you connect the instrument(s) to the ScanOutPort only and let the ScanInPort be unused.

ICL120

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the function_modifier attribute definition of an ICL port is compatible with the function_modifier attribute definition of the source of that ICL port.

ICL121

Category: ICL Semantics

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Warning (severity cannot be changed)

report_drc_rules: Supported

Verifies that the value of the function_modifier attribute is a predefined value.

See the list of predefined function_modifier attributes for icl_port object types in "Built-In Attributes" on page 3785. For more information on attributes, see "Attributes" on page 3829.

ICL122

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the DefaultLoadValue specification and the ResetValue specification of a register match at those bits which are non-X in both values.

ICL123

Category: ICL Semantics

Rule Check Type: ICL Primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the source of certain output ports can be unambiguously implied, if it has not been explicitly specified.

If there is a ToTRSTPort without "Source" property, the module must not have more than one input port of type TRSTPort.

If there is a ToTMSPort without "Source" property, the module must not have more than one input port of type TMSPort.

If there is a ToResetPort without "Source" property, the module must not have more than one input port of type ResetPort.

ICL124

Category: ICL Semantics

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the source of certain instance input ports can be unambiguously implied, if the instance input port binding has not been explicitly specified.

If there is an instance port of type TRSTPort without input port binding, the module that contains the instance must not have more than one input port of type TRSTPort.

If there an instance port of type TMSPort without input port binding, the module that contains the instance must not have more than one input port of type TMSPort.

If there is an instance port of type ResetPort without input port binding, the module that contains the instance must not have more than one input port of type ResetPort.

ICL125

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the module contains the required input ports for the implied sources of certain module output ports.

If there is a ToTMSPort without “Source” property, then the module must also have a TMSPort. The ToTMSPort is then assumed to be connected to that unique TMSPort. ICL123 ensures there is only one TMSPort in that case.

ICL126

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the module contains the required input ports for the implied sources of certain instance input port bindings.

If there is an instance without explicit input port binding for a port of type TMSPort, then the parent module must also have a TMSPort. The instance input port of type TMSPort is then

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

assumed to be connected to that unique TMSPort of the parent module. ICL124 ensures there is only one TMSPort in that case.

ICL127

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Verifies that a module containing registers with ResetValue property has a unique reset signal.

For the unambiguous association between registers and their local reset source, there must be at most one reset signal in a module containing DataRegisters or ScanRegisters with ResetValue property. The ICL127 check considers module input ports of type ResetPort and instance output ports of type ToResetPort as a “reset signal”.

If the module does not have a reset signal, the registers with ResetValue property only react to the Global Reset.

ICL128

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that an ICL module with addressable instances and/or addressable data registers also has AddressPort specifications.

Currently, explicit input port bindings of the AddressPorts of an addressable instance are not supported (see ICL114). An addressable instance can only be instantiated in a module with AddressPort specifications, and those specifications are used to provide the address value to the addressable instance. The ultimate source of the address signal (scan register or primary input) must be declared outside of the module that contains the addressable instance.

ICL129

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the multiplexers, which are part of the control logic of reset signals, TRST signals or TMS signals, select the appropriate input after a reset is performed.

The ICL construct DataMux can be used to intercept reset signals, TRST signals and TMS signals. This is required, for example, for the implementation of a Local Reset or for the instantiation of child TAPs. Such a "specialized" DataMux has exactly one data input with a special function (reset, TRST, TMS), while the other inputs are connected to data signals providing, for example, the local reset value. The ultimate sources of the select input(s) of the DataMux must be either primary inputs or registers with ResetValue. If the ResetValues of the registers are chosen such that the DataMux does not select the input with the special function (reset, TRST, TMS), then the selected data signal must evaluate to the active state of the intercepted signal (any of 0 or 1 in case of a TMS signal) after a Global Reset.

This rule ensures that the control logic of Reset signals, TRST signals and TMS signals is always in a defined state, and it ensures that the Global Reset (the event triggered by the iReset command) always affects all parts of the ICL network. When the ICL network complies with this rule, the Local Reset circuitry cannot accidentally block the Global Reset.

ICL130

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

Verifies that there is no reference to an output port from within the ICL module.

Unlike Verilog, the IEEE 1687-2014 standard does not allow the specification of an output port as the source of a signal within the ICL module. Tessent IJTAG can be instructed to tolerate this configuration by setting the handling of this DRC to "warning", "note" or "ignore".

ICL132

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Warning

report_drc_rules: Supported

Verifies that vector port specifications with the same base name have consistent port directions.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL allows the specification of different port functions for disjoint subsets of the port's index range. However, all port specifications with the same base name must have either input port functions or output port functions.

ICL133

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Warning

report_drc_rules: Supported

Verifies that vector port specifications with the same base name form a continuous index range.

If the specification of a vector port is split into several parts, the union of the specified index ranges must be a continuous range without gaps. The DRC handling of this rule can be set to error in order to enforce strict standard compliance of the ICL module specification.

ICL134

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error

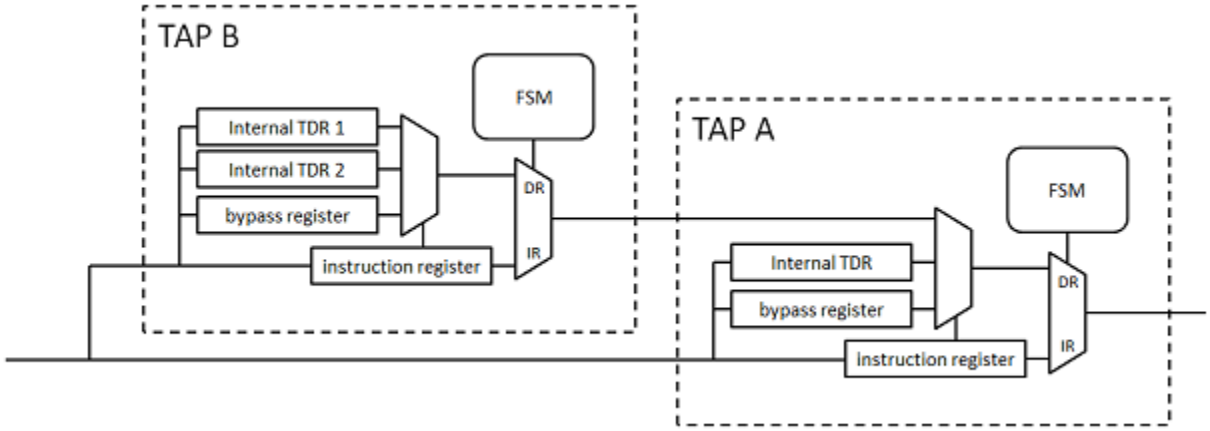
report_drc_rules: Supported

Verifies the accessibility of IR/DR multiplexers in both types of scan loads, IR scan loads as well as DR scan loads.

In a design with multiple TAP controllers, the active TAP controllers are controlled simultaneously. All state machines of the active TAP controllers are operated such that they are all in the same TAP state. Consequently, it is not possible to perform, for example, a scan load through the instruction register of an embedded TAP, while the master TAP is in Shift-DR state. This scenario is shown in Figure 10-48 . The instruction register of TAP controller "TAP B" is not accessible, because the configuration of the scan path for the access of the register requires the TAP controller "TAP A" to be in a DR state. However, the two TAP controllers cannot be in different states at the same time.

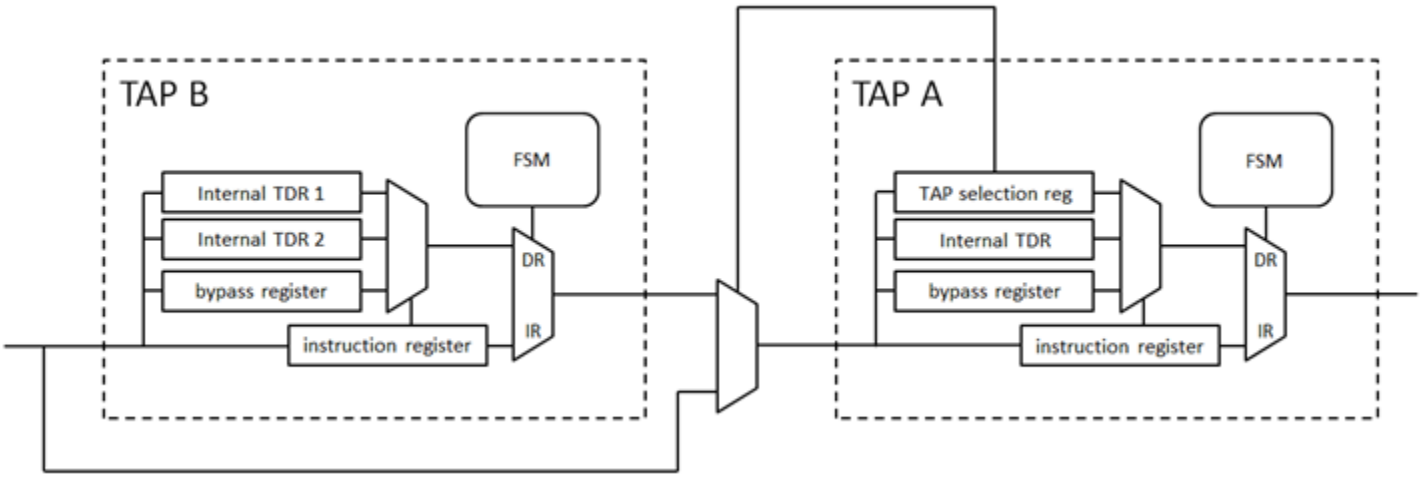
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-48. Invalid Multi-TAP Configuration



To prevent such inaccessible registers, ICL134 verifies that IR/DR multiplexers are not on a pure DR-path or on a pure IR-path of another (superordinate) TAP controller. Each IR/DR multiplexer must be accessible in DR scan loads as well as in IR scan loads. Figure 10-49 shows a valid multi-TAP configuration.

Figure 10-49. Valid Multi-TAP Configuration



ICL135

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

Verifies the correct association of ScanInterfaces with IR and DR scan paths.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The IEEE 1687 standard describes four types of ScanInterfaces that can be identified by the port function of their controlling port:

- Client ScanInterface (CaptureEnPort, ShiftEnPort, UpdateEnPort, SelectPort, ResetPort)
- Host ScanInterface (ToCaptureEnPort, ToShiftEnPort, ToUpdateEnPort, ToSelectPort, ToResetPort)
- Client-TAP ScanInterface (TMSPort, TRSTPort)
- Host-TAP ScanInterface (ToTMSPort, ToTRSTPort)

In addition, Tessent IJTAG supports client and host ScanInterfaces of the types “IR” and “IR_DR”. The controlling ports of these ScanInterfaces are marked with the function modifier “ir” or “ir_dr”, respectively.

Based on the top-level scan interfaces and the IR/DR multiplexers of the TAP controllers, ICL135 verifies the following:

- Each ScanInterface, which is only accessible during DR scan loads, must be an ordinary (that is, a non-TAP, non-IR, and non-IR_DR) ScanInterface. Ordinary client and host ScanInterfaces must only be accessible during DR scan loads. (A scan load through a client or host top-level ScanInterface is automatically considered a DR scan load.)
- Each ScanInterface, which is accessible during DR scan loads as well as during IR scan loads, must be a client-TAP ScanInterface, a host-TAP ScanInterface, an IR_DR client ScanInterface, or an IR_DR host ScanInterface. All client-TAP, host-TAP, IR_DR client, and IR_DR host ScanInterfaces must be accessible during IR scan loads as well as during DR scan loads.
- Each ScanInterface, which is only accessible during IR scan loads, must be an IR client or IR host ScanInterface. IR client and IR host ScanInterfaces must only be accessible during IR scan loads.

ICL136

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Warning

report_drc_rules: Supported

For backward compatibility reasons, Tessent IJTAG accepts some ICL constructs like LaunchEdge that are not part of the IEEE 1687-2014 standard, but which have been part of a previous draft version. This check fails if such a deprecated ICL construct is encountered. The DRC handling can be set to “error” in order to ensure strict standard compliance.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL137

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

This DRC rules out certain scenarios in which different ScanInterfaces of the same ICL module cannot be activated independently of each other. This check considers only the presence of SelectPorts in client ScanInterface specifications and the presence of ToSelectPorts in host ScanInterface specifications, it does not check if the SelectPorts can actually be controlled in such way that either the one or the other ScanInterface is active.

If two client ScanInterfaces contain SelectPorts, then the set of SelectPorts in one of the ScanInterfaces must not be equal to and must not be a subset of the set of SelectPorts in the other ScanInterface. The same holds for host ScanInterfaces and ToSelectPorts.

ICL138

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

Verifies the consistency between the IR and DR scan interfaces of an ICL module. This rule check is only applicable for client or host scan interfaces with control ports that have the function modifier “ir” or “ir_dr”. If the control ports of a ScanInterface have the function modifier “ir” (“ir_dr”), the ScanInterface is called an “IR ScanInterface” (“IR_DR ScanInterface”) for the remainder of this section.

The following rules are checked:

1. If a module has a host IR_DR ScanInterface, it must have at least one of the following client scan interfaces as well:
   - Client IR_DR ScanInterface
   - Client TAP ScanInterface
2. If a module has a host IR ScanInterface, it must have at least one of the following client scan interfaces as well:
   a. Client IR ScanInterface
   b. Client IR_DR ScanInterface and an IRSelectPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

   c. Client TAP ScanInterface

3. If a module has a host ScanInterface without function modifier, it must have at least one of the following client scan interfaces as well:
   - Client ScanInterface without function modifier
   - Client IR_DR ScanInterface and an IRSelectPort
   - Client TAP ScanInterface

ICL139

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

The ICL port attributes “tessent_pipeline_stages” and “tessent_pipeline_enable” indicate the presence and the properties of retiming pipelines.

See the description of these attributes in the table “Built-In Attributes” of object type icl_port.

A scan load/unload can only be complete and consistent if each of the retiming pipelines on the scan path is either active or inactive throughout the complete scan load. Therefore, the control logic for the activation of the pipeline should not depend on any scan register that is on the same scan path as the pipeline itself. The rule check ICL139 looks for potential risks of pipelines being on the same scan path as their enabling logic.

Note that this is a structural check that does not consider the individual possible scan path configurations and/or pipeline configurations. Special measures may ensure proper operation of the scan network even if the pipeline and the pipeline enabling logic are on the same path, but the ICL139 check is not capable of identifying those special situations. The check pessimistically errors out in all situations in which the pipeline is structurally on the same path as its enabling logic. You can downgrade the handling of ICL139 to “Ignore” using the command set_drc_handling if you are sure that the IJTAG scan network is fully operational and that the presence of a pipeline and its enable logic on the same path does not pose any problems.

ICL140

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_drc_rules: Supported

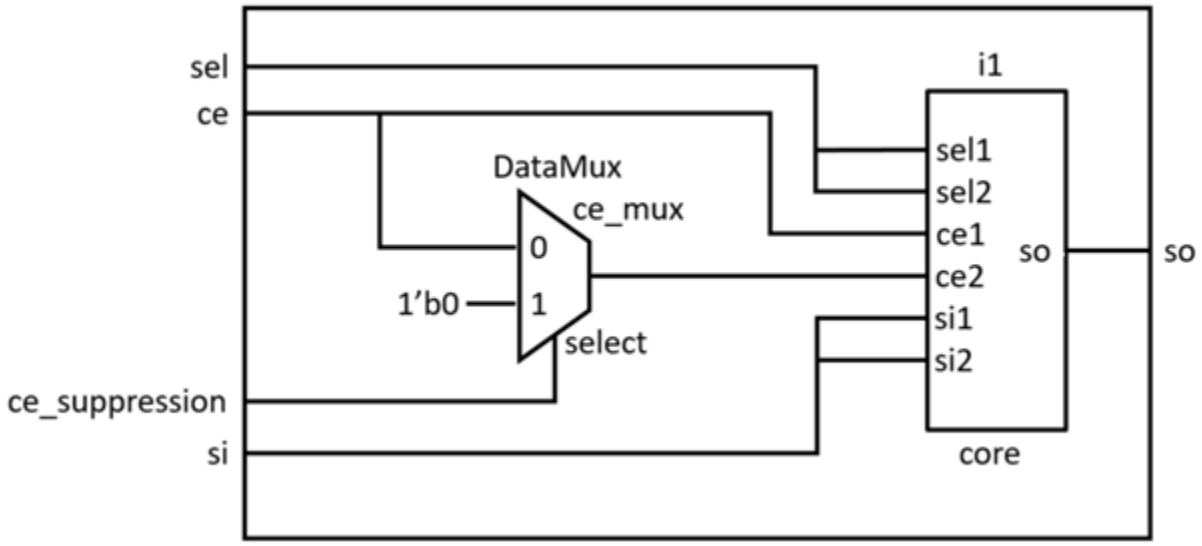
Verifies the correct association of suppressible CaptureEnPorts and UpdateEnPorts with ScanInPorts and ScanOutPorts.

Multiple ScanInterfaces of the same module can share ScanInPorts or ScanOutPorts. In this case, a number of CaptureEnPorts or UpdateEnPorts can control a single ScanInPort or ScanOutPort. The IEEE 1687 (IJTAG) standard allows for the suppression of CaptureEn and UpdateEn signals through DataMuxes. During the retargeting process, the states of the CaptureEn and UpdateEn signals along the active scan path need to be well-defined.The ICL140 DRC checks that the CaptureEnPorts and UpdateEnPorts associated with each ScanInPort and ScanOutPort are either driven by sources that are not suppressible, or that the drivers are all suppressible by the same DataMux.

Example

In the following example, the ScanInterfaces “client1” and “client2” share the ScanOutPort “so” of the module “core”. The ScanInterfaces have the CaptureEnPorts “ce1” and “ce2”. When instantiating the module “core” in the “top” module, “ce1” and “ce2” are driven by different sources. “ce2” can be suppressed with the DataMux “ce_mux” whereas “ce1” cannot be suppressed. This violates the ICL140 DRC.

Figure 10-50. Example



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module top {
  ScanInPort si; ScanOutPort so {Source i1.so; }
  SelectPort sel; CaptureEnPort ce;

  ScanInterface client { Port si; Port so; Port sel; Port ce; }

  DataInPort ce_suppression;
  DataMux ce_mux SelectedBy ce_suppression {
    1'b0 : ce;
    1'b1 : 1'b0;
  }

  Instance i1 Of core {
    InputPort si1 = si;
    InputPort sel1 = sel;
    InputPort ce1 = ce;
    InputPort si2 = si;
    InputPort sel2 = sel;
    InputPort ce2 = ce_mux;
  }
}

Module core {
  ScanInPort si1; SelectPort sel1; CaptureEnPort ce1;

  ScanInPort si2; SelectPort sel2; CaptureEnPort ce2;

  ScanOutPort so {Source […]; }

  ScanInterface client1 { Port si1; Port so; Port sel1; Port ce1; }
  ScanInterface client2 { Port si2; Port so; Port sel2; Port ce2; }

  […]
}
```

ICL141

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that the edge-detection input of the instance of an edge-triggered reset module is driven by a valid source.

The DataInPort of an ICL instance of an edge-triggered ICL module must have a valid source. A valid source is a data signal that drives the DataInPort and all inputs for the data signal originate from DataRegisters, ScanRegisters, or top-level DataInPorts.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL142

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies consistent use of the attribute tessent_transmit_enable_signal.

Use the tessent_transmit_enable_signal attribute only with "client" or "client-TAP" ScanInterface types. When this attribute is set to “on”:

- If a ScanInPort or a ScanOutPort is a member of a ScanInterface, then all ScanInterfaces the port is part of must also have this attribute set to "on".
- If a ScanInPort is a member of a ScanInterface, then it must be associated with the same ScanOutPort in all ScanInterfaces it is part of.
- If a ScanOutPort is a member of a ScanInterface, then it must be associated with the same ScanInPort in all ScanInterfaces it is part of.
- If a ScanOutPort is a member of a ScanInterface, then it must not have the 'Enable' property.

ICL143

Category: ICL Semantics

Rule Check Type: ICL Primitive Model

Contexts supported: All contexts

Default Handling: Error

report_drc_rules: Supported

Verifies that DataInOutPorts are only used for TPSP controllers.

DataInOutPorts are not part of the IEEE 1687-2014 (IJTAG) standard. Tessent IJTAG supports DataInOutPorts only to model the SPIO port of a Two-Pin Serial Port (TPSP) interface.

This DRC validates that top level DataInOutPorts have the correct TPSP function_modifier attribute ("two_pin_serial_inout_port") and that DataInOutPorts are connected to a TPSP controller.

ICL144

Category: ICL Semantics

Rule Check Type: Parser

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that a TCKPort or a ToTCKPort with a Source property is driving a ToTCKPort.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

EDT Rules (K Rules)

EDT rules only apply to designs that incorporate EDT technology. They are generally performed by Tessent TestKompress when switching from Setup system mode. The tool also performs some checks at other times such as in response to a write_edt_files command or before writing EDT files. These rule checks find EDT design definition inconsistencies.

The following subsections describe each of the EDT rules and the special handling you can set for them.

K1 3446
K2 3446
K3 3446
K4 3447
K5 3447
K6 3447
K7 3449
K8 3449
K9 3449
K10 3451
K11 3451
K12 3452
K13 3452
K14 3452
K15 3453
K16 3458
K17 3458
K18 3458
K19 3459
K20 3460
K21 3461
K22 3461
K23 3463
K24 3463
K25 3464

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

K1

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

EDT requires a circuit with scan chains. This rule check verifies that the design contains scan chains. The handling for this rule violation is error. You cannot change the handling with the set_drc_handling command.

K2

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

EDT does not support designs with more than one scan group. This rule check verifies that the design defines exactly one scan group. The handling for this rule violation is error. You cannot change the handling with the set_drc_handling command.

K3

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

This rule check verifies that the scan chain input and scan chain output pin of each scan chain are both connected to either top-level ports or to internal pins.

The following are valid ways to connect the scan chains:

- In the IP Creation Phase, all of the scan chains must be connected to primary inputs (PIs) and primary outputs (POs).
- In the Pattern Generation Phase, the scan-in and scan out pin of each chain must be connected either to a PI and PO (for an uncompressed scan chain) or to internal nodes (for a compressed scan chain driven by and observed through the EDT logic).

The handling for this rule violation is error. You cannot change the handling with the set_drc_handling command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

K4

Category: EDT

Contexts Supported: dft -edt

Default Handling: Error

report_drc_rules: Supported

The EDT decompressor provides test patterns through unidirectional outputs to the inputs of the internal scan chains. Similarly, the EDT compactor accepts test responses through unidirectional inputs from the scan chain outputs. This rule check verifies that bidirectional top-level scan chain pins do not exist. The default handling for this rule violation is error.

K5

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This rule check verifies the existence of all required EDT control and channel pins at the top level of the design. The default handling for this rule violation is error.

If K5 reports as nonexistent an EDT control or channel pin you believe exists, it is typically because the pin does not exist at the top level of the design from which the tool can control the pin, or because the pin's current name differs from the pin name the tool expected. This difference could occur, for example, if you manually edited the pin name within the netlist after the IP Creation Phase. Use the set_edt_pins command to provide the tool with the correct pin name in these cases.

Note

You can use the set_edt_pins{edt_pin} - option to explicitly tell the tool that an EDT signal is driven internally and does not have an associated top-level pin. Specifying this option prevents the tool from generating a K5 violation for that pin.

K6

Category: EDT

Scope: IP Creation Phase and Pattern Generation Phase

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

EDT pins, when shared with a design's functional pins, must be shared with a functional pin of the same type. For example, when an EDT input pin is shared with a functional pin, the functional pin must be an input. Similarly, an EDT output pin must only be shared with a functional output pin. This rule check verifies that each EDT pin that is shared with a functional pin is of a type consistent with the functional pin. Sharing with bidirectional pins is not allowed, and is covered by other design rule checks.

The K6 rule check also verifies the EDT control and channel pins have valid names. To be consistent with Tessent FastScan and to provide immediate feedback if you change the default pin names, Tessent TestKompress checks the validity of the pin names and pin directions when you enter a set_edt_pins command. Because the default EDT pin names might also conflict with names of existing pins, the tool performs all checks again when you switch from setup mode to analysis mode. The default handling for this rule violation is error.

The tool performs K6 rule checks at three different times. The items checked at each time by this rule are as follows:

These six requirements are checked immediately after the tool reads an EDT pin name. An error message results if the stated requirement is violated:

1. The pin name must not be hierarchical.
2. If the pin name is shared with a core pin, the directions must match.
3. If the pin is part of a bus, the base name for the pin must not be the same as the name of a core pin.
4. The pin name must not be the same as the base name of a core bus.
5. If the pin name matches a core bus pin name, the index must be within the same range.
6. If a pin name is a bus bit, the bus must exist in the core. Presently, bus-based EDT pins are not supported.

In addition to the preceding six requirements, the K6 rule checks the following requirements during DRC:

1. Each EDT pin must not be shared with either another EDT pin or with a restricted core pin. For information about restricted pins, refer to "EDT Control and Channel Pins" in the *Tessent TestKompress User's Manual*.
2. Two EDT pins must not share the same name.
3. An EDT output channel pin must not be shared with a bidi.

Finally, when you write out the files that implement the EDT IP using the write_edt_files command, the K6 rule results in a warning message for either of the following conditions:

1. The EDT clock pin is shared with a core bus pin.
2. An EDT pin is shared with a core pin that is empty (no fanins or fanouts).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

K7

Category: EDT

Contexts Supported: dft -edt

Default Handling: Error

report_drc_rules: Supported

EDT does not support either bidirectional or tristate scan channel output pins. This is because Tessent TestKompress includes a multiplexer between the EDT IP and the output pad when a scan channel output pin is shared with a functional output pin. This rule check verifies that neither bidirectional nor tristate scan channel output pins exist. The default handling for this rule violation is error.

K8

Category: EDT

Contexts Supported: dft -edt

Default Handling: Warning

report_drc_rules: Supported

This rule check verifies that neither bidirectional scan channel input pins nor bidirectional EDT control pins exist. This is necessary because the EDT hardware has no control over the signal direction of bidirectional pins, and they might be driven by the core logic during load_unload, resulting in contention problems. The default handling for this rule violation is warning.

K9

Category: EDT

Contexts Supported: dft -edt

Default Handling: Warning

report_drc_rules: Supported

Top-level scan chain pins cannot be shared with a design's functional pins. This is because Tessent TestKompress connects the scan chain pins to the EDT IP and they are no longer available as primary inputs or outputs in the new top level. This rule check verifies that scan chain pins are not used as functional pins. The default handling for this rule violation is warning.

The rule check assumes pin sharing in the following situations:

- **Scan Inputs** — If a scan input drives a primary output, drives more than one scan cell, or is a bidirectional pin, pin sharing is assumed.
- **Scan Outputs** — If the last scan cell of a scan chain does not drive other scan cells, and also drives exactly one primary output, pin sharing is assumed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The K9 rule check does not catch all cases of inappropriate pin sharing (the following discussion describes why this is the case). Therefore, be sure during scan insertion that you insert dedicated scan pins. This is true even if the output of the last scan cell in a scan chain is connected directly to a primary output.

In some cases, a gate-level netlist does not contain the information to enable Tessent TestKompress to determine if there is pin sharing. For example, Figure 10-51 shows a design where the last cell of a scan chain drives functional logic as well as the scan output pin. It is impossible to extract the information from the gate-level netlist about whether the pin was there before scan insertion or has been introduced for scan purposes.

Figure 10-51. Failing Identification

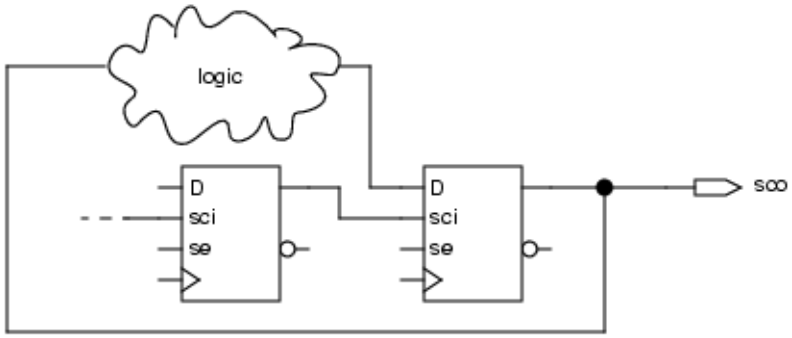
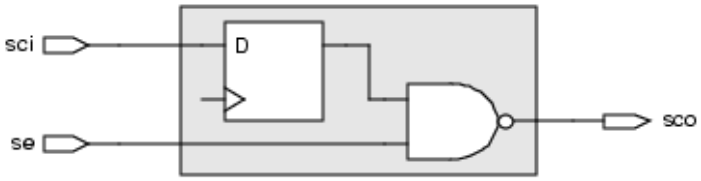


Figure 10-52 shows another case where the K9 rule check cannot determine if pin sharing occurs. The scan cell contains a logic gate that drives the scan output. By examining the netlist, it cannot be determined if the logic gate is part of the scan cell or part of functional logic. Therefore, the design rule check would not detect the sharing.

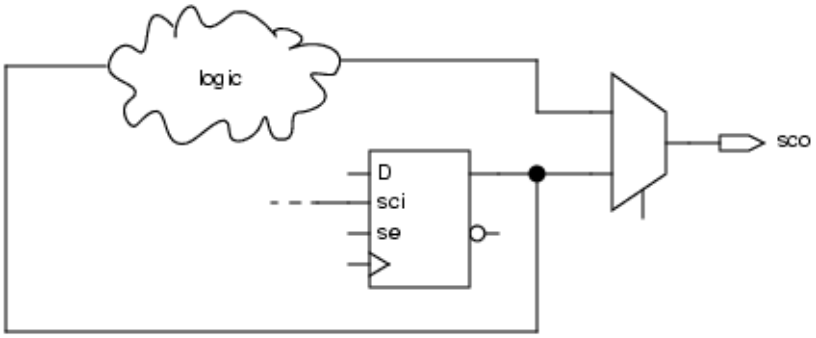
Figure 10-52. Common Cell Structure



For the same reason, the design rule check would fail also for the circuit shown in Figure 10-53.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-53. Failing Identification



K10

Category: EDT

Contexts Supported: dft -edt

Default Handling: Warning

report_drc_rules: Supported

Top-level scan chain pins should not be part of a functional bus. A DRC violation is reported if top-level scan chain pins are part of a functional bus. This DRC does not apply to internal scan pins.

Note

Top-level scan chain pins can be part of a pure scan bus.

K11

Category: EDT

Contexts Supported: dft -edt

Default Handling: Error

report_drc_rules: Supported

The EDT clock, update, and bypass pins must not be shared with any other clock or RAM control signal.

The reasons for this restriction are as follows:

- If the EDT clock or EDT update are shared with a scan clock, the scan cells would be disturbed during the load_unload procedure.
- If the EDT clock, update or bypass are shared with RAM control signals, RAM sequential patterns and multiple load patterns may not be applicable.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- If the EDT clock or EDT bypass are shared with a non-scan clock, the test coverage might decrease because the EDT clock is constrained to its off state during the capture cycle.

This rule check verifies that the EDT clock, update, and bypass pins are not shared with any other clock or RAM control signal.

K12

Category: EDT

Contexts Supported: dft -edt

Default Handling: Warning

report_drc_rules: Supported

To guarantee that the EDT IP is not disturbed during capture, the EDT clock must be constrained to its off-state during the capture cycle. This rule check verifies that the EDT clock pin does not drive any logic.

Note

If the EDT clock is shared with a functional pin, the pin constraint may result in lower test coverage.

K13

Category: EDT

Contexts Supported: dft -edt

Default Handling: Note

report_drc_rules: Supported

This rule check reports all pins that are added to the EDT top-level wrapper in order to implement the EDT hardware.

Note

The tool reports scan channel and control pins as new pins even if a scan chain pin with the same name exists in the core circuit; otherwise that scan chain pin would not have been connected to the wrapper.

K14

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_drc_rules: Supported

In order to prevent EDT from assigning values to the scan chain inputs during the capture cycle, all internal scan chain inputs are automatically constrained to X by EDT. This rule check verifies that there are no user-defined constraints on the scan chain inputs that are different from TIE-X.

K15

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

set_drc_handling: Not supported

This DRC verifies several EDT circuit characteristics.

Specifically:

1. Verifies that all scan channels have the proper top-level pins. All scan channels, except for blocks set up for input channel broadcasting or input channel sharing, require a dedicated top-level pin.
2. Verifies that all EDT control pin sharing is set up properly.
3. Verifies that the internal scan chains driven by the EDT decompressor are not sharing scan inputs.

The tool reports error messages slightly differently depending on whether the EDT block is inside a core. If an EDT block is not inside a core, the tool uses a format that only includes the block name as shown in the following example error messages.

- Verifies that the top-level input port does not connect to different control signals (like clock and update) on the same block. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_clock": (K15-1)
//        Block "blk1" EDT  clock
//        Block "blk1" EDT update
```

- Verifies that the top-level input port does not connect to different control signals (like clock and update) on different blocks. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_clock": (K15-1)
//        Block "blk1" EDT  clock
//        Block "blk2" EDT update
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Verifies that the top-level input port does not connect to the control signal and channel input of the same block. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_a": (K15-1)
//        Block "blk1" EDT  update
//        Block "blk1" EDT input channel 1
```

- Verifies that the top-level input port does not connect to the control signal and channel input of different blocks. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_a": (K15-1)
//        Block "blk1" EDT  update
//        Block "blk2" EDT input channel 1
```

- Verifies that the top-level input port does not connect to different channel inputs on the same block. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channels_in1": (K15-1)
//        Block "blk1" EDT input channel 1
//        Block "blk1" EDT input channel 2
```

- Verifies that the top-level output port does not connect to different channel outputs on the same block. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channels_out1": (K15-1)
//        Block "blk1" EDT output channel 1
//        Block "blk1" EDT output channel 2
```

- Verifies that the top-level output port does not connect to channel outputs on different blocks. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channels_out1": (K15-1)
//        Block "blk1" EDT channel output 1
//        Block "blk2" EDT channel output 1
```

- Verifies that a top-level input port does not broadcast to both compressed and uncompressed blocks. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "scan_in1" for EDT channel and
//        uncompressed scan chain: (K15-1)
//        Block "blk1" EDT input channel 1
//        Scan chain "chain1" input
```

- Verifies that two scan chains driven by the EDT decompressor do not share the same scan input pin. If this is not true, the following error message is reported:

```
// Error: Incorrect sharing of pin "/scan_input1": (K15-1)
//        Scan chain "chain1" input
//        Scan chain "chain2" input
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

To support channel sharing between identical and non-identical blocks, this DRC does the following:

- Verifies that the control channels on non-identical blocks do not share the same top-level port. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channels_in1": (K15-1)
//          Block "blk1" EDT input channel 1
//          Block "blk2" EDT input channel 1
//        When broadcasting channels to non-identical blocks, the
//        shared channels at core level cannot include any EDT
//        control registers.
```

- Verifies that if two non-identical EDT blocks have the same input channel, they also have the same “pulse_edt_before_shift_clocks” signal. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channel1" for blocks with
//        inconsistent pulse_edt_before_shift_clocks options:
//        (K15-1)
//        Block "blk1" has pulse_edt_before_shift_clocks set to "on"
//        Block "blk2" has pulse_edt_before_shift_clocks set to
//        "off"
```

- Verifies that if two input channels on two non-identical blocks share the same top-level port, they must have the same number of external pipeline stages. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channel1" for blocks with
//        different number of input pipeline stages: (K15-1)
//        Block "blk1" EDT input channel 1 has 4 pipeline stages
//        Block "blk2"EDT input channel 1 has 5 pipeline stages
```

- Verifies that if two input channels on two non-identical blocks share the same top-level port, they must have the same inversions. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channel1" for blocks of
//        shared input channels with different input channel
//        inversions: (K15-1)
//        Block "blk1" EDT input channel 1 has no inversion
//        Block "blk2" EDT input channel 1 has inversion
```

- If an EDT block is inside a core, the tool includes the parent core name of the EDT block in the error message. This example verifies that if two input channels on two non-identical blocks (either from the top-level, from the same core, or from different cores)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

share the same top-level port, they must have the same inversions. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channels_in2" for blocks of
//        shared input channels with different input channel
//        inversions: (K15-1)
//        Block "blk1" EDT input channel 2 has no inversion.
//        Block "top_core1__unwrapped__blk1" (from core instance
//        "top_core1") EDT input channel 2 has inversion.
//        Block "top_core2__unwrapped__blk1" (from core instance
//        "top_core2") EDT input channel 2 has inversion.
//        Block "top_core2__unwrapped__blk2" (from core instance
//        "top_core2") EDT input channel 2 has inversion.
```

Identical Block Checks

When checking two blocks that are identical, the K15 DRC separates the conditions into two sets:

- Conditions related to the EDT IP, which include the following:
  - The number of scan chains in each block.
  - The number of input/output channels.
  - The decompressor structure and the connections between decompressor and scan chains.
  - The compactor structure and the connections between scan chains and compactors.
  - The low power controller.
- Conditions related to the embedded logic, which includes everything outside EDT IP, such as scan chain lengths and external pipeline stages.
  - Scan chain lengths.
  - Input /output pipeline stages.
  - Input channel inversions.
  - Input channel port ordering.

If the tool encounters a K15 DRC error sharing channels among non-identical blocks, the tool first checks if the set of conditions related to EDT IP is same or not for two blocks. If they are not the same, then the tool issues the error messages noted above.

If, however, the EDT IPs are identical, then the embedding logic must be different. The intention of users was possibly to use the channel broadcasting rather than channel sharing. In this case, the tool reports the following K15 error messages:

- Two corresponding chains do not have the same lengths.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Error: Invalid sharing of port "B1_edt_channels_in1": (K15-1)
// Block "B1" EDT input channel 1
// Block "B2" EDT input channel 1
// When broadcasting channels to non-identical blocks, control
// channels cannot be shared.
// Broadcasting all input channels including control channels is
// only allowed when the EDT blocks are identical.
// In this case, the two EDT controllers are identical, but they are
// not considered as identical EDT blocks because // they are
// embedded differently in the design. One of the differences is:
// Chain #1 (“chain1”) of block “B1” has 5 scan cells while chain #1
// (“chain6”) of block “B2” has 4 scan cells.
```

- Two corresponding EDT input channels have different pipeline stages.

```
// Error: Invalid sharing of port "B1_edt_channels_in1": (K15-1)
// Block "B1" EDT input channel 1
// Block "B2" EDT input channel 1
// When broadcasting channels to non-identical blocks, control
// channels cannot be shared. Broadcasting all input channels
// including control channels is
// only allowed when the EDT blocks are identical.
// In this case, the two EDT controllers are identical, but they are
// not considered as identical EDT blocks because they are embedded
// differently in the design. One of the differences is:
// Input channel #1 of block “B1” has 2 pipeline stages while input
// channel #1 of block “B2” has 3 pipeline stages.
```

- Two corresponding EDT output channels have different pipeline stages.

```
// Error: Invalid sharing of port "B1_edt_channels_in1": (K15-1)
// Block "B1" EDT input channel 1
// Block "B2" EDT input channel 1
// When broadcasting channels to non-identical blocks, control
// channels cannot be shared.
// Broadcasting all input channels including control channels is
// only allowed when the EDT blocks are identical.
// In this case,the two EDT controllers are identical, but they are
// not considered as identical EDT blocks because they are embedded
// differently in the design. One of the differences is:
// Output channel #1 of block “B1” has 2 pipeline stages while
// output channel #1 of block “B2” has 3 pipeline stages.
```

- Two corresponding EDT input channels have different inversions.

```
// Error: Invalid sharing of port "B1_edt_channels_in1": (K15-1)
// Block "B1" EDT input channel 1
// Block "B2" EDT input channel 1
// When broadcasting channels to non-identical blocks, control
// channels cannot be shared.
// Broadcasting all input channels including control channels is
// only allowed when the EDT blocks are identical.// In this case,
// the two EDT controllers are identical, but they are not
// considered as identical EDT blocks because they are embedded
// differently in the design. One of the differences is:
// Input channel #1 of block “B1” has inversion while input channel
// #1 of block “B2” has no inversion.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Two blocks share the same set of input channel ports but not in the same order.

```
// Error: Invalid sharing of port "B1_edt_channels_in1": (K15-1)
// Block "B1" EDT input channel 1
// Block "B2" EDT input channel 1
// When broadcasting channels to non-identical blocks, control
// channels cannot be shared.
// Broadcasting all input channels including control channels is
// only allowed when the EDT blocks are identical.
// In this case, the two EDT controllers are identical, but they are
// not considered as identical EDT blocks because they are embedded
// differently in the design. One of the differences is:
// Input channel #1 of block “B1” is driven by port “A” while input
// channel #1 of block “B2” is driven by port “B”.
```

K16

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This rule check verifies that the EDT clock signal has been added as a clock using the add_clocks command.

K17

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This rule check verifies the EDT clock is constrained to its inactive state. This is required in order to avoid disturbing the EDT IP during the capture cycle.

K18

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

This rule check verifies three required properties of the test procedure file.

Specifically:

1. Analyzes the off states of the EDT control signals.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In the IP Creation Phase, Tessent TestKompress creates the EDT IP such that all the control signals are active high. This check verifies that all control signals are, in fact, active high.

Note

If a control signal (EDT clock, for example) is inverted between the chip pin and the EDT IP, and you defined the inversion using the "set_edt_pins" on page 2521 command in the IP Creation Phase, the signal at the chip level may be active-low.

2. Analyzes the EDT update control signal in the load_unload and shift procedures.

   In order to operate correctly, the EDT update signal is asserted during load_unload before the leading edge of the EDT clock signal. EDT update must be reset after the leading edge of EDT clock in load_unload and before the leading edge of EDT clock in the first shift procedure. It does not matter if EDT update is reset before or after the trailing edge of EDT clock in load_unload.

3. Analyzes the EDT clock signal in the shift procedure.

   This check verifies that the EDT clock is pulsed in the shift procedure in order to decompress vectors provided at the scan channel inputs.

K19

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This design rule check simulates the EDT decompressor netlist by applying pseudorandom patterns and compares the results with the emulated values. If a simulation-emulation mismatch occurs, the DRC performs diagnostic checks to narrow down the number of possible problem sources.

This design rule check also detects missing channel input constraints if a pipeline stage is clocked by a non-EDT clock.

How to Debug K19 Violations

For most common K19 debug tasks, such as checking for correct values on EDT control signals as well as sensitized paths from channel pins to the decompressor and from the decompressor to the scan chains during shift, it is typically sufficient to use the set_gate_report command to specify Drc_pattern reporting:

set_gate_report ***drc_pattern procedure_name***

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Then use the report_gates command to display simulated values for gates. The positions and values of mismatching monitor gates (certain gates used by the K19 rule check as reference points in its automated analysis of K19 mismatches) are reported as part of the K19 error message and provide clues to where the problem may lie.

When you need to see the specific simulation values applied in each shift cycle, you can view the simulated values for all K19-simulated gates, not just the monitor gates by setting the gate report to K19:

set_gate_report *drc_pattern* k19

This is not commonly required, as most problems are usually due to setup problems such as path sensitization and setting of control signals that are usually not pattern-specific or cycle-specific.

Note

Use "set_gate_report drc_pattern k19" reporting only when necessary. K19 reporting can slow EDT DRC run time and increase memory usage compared to Drc_pattern reporting because the tool has to log simulation data for all simulated setup and shift cycles.

For additional in-depth information on how to reduce the occurrence of K19 violations, refer to the "K19 through K22 DRC Violations" and "Understanding K19 Rule Violations" sections of the *Tessent TestKompress User's Manual*.

K20

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This design rule check identifies the number of pipeline stages within the compactors, based on simulation. It also considers channel output pipelines; reporting any discrepancy between the number of identified and specified pipeline stages between the scan chains and pins (including compactor and channel output pipelines).

Note

If the K19 rule check that verifies the operation of the decompressor fails, then the K20 rule check is not able to identify the number of pipeline stages.

If this rule check fails, you can still use diagnostic information from the K22 design rule check to help identify a problem. Design rule K22 is checked whether the K20 rule check fails or not.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Be sure scan channel output pins that are bidirectional are forced to Z at the beginning of the load_unload procedure. Otherwise, the tool is likely to issue a K20 or K22 rule violation, without indicating the reason for the violation.

K21

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

In the IP Creation Phase, this design rule check verifies that lockup cells are synthesized if there is a scan chain whose first scan cell captures data at, or after, the rising edge of the EDT clock. The rule check also verifies that lockup cells are synthesized if the output of the last scan cell changes at, or before, the rising edge of the EDT clock. If bypass logic is included, the rule verifies that lockup cells are synthesized between the last scan cell of one chain and the first scan cell of another chain when they are connected by the bypass logic, and the output of the last scan cell changes before, or at the same time, the first scan cell captures.

In the Pattern Generation Phase, this rule check verifies the first scan cell of each chain does not capture scan data at the same time the corresponding EDT decompressor output changes its signal. If the EDT compactor is pipelined, the rule check also verifies the compactor does not capture at the same time the output of the last scan cell in each chain changes. This could result in clock skew problems, as the EDT and the core circuitry are part of different clock domains. If a violation is reported, you should add lockup cells or modify the timing. The default handling for this rule violation is warning.The K21 rule check is not able to identify timing problems in the Pattern Generation Phase if the K19 rule check, which verifies the operation of the decompressor, fails.

K22

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This design rule check simulates the EDT compactor fed by pseudorandom patterns, and compares the results to the emulated values.

Two different checks are performed, as follows:

1. The tool applies a pseudorandom pattern to the compactor inputs, with all scan chains enabled.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

2. The tool tests each compactor input separately. In this case, one pseudo-random pattern is applied to the selected compactor input, and the compactor is programmed to mask all other inputs (scan chains). Decoder problems and scan chain permutations are detected by this test.

If a simulation-emulation mismatch occurs, the tool automatically performs diagnostic checks to eliminate possible sources of the mismatch. Messages regarding K22 violations usually incorporate relevant information from these diagnostic checks to help you find the source of the mismatch.

Note

Be sure scan channel output pins that are bidirectional are forced to Z at the beginning of the load_unload procedure. Otherwise, the tool is likely to issue a K20 or K22 rule violation, without indicating the reason for the violation.

How to Debug K22 Violations

For most common K22 debug tasks, such as checking for correct values on EDT control signals as well as sensitized paths from the scan chains to the compactor and from the compactor to the channel pins during shift, it is typically sufficient to use the set_gate_report command and specify Drc_pattern reporting:

set_gate_report *drc_pattern procedure_name*

Then use the report_gates command to display simulated values for gates. The positions and values of mismatching monitor gates (certain gates used by the K22 rule check as reference points in its automated analysis of K22 mismatches) are reported as part of the K22 error message and provide clues to where the problem may lie.

Note

In Tessent Visualizer, you can press Ctrl + R to execute a report_gates command on selected gates. For more information, see "How to Report Gate Data" on page 3173.

When debugging incorrect compactor mask settings for specific patterns simulated by the design rule check, you can view the simulated values for any K22-simulated gate, not just the monitor gates, by setting the gate report to K22:

set_gate_report drc_pattern k22

This is not commonly required, as most problems are usually due to setup problems such as path sensitization and setting of control signals that are usually not pattern-specific or cycle-specific.

Note

Use "set_gate_report drc_pattern k22 reporting" only when necessary. K22 reporting can slow EDT DRC run time and increase memory usage compared to Drc_pattern reporting because the tool has to log simulation data for all simulated setup and shift cycles.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For additional in-depth information on how to reduce the occurrence of K22 violations, refer to the "K19 through K22 DRC Violations" and "Understanding K22 Rule Violations" sections of the *Tessent TestKompress User's Manual*.

K23

Category: EDT

Contexts Supported: dft -edt

Default Handling: Error

report_drc_rules: Supported

This design rule check verifies (when the tool generates level-sensitive EDT hardware) that there is no scan chain whose first or last scan cell contains edge-triggered flip-flops.

Tessent TestKompress can generate EDT IP based on either edge-triggered flip-flops or level-sensitive latches. By default, it generates EDT hardware based on flip-flops, with a single clock. The edge-triggered EDT hardware can operate with a design that has scan chains based on edge-triggered scan cells or level-sensitive scan cells. Depending on the clocks used by the first and last scan cells in each chain, the tool generates lockup cells when necessary to prevent clock skew problems.

For pure LSSD designs, the tool can optionally generate EDT hardware based on level-sensitive latches with a master clock and a slave clock. But the level-sensitive EDT hardware requires a design that uses only level-sensitive scan cells, with an appropriate clock scheme for the master and slave clocks. Such a scheme ensures there is no clock skew between the EDT hardware and the first and last scan cells, so the tool does not add any lockup cells.

Note

If you use level-sensitive EDT hardware with bypass logic, you must make sure that scan chain outputs connected to scan chain inputs via bypass logic do not require lockup cells.

K24

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Note, Warning, and Error

report_drc_rules: Supported

This design rule check is performed when EDT Finder is enabled, which it is by default, or when you have enabled the set_edt_mapping command and are reusing block-level dofiles for top-level pattern creation. When EDT mapping is enabled, this rule check verifies that the clock, read control, write control, and pin constraint signals specified in the block-level dofiles are consistent with the signals specified at the top level.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

K24 does not run when core mapping is used, regardless of whether EDT Finder is on or off. When core mapping is used, the connections are automatically extracted to the top and connection mapping issues are reported by other DRCs.

The tool performs this verification by tracing connectivity between specified block-level signals (ports) and primary input or output pins. The tool verifies clock off states and pin constraint values while compensating for inversions in the path. During this rule check, all EDT signals that exist as ports on the block, but not as top-level pins, are updated to refer to the top-level input or output pins as determined by the tracing algorithm.

The K24 rule check outputs three different types of messages depending on the severity level of the results:

- **Note —** K24 outputs an informational note in cases such as when an EDT channel pin name is provided and a pin of that name exists at the top-level but not at the block-level or, in the case where the number of traced pipeline stages along the channel I/O do not match user-specified values.
- **Warning —** K24 issues a warning in cases such as when the traced top-level signal is different from the user-specified signal.
- **Error —** K24 issues an error message in cases such as when an I/O pin or clock port cannot be found.

K24 verifies the existence of the specified channels and maps block-level channels to top-level ports (unless EDT Finder has been disabled). K24 severity is set to Error (unless EDT Finder has been disabled) and you cannot override this severity setting.

For detailed information about creating a top-level dofile for the modular Tessent TestKompress flow, see the "Modular Compressed ATPG" section of the *Tessent TestKompress User's Manual*.

K25

Category: EDT

Contexts Supported: dft -edt

Default Handling: Error

report_drc_rules: Supported

All block-level EDT pins must be functionally connected in the netlist or have a connection specified during the integration session.

K25 verifies that each block-level EDT pin is either functionally connected in the netlist or has a connection specified. If no connection exists or is specified, an error is returned.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The error message is:

```
//  Error: Update pin in block "B1" is not driven in the netlist and does
//  not have a defined connection. (K25-2)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Procedure Rules (P Rules)

The test procedure file for each scan chain group is checked to ensure adherence to the format rules and accuracy of the test procedure data. All violations of procedure rules are treated as error conditions and cannot be changed—with the exception of rules P30, P31, P32, and P33, which can be ignored.

P1 3468
P2 3469
P3 3469
P4 3470
P5 3470
P6 3470
P7 3471
P8 3471
P9 3472
P10 3472
P11 3472
P12 3473
P13 3474
P14 3474
P15 3475
P16 3475
P17 3475
P18 3476
P19 3476
P20 3476
P21 3477
P22 3477
P23 3478
P24 3478
P25 3478
P26 3479
P27 3479
P28 3480
P29 3480
P30 3481

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

P31 3481
P32 3482
P33 3482
P34 3483
P35 3483
P36 3483
P37 3484
P38 3484
P39 3484
P40 3485
P41 3485
P42 3485
P43 3486
P44 3486
P45 3487
P46 3487
P47 3488
P48 3488
P49 3488
P50 3489
P51 3489
P52 3489
P53 3490
P54 3491
P55 3491
P56 3492
P58 3492
P59 3492
P60 3493
P61 3494
P62 3494
P63 3495
P64 3495
P65 3495
P66 3496

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

P70 3496
P71 3497
P72 3497
P73 3497
P74 3498
P75 3498
P76 3499
P77 3500
P78 3500
P79 3500
P80 3501
P81 3502
P82 3502
P83 3503
P84 3504
P85 3504
P86 3504
P87 3505
P88 3505
P89 3506
P90 3506
P91 3507
P92 3507
P93 3508
P94 3508
P96 3511
P97 3513

P1

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Each statement in the test procedure file must have the proper syntax. A syntax error occurs for a statement if there is an incorrect number of arguments or an incorrect ending character (“=”

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

for procedure statements and ";" for all other statements). You can correct this error condition by editing the indicated line of the test procedure file.

The error message is:

```
Error: syntax error near token (Token). (P1)
```

Where Token represents the token in the input file close to where the error occurred.

P2

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

For statements inside a procedure, the time of the statement must not be less than the time of a preceding statement. You can correct this error condition by editing the time value on the indicated line of the test procedure file.

The error message is:

```
Time value T less than preceding time value. (P2)
```

Where T is the time defined in the specified statement.

P3

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The P3 rule checks if there is any event in the load_unload procedure at the boundary of the apply shift procedure. You can correct this error by editing the time value on the indicated line of the test procedure file.

The error message is:

```
//  The following occurred at line L:
//  Error: time value not greater than preceding time value for apply
//  procedure. (P3-1)
```

L is the line number in which the failure occurred.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

P4

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

All procedures must end with an end statement. To correct this error condition, add an end statement at the indicated line of the test procedure file.

The error message is:

```
Line number L, P procedure not ended. (P4-1)
```

L is the line number in which the failure occurred, and P is the procedure name.

P5

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Procedure definitions follow the syntax, procedure proc_type [proc_name] = …. For most procedures, the procedure name is optional and it is ignored. Procedures of type seq_transparent, clock, and sub_procedure require a name. This error occurs if one of these procedures is missing its name. Also, this error occurs if the you try to define more than 32 clock or seq_transparent procedures.

```
Incorrect or missing name for procedure of type P. (P5)
```

Where P represents the procedure type that the error occurred on. Unknown procedure types do not cause a P1 violation.

P6

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may define a procedure only once in a single test procedure file. You can correct this error condition by deleting the duplicated procedure at the indicated line of the test procedure file.

The error message is:

```
Duplicate procedure P.  Original procedure from file F.  (P6)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Where F is the file that the first occurrence of the procedure P was found in.

P7

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis, patterns -ijtag

Default Handling: Warning

report_drc_rules: Not Supported

When an IJTAG port has an ijtag_function_modifier attribute value of update_clock or an ijtag_function attribute value of update_en, it must conform to certain timing requirements. If it is not possible to automatically infer the timing for this port, the tool reports a P7 warning.

The warning message is:

```
Warning: IJTAG timing problem for {iCall|iReset}: <description>. (P7-n)
```

Where description is one of:

```
Force timing (<time>) for '<signal_name>' with function modifier of
update_clock not at time of falling edge of tck (<time>).
```

or:

```
Force timing (<time>) for '<signal_name>' function of update_en not before
falling edge of tck (<time>).
```

When the tool reports a P7 warning, it adds a timing event to the appropriate timeplate to have the correct timing for the named signal. For the first warning message above, the tool adds a timing event at the falling edge of tck. For the second, the tool adds a timing event at time 0.

When the timeplate timing is an explicit timing event for <*signal_name*>, the tool reports a P60 error message.

P8

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The load_unload procedure must contain an apply shift statement. To correct this error condition, add an apply shift statement at the appropriate place in the load_unload procedure of the test procedure file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The error message is:

```
Line number L, no apply shift in load_unload procedure. (P8-1)
```

L is the line number of the end of the load_unload procedure.

P9

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The shift procedure must contain a force_sci statement. To correct this error condition, add a force_sci statement at the appropriate place in the shift procedure of the test procedure file.

The error message is:

```
Line number L, no force_sci in shift procedure. (P9-1)
```

L is the line number of the end of the shift procedure.

P10

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The shift procedure must contain a measure_sco statement. To correct this error condition, add a measure_sco statement at the appropriate place in the shift procedure of the test procedure file.

The error message is:

```
Line number L, no measure_sco in shift procedure. (P10-1)
```

L is the line number of the end of the shift procedure.

P11

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If you define a period for a timeplate, then the period must not be less than or equal to the time of the last event. To correct this error condition, increase the period for the indicated timeplate.

The error message is:

```
Period T for N is less than or equal to time of last statement. (P11)
```

T represents the period and N specifies the name of the timeplate.

How to Debug P11 Violations

Your timeplate has the following content:

```
set time scale 1.000000 ps ;

timeplate gen_tp1 =
   force_pi 0 ;
   measure_po 250 ;
   pulse clk1 500 750 ;
   period 1000;
end;
```

In this case, the tool reports a following message:

```
// The following occurred at line 7 in file test.testproc:
// Error: Period 1000 for gen_tp1 is less than or equal to time
//        of last statement. (P11-1)
```

In the timeplate gen_tp1, the clock pulse starts at 500 ps with a width of 750 ps, making the minimum period for the cycle 1250 ps. The period is defined as 1000 ps which ends before the clock pulse returns to its off state at 1250 ps, resulting in the P11 violation.

To resolve the violation, you must either increase the period or decrease the pulse.

P12

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The pin name argument for a force or pulse statement must be a valid pin name of a primary input or clock signal.

The occurrence message for a force statement is:

```
The following occured at T, incorrect pin name P. (P12-1)
```

Where T represents the token in the input file near to where the error occurred and P is the pin name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The occurrence message for a pulse statement is:

```
Incorrect pin [or path] name, P. (P12-1)
P is not a valid clock pin.
```

Where P is the pin or path name.

How to Debug P12 Violations

Edit the pin or path name on the indicated line of the test procedure file.

If the pin or path name refers to a clock signal, the clock must be defined using the add_clocks command.

P13

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

For the force statement, you may only use the force values 0, 1, X, and Z. To correct this error condition, edit the force value on the indicated line of the test procedure file.

The error message is:

```
The following occured at T, incorrect force value V. (P13-1)
```

Where T represents the token in the input file near to where the error occurred and V is the incorrect value.

P14

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may only use the force_sci statement in the shift procedure. To correct this error condition, delete the force_sci statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, force_sci only allowed in the shift procedure.  (P14-1)
```

L is the line number in which the failure occurred.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

P15

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may only use the force_sci statement once in the shift procedure. To correct this error condition, delete the force_sci statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, duplicate force_sci statement. (P15-1)
```

L is the line number in which the failure occurred.

P16

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may only use the measure_sco statement in the shift procedure. To correct this error condition, delete the measure_sco statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, measure_sco only allowed in the shift procedure. (P16-1)
```

L is the line number in which the failure occurred.

P17

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may only use the measure_sco statement once in the shift procedure. To correct this error condition, delete the measure_sco statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, duplicate measure_sco statement. (P17-1)
```

L is the line number in which the failure occurred.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

P18

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may only use the apply shift statement in the load_unload procedure. To correct this error condition, delete the apply statement on the indicated line of the test procedure file. Note that this DRC does not check apply statements used for subprocedures.

The error message is:

```
Line number L, apply only allowed in load_unload procedure.  (P18-1)
```

L is the line number in which the failure occurred.

P19

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You cannot have two apply shift statements with a value greater than 1 in the load_unload procedure.

The error message is:

```
Line number L, duplicate apply shift statement. (P19-1)
```

L is the line number in which the failure occurred.

Note

You can repeat the apply shift statement for as many serial shifts as necessary; however, you must specify 1 for the value in the additional apply shift statements.

P20

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You may only use the apply shadow_control statement in the load_unload procedure. To correct this error condition, select the apply shadow_control statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, duplicate apply shadow_control statement.  (P20-1)
```

L is the line number in which the failure occurred.

P21

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The apply shadow_control statement may only be used immediately after the apply shift statement. To correct this error condition, move the apply shadow_control statement from its current position on the indicated line of the test procedure file to the position following the apply shift statement.

The error message is:

```
Line number L, apply shift must precede apply shadow_control.  (P21-1)
```

L is the line number in which the failure occurred.

P22

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You must set the number of repetitions for the apply shadow_control statement to 1. To correct this error condition, change the repetition argument of the apply shadow_control statement on the indicated line of the test procedure file to the value of 1.

The error message is:

```
Line number L, repetitions for apply shadow_control must be  1. (P22-1)
```

L is the line number in which the failure occurred.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

P23

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may only use the apply statement for the shift and shadow_control procedures. To correct this error condition, delete the apply statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, apply procedure P not allowed. (P23-1)
```

L is the line number in which the failure occurred and P is the procedure name.

P24

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

The timing in a port specific pulse or force statement for a clock must specify the correct number of pulse statements to match the frequency multiplier. Correct this error condition by editing the timing for the clock identified.

The error message is:

```
Specified timing for frequency multiplied clock C in timeplate T only has
P pulses where R are required. (P24-1)
```

C is the clock name, T is the timeplate, P is the number of pulses and R are the required number of pulses.

P25

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

You may only use the initialize command at time 0 of the test_setup procedure. Correct this error condition by moving the statement, on the indicated line of the test procedure file, to the beginning of the test_setup procedure.

The error message is:

```
Line number L, initialize command can only be used at time 0  of
test_setup procedure. (P25-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

L is the line number in which the failure occurred.

P26

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The instance name argument for the initialize statement must correspond to at least one primitive-level latch or flip-flop gate instance. Correct this error condition by editing the statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, incorrect instance name N. (P26-1)
```

L is the line number in which the failure occurred and N is the instance name.

P27

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

This rule checks two aspects of force values on clock pins.

Specifically:

- The only allowed force values for a clock pin are 0 and 1. Correct this error condition by changing the force value of the statement on the indicated line of the test procedure file.

  The error message is:

  ```
  // Error:  Line number L, clock C may not be forced to a V. (P27-1)
  ```

  L is the line number in which the failure occurred, C is the clock pin name, and V is the incorrect value.

- The only allowed force state for a clock pin is OFF when pulsed in the timeplate. To fix a P27 error where a clock pin with pulse timing is forced to its ON state, create another timeplate that does not pulse the clock and use this timeplate for the procedure or cycle in question.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The error message is:

```
// The following occurred at line L in file F:

// Error:  Clock C which is pulsed in timeplate T may not be forced
// to on state. (P27-1)
```

F is the filename, L is the line number in which the failure occurred, C is the clock pin name, and T is the timeplate name.

- The force_capture_clock_on and force_capture_clock_off procedure file statements are used in a procedure timeplate that contains pulse statements:

```
// The following occurred at line L in file procfile_name.

// Error: Cannot use force_capture_clock_on in procedure when
// referenced timeplate timeplate has pulse statements. (P27-1)
```

P28

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The only allowed force value for a write control pin is its defined off-state value, unless it is also defined as a clock. Correct this error condition by changing the force value of the statement on the indicated line of the test procedure file.

The error message is:

```
// The following occurred at line L:
// Error: Write control W may not be forced to a V.  (P28-1)
```

L is the line number in which the failure occurred, W is the write control pin name, and V is the incorrect value.

P29

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

All clocks must be at their off-state prior to any pattern which places a clock line at an on-state. Correct this error condition by changing force statements prior to and including the indicated line of the test procedure file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Partially overlapping clocks are not supported.

The error message is:

```
// The following occurred at line L:
// Error: Clock C not at off-state prior to clock_on pattern.(P29-1)
```

L is the line number in which the failure occurred and C is the clock pin name.

P30

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

A procedure may not place a clock at its on-state at the same time it forces a non-clock pin to a value or place another clock at its off-state. Correct this error condition by changing force statements prior to and including the indicated line of the test procedure file. The rules checker ignores this condition if you set the handling to “ignore” with the set_drc_handling command.

Note

Partially overlapping clocks are not supported.

The error message is:

```
// The following occurred at line L:
// Error: Clock C cannot be forced on at this time. (P30-1)
```

L is the line number in which the failure occurred, and C is the clock pin name.

P31

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

A procedure may not force a non-clock pin to a value at the same time it forces a clock pin to a value. Correct this error condition by changing **force** statements prior to and including the indicated line of the test procedure file. The rules checker ignores this condition if you set the handling to “ignore” with the set_drc_handling command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The error message is:

```
// The following occurred at line L:
// Error: Non-clock pin N cannot be forced at this time.  (P31-1)
```

L is the line number in which the failure occurred, and N is the non-clock pin name.

P32

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

A procedure may not place a clock at its off-state at the same time it places another clock at its on-state. Correct this error condition by changing force statements prior to and including the indicated line of the test procedure file. The rules checker ignores this condition if you set the handling to “ignore” with the set_drc_handling command.

The error message is:

```
// Error: Line number L, clock pin C cannot be forced off at this  time.
// (P32-1)
```

L is the line number on which the failure occurred, and C is the clock pin name.

P33

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

When a pattern places a clock at its off-state, all clocks must be at their off-state. Correct this error condition by forcing the indicated clock to its off-state at the same time as the indicated line of the test procedure file. The rules checker ignores this condition if you set the handling to “ignore” with the set_drc_handling command.

Note

Partially overlapping clocks are not supported.

The error message is:

```
// The following occured at  line L:
// Error: Clock C not at off-state at end of clock_off pattern.(P33-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

L is the line number in which the failure occurred, C is the clock pin name, and P33-1 is the rule and violation ID number.

P34

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

At the end of all test procedures (except test_setup procedure), all clocks must be at their off-state. Correct this error condition by forcing the indicated clock to its off-state prior to the indicated line of the test procedure file.

The error message is:

```
// The following occurred at this line L:
// Error: Clock C not off at end of P procedure. (P34-1)
```

L is the line number in which the failure occurred, C is the clock pin name, and P is the procedure name.

P35

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

At the end of the master_observe and shadow_observe procedures, all constrained pins must be at their constrained states. The tools assume they are at that state prior to the procedures. Correct this error condition by forcing the indicated pin to its constrained state prior to the indicated line of the test procedure file.

The error message is:

```
Constrained pin L not at constrained value at end of P  procedure. (P35-1)
```

L is the line number in which the failure occurred, and P is the procedure name.

P36

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_drc_rules: Not Supported

The test procedure file must contain a load_unload procedure. Correct this error condition by adding a load_unload procedure to the test procedure file.

The error message is:

```
No load_unload procedure in group procedure file. (P36-1)
```

P37

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The test procedure file must contain a shift procedure. Correct this error condition by adding a shift procedure to the test procedure file.

The error message is:

```
No shift procedure in group procedure file. (P37-1)
```

P38

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

If the test procedure file contains an apply shadow_control statement, the test procedure file must contain a shadow_control procedure. Correct this error condition by adding a shadow_control procedure to the test procedure file or by deleting the apply shadow_control statement.

The error message is:

```
No shadow_control procedure in group procedure file. (P38-1)
```

P39

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the test procedure file contains a shadow_control procedure, the test procedure file must also contain an apply shadow_control statement. Correct this error condition by adding an apply shadow_control statement or by deleting the shadow_control procedure in the test procedure file.

The error message is:

```
Unused shadow_control procedure in group procedure file.  (P39-1)
```

P40

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

If you turn on the skew load option, the test procedure file must contain a skew_load procedure.

The error message is:

```
Skewed_load may not be used without a skew_load procedure.  (P40-1)
```

Correct this error by adding a skew_load procedure to the test procedure file or by turning off the skew load option with the set_skewed_load command.

P41

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The values of all pins at the beginning of the shift procedure must be the same as at the end of the shift procedure. Correct this error condition by changing force statements in the shift or load_unload procedures.

The error message is:

```
P initial value different from final value in shift  procedure. (P41-1)
```

P is the pin name.

P42

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

If there are multiple test procedure files, there can be no more than one test_setup procedure. Correct this error condition by deleting the extra test_setup procedures.

The error message is:

```
Multiple test_setup procedure defined in test procedure  files. (P42-1)
```

If there are multiple test procedure files, there can be no more than one test_end procedure. Correct this error condition by deleting the extra test_end procedures.

The error message is:

```
Multiple test_end procedure defined in test procedure  files. (P42)
```

P43

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You must place all write and read control lines at their off-state at time 0 of the load_unload procedure. Correct this error condition by adding the appropriate force statements to the test procedure file or by deleting the write or read control lines.

The error message is:

```
T control N not at off-state at time 0 of load_unload  procedure. (P43-1)
```

T is the type of control (write or read) and N is the name of the control line.

P44

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may only place the restore_pis statement at the end of a seq_transparent procedure. Correct this error condition by either removing the statement or by placing it at the end of a valid seq_transparent procedure.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The error message is:

```
Line number L, restore_pis can only be used at the end of  the
seq_transparent procedure. (P44-1)
```

L is the test procedure file line number where the error occurred, and P44-1 is the rule and violation ID number.

P45

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You may only place the condition statement at the beginning of a seq_transparent procedure. Correct this error condition by either removing the statement or by placing it at the beginning of a valid seq_transparent procedure.

The error message is:

```
Line number L, conditions can only be used at time 0 of  seq_transparent
procedures. (P45-1)
```

L is the test procedure file line number where the error occurred, and P45-1 is the rule and violation ID number.

P46

Category: Procedure

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The condition statement must identify a pin pathname that connects to the output of a scan state element. The path between the two points can only contain buffers and inverters. Correct this error condition by either removing the condition statement or by correcting the pin pathname

The error message is:

```
Invalid condition for nonscan cell N (G) during procedure P. (P46-1)
```

N is the name of the non-scan cell, G is the gate ID number, P is the seq_transparent procedure name, and P46-1 is the rule and violation ID number.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

P47

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

You cannot use the statement mentioned in the type of procedure that is named.

For example, it is not legal to have a measure statement in a load_unload procedure.

The error message is:

```
Statement S cannot be used in P procedure. (P47)
```

Where S is the statement, and P is the name of the procedure.

P48

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The set strobe_window time statement specifies a bad value for the strobe_window time.

For more information, refer to the "Set Statement" in the *Tessent Shell User's Manual*.

The error message is:

```
Negative or zero strobe window specified. (P48)
```

P49

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The set time scale statement specifies a bad time value.

For more information, refer to the "Set Statement" in the *Tessent Shell User's Manual*.

The error message is:

```
Negative or zero value specified for the time scale. (P49)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

P50

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The pulse width value for the named pulse statement in a named procedure or timeplate is zero or negative. The error message may be displayed for either timeplates or procedures, and uses either the word “timeplate” or “procedure.”

The error message is:

```
Negative or zero pulse width for S specified in [ timeplate | procedure ]
P. (P50)
```

Where S is the pulse statement, and P is the procedure name.

P51

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The named statement occurs more than once in the named timeplate.

For example, you cannot force the same pin twice in a timeplate. Neither can you pulse a clock more than once in a single timeplate.

The error message is:

```
Duplicate S statement in timeplate T. (P51)
```

Where S is the statement that occurs more than once, and T is the timeplate.

P52

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

A pulse statement was used with a pin name which is not a clock pin. This message can also appear for either timeplates or procedures.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The error message is:

```
Non-clock pin P cannot be pulsed in [ timeplate | procedure ] S. (P52)
```

Where P is the pin name, and S is the timeplate or procedure.

P53

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The named timeplate does not specify a pulse statement required by one of the procedures that use this timeplate. For example, if a procedure contains a pulse_read_clock statement, but the timeplate that the procedure uses does not contain a pulse statement for any read clock, this warning is issued. You can correct this warning by adding pulse statements for all needed clocks in the timeplate definitions.

The error message is:

```
No clock pulse timing in timeplate T which matches event E in procedure P,
two cycle clock pulses will be used. (P53)
```

Where T is the timeplate statement, E is the pulse event, and P is the procedure.

Note

A P53 warning does not cause the Vector Interfaces outputs to create incorrect patterns, just larger patterns. During pattern output, if a timeplate does not contain a needed clock pulse statement, the clock pulse is created by using two cycles and the force_pi time of the timeplate.

How to Debug P53 Violations for analyze_simulation_mismatches

When using the analyze_simulation_mismatches command for a pattern with a P53 violation, the tool issues an error and stops the DRC checking:

```
// Error: P53 violation may produce incorrect "analyze simulation
// mismatches" result.
// Suggest to fix the test procedure timeplate to avoid P53.
```

The problem of the P53 error is due to an undefined clock waveform in the used timeplate.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This violation can be seen when saving patterns as in the following example:

```
// command: write_pattern testcase_failing_patt.v -verilog -serial -all
// -rep
//  Warning: No clock pulse found in timeplate tp_shift which matches
//  event /RESETN in procedure used in patterns, two cycle clock pulse
//  will be used. (P53)
```

This example shows that the P53 error is issued when saving the Verilog testbench.

The reason for this is that the /RESETN is pulsed in some patterns, but the timeplate used for the pattern “tp_shift” does not contain a pulse time for /RESETN, which causes the pattern to expand the capture cycle into two cycles. The P53 rule changes the timing of test patterns and causes an unnecessary increase in test application.

It is recommended that you fix any P53 issues first when you want to debug the mismatch issue.

P54

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Some statement has referenced a name that is not defined. For example, using the timeplate statement in a procedure to reference a timeplate that has not yet been defined. Or, using the force statement on a pin name that the ATPG tool cannot find.

The error message is:

```
Undefined identifier N referenced by S. (P54)
```

Where N is the undefined name and S is the statement referring to N.

P55

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

This error message indicates that an old procedure type of statement is used within a cycle-based procedure, or that a cycle-based procedure statement is used in an old style procedure. For example, it would be illegal to put a break_repeat statement in a procedure that uses timeplates and cycles; put a force statement with a time value in a cycle-based procedure; or use old procedure statements in a new procedure (for example, the capture procedure).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The error message is:

```
Illegal mix of time based and cycle based statements in procedure P. (P55)
```

Where P is the procedure name.

P56

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The named timeplate or procedure is empty. If it is a procedure, it has no statements, including no timeplate references.

The error message is:

```
[ Timeplate | Procedure ] S has no statements. (P56)
```

Where S is the named timeplate or procedure.

P58

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The time specified for a break_repeat statement is too small to be useful, for example, it is equal to one or less.

The error message is:

```
Break_repeat time value T is too small in procedure P. (P58)
```

Where T is a number, and P is the name of the procedure in which T resides.

P59

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Checks for incorrect loop values on loop statements in the procedure file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The error message is:

```
//  The following occurred at line N in file <filename>:
//  Error: In <procedure_name>, loop with repetition less than 1 is not
//  allowed. (P59-1)
```

P60

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Checks whether the set tck_ratio is compatible with the timeplate that the test_setup and test_end procedure use. When a specific pin explicitly specifies timing that violates an IJTAG timing check, the tool reports a P60 error.

The error message for P60 can take one of two formats.

Error Message Format 1

This error message is as follows:

```
Error:  IJTAG error for {iCall|iReset}: <description>. (P60-n)
```

Where <*description*> is one of the following:

```
Force timing (<time>) for '<signal_name>' with function modifier of
update_clock not at time of falling edge of tck (<time>).
```

or:

```
Force timing (<time>) for '<signal_name>' function of update_en not before
falling edge of tck (<time>).
```

For certain conditions, the tool issues optional message continuations for the error. For example:

```
// A tck ratio of 1 requires pulse timing on the tck clock.

// A tck ratio of 4 or greater requires force timing on the tck clock not
// pulse timing.

// A tck ratio of 4 or greater requires force timing on the tck clock to
// occur before measure_po time.

// A tck ratio of 2 requires force timing for tck clock to occur after
// the force_pi and measure_po times.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Error Message Format 2

This error message is as follows:

```
Error:  In <proc_name>, IJTAG error for {iCall|iReset}. TCK clock
'<clock_name>' cannot be pulse_always and be used with a tck_ratio greater
than 1. (P60-n)
```

To address this error condition, do not add the TCK clock as a pulse-always clock.

P61

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported

An event on a capture_only clock in a non-capture (test_setup or load_unload) procedure has been ignored.

The warning message is:

```
//  Note:  The following occurred at line <num> in file <name>:
//  Warning:  Event on capture_only clock <name> in procedure <name> is
//  being ignored. (P61-1)
```

If this DRC is set to an error, an event on a capture_only clock in a non-capture procedure is a P61 error, and the tool is prevented from exiting setup mode.

For additional information about capture_only clocks, refer to the -capture_only argument in the add_clocks command.

P62

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

In a definition statement, the identifier N has already been used in a previous definition. For example, if there is a pin name in the design called “GROUP1” and there is an “alias” statement which tries to define an alias called “GROUP1,” the user receives the following P62 message, “alias name GROUP1 already in use as pin name.”

The error message is:

```
X name N already in use as Y. (P62)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Where X is the type of definition, and Y is the previous definition type.

P63

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

There can be only one occurrence of a pin in an alias statement. Pin P occurs more than once in the current alias statement.

The error message is:

```
Duplicate pin P in alias statement.  (P63)
```

P64

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Pin P has a direction (input, output, bidirectional) that is not compatible with statement S.

```
Direction of pin P is not compatible with S statement.  (P64)
```

For example, the following occurrence message:

```
Direction of pin Out1 is not compatible with force statement. (P64)
```

Out1 is an output pin in the netlist that is not compatible with a force statement.

P65

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

A duplicate statement S has been found in procedure P. The tool ignores the first occurrence of this statement. A duplicate statement is one that occurs at the same time and on the same pin (if applicable) as a previous statement.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For example, if a statement in a procedure causes pin A to be forced to a zero at time zero, and then a subsequent statement causes pin A to be forced to a one at time zero, the user receives a P65 message and the first force statement is ignored.

The error message is:

```
Duplicate S statement in procedure P, first statement being ignored. (P65)
```

S is the either just the event type or the event type and pin name when needed. P is the procedure type.

P66

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The P66 rule is used to check for missing statements within a load_unload_registers procedure. For example, if no Shift block is specified, or if an event statement using the '#' character is not present for each shift_assignment passed to the load_unload_registers procedure, a P66 is issued.

The error message is:

```
Error:  Procedure procedure_name is missing required statement_string
statement. (P66)
```

See "Serial Register Load and Unload for LogicBIST and ATPG" in the *Tessent Shell User's Manual* for complete information.

P70

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (can be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Source clocks defined in clock control definitions cannot be constrained.

The error message is:

```
// Warning:  Source clock <name> declared in clock control <name> cannot
// be constrained. (P70-<occurence>)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

P71

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

The pin pathname listed in an internal clock definition must be the pin pathname of an existing clock.

This rule verifies that the pin pathname specified by the CLOCK_CONTROL keyword in the clock definition defines an existing clock. Clocks are added with the add_clocks command. Internal clocks are added with the add_clocks command.

The error message is:

```
Error: Name "p1" used in clock control is not defined as a clock. (P71)
```

Where P1 is the pin pathname of the clock.

P72

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

The global control conditions and source clocks defined in multiple clock controls of the same or equivalent clocks must be exactly the same.

If the global controls defined in the multiple clock controls are not exactly the same, the following error message is reported.

```
Error: Unable to load clock control for clock "p1" due to multiple clock
controls with different global controls defined for the same clock. (P72)
```

If the source clock list defined in the multiple clock controls are not exactly the same, the following error message is reported.

```
Error: Unable to load clock control for clock "p1" due to multiple clock
controls with different source clocks defined for the same clock. (P72)
```

Where P1 is the pin pathname of the clock.

P73

Category: Procedure Rules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

A FORCE statement cannot be used to force an on value on a clock pin. P73 verifies that no clock pins are forced on in clock control definitions.

The error message is:

```
Error: Cannot force clock "p1" to its on value in clock control for clock
"p2".(P73)
```

Where both P1 and P2 are the pin pathnames of the clocks.

P74

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

When multiple clock control definitions are used, compatible conditions must exist to turn off other clock pins while the clock being controlled is pulsed.

Also, a clock pin cannot be forced off when it is defined as a source clock.

This rule check verifies that the clock control conditions defined by multiple clocks are compatible and that the clock being forced to off is not defined as a source clock in the same clock control definition.

If incompatible clock conditions exist, the following error message is reported.

```
Error: Clock “P1” is explicitly forced off in clock control for clock
 “P2”. However, there is no way to force “P1” off while pulsing “P2”.(P74)
```

If the clock that is disabled is defined as the source clock in the clock control definition of the equivalent clock, the following error message is reported.

```
Error: Clock “P1” is forced to be off explicitly in clock control for
clock “P2” while it is defined as source clock at the same time. (P74)
```

Where P1 and P2 are the pin pathnames of the clocks.

P75

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Clock control definitions cannot contain conditions that force the clock being controlled or an equivalent clock to an off state.

If a FORCE statement is used to force a controlled clock off, the following error message is reported.

```
Error: Clock "p1" cannot be forced off when defining clock control for the
same clock.(P75)
```

If a FORCE statement is used to force an equivalent clock off, the following error message is reported.

```
Error: Clock "p1" cannot be forced off when defining clock control for its
equivalent clock "p2".(P75)
```

Where P1 and P2 are the pin pathnames of the clocks.

P76

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

At least one of several components must be defined in a clock control definition.

Specifically:

- ATPG_CYCLE #, END block
- ATPG_SEQUENCE, END block
- ATPG_SEQUENCE <N> <M>, END block
- FORCE statement that forces the non-clock pin to off
- CONDITION statement

If none of the statements/procedures listed are present, one of the following warnings display:

```
Warning: The clock control for clock “P1” is unconditional for ATPG_CYCLE
"p2".(P76)

Warning: The clock control for clock “P1” is unconditional for
ATPG_SEQUENCE applied to all capture cycles.(P76)

Warning: The clock control for clock “P1” is unconditional for
ATPG_SEQUENCE "p1" "p2". (P76)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Where P1 is the pin pathname of a clock and P2 and P3 are capture cycle numbers. The tool assumes that the clock is always pulsed in the specified capture cycles when test patterns are generated.

P77

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

A non-clock pin cannot be defined as a source clock.

This rule check verifies that the pin pathname specified by the SOURCE_CLOCK keyword is a clock pin.

If the pin pathname specified by the SOURCE_CLOCK keyword is not a clock pin, the following error message is reported.

```
Error: Cannot define non-clock pin "p1" as SOURCE_CLOCK. (P77)
```

Where P1 is the pin pathname of the clock.

P78

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Multiple sequence blocks for a clock must contain local force or condition statements.

If a {ATPG_SEQUENCE, END} or {ATPG_SEQUENCE <N> <M>, END} block is declared without any local force or condition statement, the following error message is reported.

```
Error: Multiple ATPG_SEQUENCE blocks without local condition are defined
in clock control for clock “P1”.(P78)
```

Where P1 is the pin pathname of the clock.

P79

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_drc_rules: Not supported

A clock control definition containing both sequence and per-cycle blocks must provide mutually exclusive force and/or condition statements to control clock pins.

If an ATPG_SEQUENCE block is declared both with and without force or condition statements, the following error message is reported.

```
Error: ATPG_SEQUENCEs with and without local condition are defined in
clock control for clock “P1”.(P79)
```

If an ATPG_SEQUENCE block is declared without force or condition statements and an ATPG_CYCLE block declares force or condition statements, the following error message is reported.

```
Error: Both ATPG_SEQUENCE mode without local condition and per-cycle
mode with local condition are defined in clock control for clock
"p1".(P79)
```

Where P1 is the pin pathname of the clock.

P80

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Conditions defined to pulse clocks in multiple clock control block definitions must be mutually exclusive.

This rule check verifies that:

- Sequence clock control conditions that pulse the clock are mutually exclusive with the per-cycle clock control conditions that pulse the same clock.
- Sequence clock control conditions that pulse the clock are mutually exclusive of the clock control condition used to pulse the same clock in another sequence definition.

If the clock control condition to pulse a clock in sequence block is not mutually exclusive with the clock control condition for the same clock in a per-cycle block a P80 violation is reported. Depending on the violation, one of the following error messages is reported.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Error: In clock control for clock “P1”, the conditions defined in
ATPG_SEQUENCE "p2" "p3" are not mutually exclusive with the
conditions defined in ATPG_CYCLE P4.(P80)

Error: In clock control for clock “P1”, the conditions defined in
ATPG_SEQUENCE "p2" "p3" are not mutually exclusive with the
conditions defined in ATPG_SEQUENCE P4 P5.(P80)

Error: In clock control for clock “P1”, the conditions defined in
ATPG_SEQUENCE "p2" "p3" are not mutually exclusive with the conditions
defined in ATPG_SEQUENCE applied to all capture cycles.(P80)

Error: In clock control for clock “P1”, the conditions defined in
ATPG_CYCLE "p2" are not mutually exclusive with the conditions defined in
ATPG_SEQUENCE applied to all capture cycles.(P80)
```

Where P1 is pin pathname of the clock and P2, P3, P4, and P5 are capture cycles.

P81

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

There must be a disabled condition defined for sequence clock control definitions, either by applying a clock control condition or setting the source clock to the off state.

This rule check verifies that a clock controlled by a sequence block can be disabled.

If there is no way to disable a clock controlled by a sequence block, a P81 DRC violation is reported. Depending on the violation, one of the following warning messages is reported:

```
Warning: No way to disable sequence mode clock control for clock “P1”.
(P81)

Warning: No way to disable sequence mode clock control for clock “P1” by
disabling clock control condition. (P81)
```

Where P1 is the pin pathname of the clock.

P82

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

There must be a way to disable clocks defined with per-cycle clock control.

If there is no way to disable a clock defined with per-cycle clock control by either applying a condition or setting the source clock to the off state, the following warning message is reported:

```
Warning: No way to disable cycle mode clock control for “P1”.(P82)
```

If there is no way to disable a clock defined with per-cycle clock control using a condition, but it can be disabled by setting the source clock to the off state, the following warning message is reported:

```
Warning: No way to disable cycle mode clock control for clock “P1” by
disabling clock control condition.(P82)
```

Where P1 is the pin pathname of the clock.

P83

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

When using per-cycle clock control, there must be a way to disable a clock after the capture cycle.

This rule check verifies the on condition for the clock during the capture cycle is mutually exclusive with the off state condition for the same clock after the capture cycle.

If there is no source clock defined or the source clock cannot be suppressed (defined as pulse-always), one of the following warning messages is reported, and the generated test patterns do not include P1 capture cycles while the clock under control is pulsed at P1.

```
Warning: No way to pulse clock “P1” at capture cycle
"p2" while turning off the same clock after the capture cycle "p2" in
cycle mode clock control for clock “P1”. (P83)
Warning: No way to turn off clock “P1” after capture
cycle "p2" in cycle mode clock control for clock “P1”. (P83)
```

If it is possible to suppress the clock under control by turning off the source clock, one of the following warning messages is reported.

```
Warning: No way to pulse clock “P1” at capture cycle
"p2" while turning off the same clock after the capture cycle "p2" in
cycle mode clock control for clock “P1” by disabling clock control
condition. (P83)

Warning: No way to turn off clock “P1” after capture
cycle "p2" in cycle mode clock control for clock “P1” by disabling clock
control condition. (P83)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Where P1 is the pin pathname of the clock and P2 is the capture cycle.

P84

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

When both sequence mode and cycle mode clock controls are defined for a clock, there must be a way to disable the internal clock for both definitions.

This rule check verifies that a clock running in both sequential and per-cycle mode can be disabled.

If a clock running in both sequence and per-cycle cannot be disabled, the following warning message is reported.

```
Warning: No way to disable both sequence mode and cycle mode clock control
for clock "p1" simultaneously.(P84)
```

Where P1 is the pin pathname of the clock.

P85

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

When clock controls exist for two clocks not defined as equivalent, there must be a way to turn off both clocks simultaneously during a capture cycle.

If both clocks cannot be disabled simultaneously during a capture cycle, the following warning message is reported.

```
Warning: Clock control is unable to turn off clock “P1” and clock “P2”
simultaneously at capture cycle "P3". (P85)
```

Where P1 and P2 are the pin pathnames of the clocks and P3 is the capture cycle.

P86

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Default Handling: Warning

report_drc_rules: Not supported

When clock controls exist for two clocks that are not defined as equivalent and one clock is pulsed during a capture cycle, there must be a way to turn off the other clock during the same capture cycle.

If there is no source clock defined or all source clocks are defined as pulse-always, the following warning message is reported:

```
Warning: Clock control is unable to turn on clock “P1” while turning
off clock “P2” during capture cycle “P3”. (P86)
```

If it is possible to suppress the clock under control by turning off the source clock, one of the following warning messages is reported:

```
Warning: Clock control is unable to turn on clock “P1” while turning
off clock “P2” during capture cycle “P3” by disabling clock control
condition.(P86)
```

Where P1 and P2 are the pin pathnames of the clocks and P3 is the capture cycle.

P87

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

A clock control cannot be defined for a pulse-always clock.

If a clock control is defined for a pulse-always clock, the following error message is reported.

```
Warning: Cannot define clock control for pulse-always clock “P1”.(P87)
```

Where P1 is the pin pathname of the clock.

P88

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

A CONDITION or FORCE statement cannot assign a value to a non-scan state element.

If a CONDITION or FORCE statement assigns a value to a non-scan state element, the following error message is reported.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Error: Value is forced at non-PI and non-scan cell "P1" in clock control
for clock “P2”. (P88)
```

Where P1 is the pin pathname of the gate a value is assigned to and P2 is the name of the clock under control.

P89

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not supported

When you use clock control definitions, all unconstrained internal clocks must have an associated clock control definition.

Clock controls are defined in the test procedure file and can be enabled or disabled using the set_clock_controls command.

When no clock control is defined for an unconstrained internal clock, the tool displays following error message:

```
Error: Clock control is not defined for the internal clock “P1”. (P89)
```

Where P1 is the clock pin pathname.

An unconstrained internal clock does not need a clock control definition and does not cause a P89 error under any of the following circumstances:

- The internal clock is defined as pin-equivalent to other clocks and one of the equivalent clocks has clock control defined.
- The internal clock is a pulse-always or pulse-in-capture clock.
- The internal clock is an asynchronous clock (defined with “add_clocks -period”).
- The internal clock is not driving any observation points (POs or scan cells).

P90

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Internal clock controls cannot be dependent on source clocks defined by other internal clock controls.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For example, when the clock control for clock clk1 defines clk2 as source clock, the clock control for clock clk2 defines clk1 as source clock.

However, you can define the clock under control as the source clock for itself to unconditionally force itself to an off state.

When no clock control is defined for an unconstrained internal clock, the following warning message is reported.

```
Error: Clock controls for clocks “P1” and “P2” depend on each other
through source clocks.(P90)
```

Where P1 and P2 are pin pathnames of clocks.

P91

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not supported

Mutually exclusive control conditions cannot exist for different ATPG_CYCLEs of the same clock.

If mutually exclusive control conditions are defined in different ATPG_CYCLES for the same clock, the following error message is reported:

```
Warning: In clock control for clock “P1”, the conditions defined in
ATPG_CYCLEs "P2" and "P3" are mutually exclusive.
```

Where P1 is the clock pin pathname and P2 and P3 are capture cycles.

P92

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Verifies that added internal PI (primary input) signals are controlled deterministically. Specifically, verifies that every internal PI signal satisfies at least one of the following criteria:

- The internal PI has a pin constraint.
- The internal PI has an ATPG constraint.
- The internal PI is a pulse-always or pulse-in-capture clock.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- If NCP are present and enabled, the internal PI must be forced all the time in the internal mode of every enabled NCP.

If none of these criteria is true, the tool reports the following error message:

```
//Error: Internal primary input /int_clk (42) is not controlled. (P92-1)
```

The tool does not check the following cases:

- An internal PI that is defined as a scan chain input pin.
- An internal PI that is blocked by pin constraints.
- An internal PI that passes P89 (an internal clock that is controlled by clock control definitions).

Use the add_input_constraints and add_atpg_constraints commands to properly constrain an internal PI to the correct value.

Caution

Changing the handling of P92 violations with the set_drc_handling command to a severity of Warning or lower can result in simulation mismatches and tester failures.

P93

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

The clock under control cannot define itself as the source clock.

If this is the case, the tool issues the following error message:

```
// Error: Clock '<clock_pin_pathname>' cannot be defined as source clock
// for itself. (P93)
```

P94

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Verifies a consistent view of the capture window between ATPG settings and external capture procedure options or PLL cycles to avoid generating invalid patterns.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The tool only performs the P94 DRC after you run one of the following commands:

- set_external_capture_options -capture_procedure
- set_external_capture_options -PLL_cycles

When you issue these commands in SETUP mode, the tool performs the check during system mode analysis. When you issue them in Analysis mode, the tool performs the check immediately.

If you have already run one of these commands and then run the "set_pattern_type -sequential" command in Analysis mode, the tool immediately performs the P94 check again. This can result in different P94 violations.

The default handling for P94 is Error, but it can be set to Warning, Ignore, or Note. When the handling for P94 is set to Warning, the tool automatically adds TX cell constraints to the potential mismatching cells. However, if the tool attempts to add a TX cell constraint to a cell that is already constrained, it produces an error:

```
//  Error: P94 DRC cannot add cell constraint 'TX' to gate 10705 in cell 2
//  of chain c0 because the cell is already constrained to 'CX'. Please
//  delete the existing cell constraint.
```

When the tool forces all pin constraints and constant cell constraints and collects all external PIs and clocks that may reach cells directly as candidates for the P94 DRC check, it applies the following checks on each candidate:

- An external clock is pulsed in an external capture procedure.

```
//  Error: External clock '/slow_clock' (89) is pulsed in external
//  capture procedure ext_fast_cap_proc, but during pattern
//  generation, it may not be pulsed or may be pulsed for a
//  different number of cycles than specified in the external
//  capture procedure. This can lead to simulation mismatches on 7
//  cells, one of which is '/jn_scell_inst0/' (10297). (P94-1)
```

Note

You can fix this violation by blocking this clock from directly reaching a cell, or by setting P94 handling to Warning, so that the tool automatically adds TX cell constraints to mask out the affected cells.

set_drc_handling P94 warning

```
//  Warning: External clock '/slow_clock' (89) is pulsed in external
//  capture procedure ext_fast_cap_proc, but during pattern
//  generation, it may not be pulsed or may be pulsed for a
//  different number of cycles than specified in the external
//  capture procedure. This can lead to simulation mismatches on 7
//  cells, one of which is '/jn_scell_inst0/' (10297). (P94-1)
//  Added 7 TX cell constraints to cells due to P94 DRC
//  violations.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- An external PI (port) is forced to a known value in an external capture procedure, but does not have a pin constraint or has a constrained value that is inconsistent with the forced value.

```
//  Error: Primary input '/SMC' (10) is forced to 1 during external
//  capture procedure ext_pll_4_pulses, but it is not constrained
//  during pattern generation. This can lead to simulation
//  mismatches on 1 cell, which is '/sff117/' (561). (P94-1)
```

Note

You can fix this violation by removing the force event from the external capture procedure, by adding a consistent pin constraint in ATPG, or by setting P94 to Warning, so that the tool automatically adds TX cell constraints to mask the affected cells.

- An external clock is NOT pulsed in the external capture procedure, and is NOT constrained to Off state in ATPG.

```
//  Warning: External clock '/CLK1' (1) is not pulsed in external
//  capture procedure ext_cap_proc, but it may be pulsed during
//  pattern generation. This can lead to simulation mismatches on 10
//  cells, one of which is '/sff4_pulse_inter/' (369). (P94-1)
```

Note

You can fix this violation by constraining the clock to off state in ATPG.

- An external clock is pulsed while -pll_cycles is set.

```
//  Warning: External clock '/RCLK2' (2) is pulsed during capture
//  when 'set_external_capture_options -pll_cycles' is used,
//  but during pattern generation, it may not be pulsed or may be
//  pulsed for a different number of cycles than specified by the
//  -pll_cycles switch. This can lead to simulation mismatches on
//  7 cells, one of which is '/dff07/' (248). (P94-1)
```

- An external clock is NOT pulsed while -pll_cycles is set.

```
//  Warning: External clock '/PHYADR1' (1) is not pulsed during
//  capture when 'set_external_capture_options -pll_cycles' is
//  used, but it may be pulsed during pattern generation. This can
//  lead to simulation mismatches on 1 cell, which is
//  '/name_reg1/' (612). (P94-1)
//  Added 1 TX cell constraints to cells due to P94 DRC
//  violations.
```

The summary message issued by "report_drc_rules -summary" command for P94:

```
P94: #fails=0 handling=error (inconsistent view between ATPG settings and
external capture options)
```

To debug P94, use the "set_gate_report drc stable_capture" command to view the values that are constant throughout the capture process for the affected capture.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

P96

Category: Procedure Rules

Context Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Verifies that, when clock restriction is set to on or domain_clock, cell and pin constraints do not force incompatible clocks to be pulsed or to be restrained from pulsing.

The P96 check is necessary when clock control definitions are enabled and clock restriction meets one of the following conditions:

- It is set to domain_clock and -pulse_interacting_clocks is disabled.
- It is set to on.

The tool performs the check at the following times:

- During system mode transition when one or more clock controls are defined.
- At the beginning of the create_patterns command if there were any changes to the clock control definitions, cell constraints, or clock restrictions.
- When you run the report_drc_rules command if there were any changes to the clock control definitions, cell constraints, or clock restrictions.

The default handling for P96 is Error, but you can set it to Warning, Ignore, or Note.

There are four possible cases that report a P96 violation:

- When multiple clocks are forced on when clock restriction is “on”:

```
// Error: The clock control definitions combined with pin/cell
//   constraints force the following internal clocks to pulse
//   concurrently:
//                  'int_clk1' (96)
//                  'int_clk2' (97)
//                  'int_clk3' (98)
//        However, clock restriction is set to 'on', which prohibits
//        more than one clock from pulsing at a time.
//        As a result, most faults will be untestable. (P96-1)
```

  In this case, ignoring the violation causes all faults to be AU.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- When a single clock is forced on, clock restriction is on, and there are other non-constrained clocks (that is, a clock that is not pin-constrained and is not a pulse-in-capture or pulse-always clock):

```
// Error: The clock control definition combined with pin/cell
//   constraints forces internal clock 'int_clk1' (96) to pulse.
//        However, clock restriction is set to 'on', which
//        prohibits the following clocks from pulsing:
//                 'int_clk2' (97)
//                 'int_clk3' (98)
//        As a result, all faults in the above 2 clock domains will
//        be untestable. (P96-1)
```

  In this case, ignoring the violation causes the faults on the clock domains that are prohibited from pulsing to be AU.

- When multiple incompatible clocks with a common source clock are forced on while the clock restriction is domain_clock:

```
// Error: The clock control definitions combined with cell
// constraints force the following internal incompatible clocks to
// pulse concurrently:
//             '/clk_testshell' (195)
//             '/clk_smart' (196)
//        However, clock restriction is set to 'domain_clock', which
//        prohibits incompatible clocks to pulse concurrently.
//        As a result, most faults will be untestable. (P96-1)
```

  In this case, when there is no source clock in the clock control, ignoring the violation causes all faults to be AU.

```
// Error: The clock control definitions combined with pin/cell
//   constraints force the following internal incompatible clocks to
//   pulse concurrently:
//                 'int_clk1' (96)
//                 'int_clk2' (97)
//                 'int_clk3' (98)
//        However, clock restriction is set to 'domain_clock', which
//        prohibits incompatible clocks to pulse concurrently.
//        As a result, all faults in the internal clock domains
//        with source clock 'sck1' (5) will be untestable. (P96-1)
```

  In this case, when there is a source clock in the clock control, only the faults in internal clocks with the same source clock are AU.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- When a single clock is forced on while the clock restriction is domain_clock, and some non-constrained clocks incompatible with that clock exist:

```
// Error: The clock control definition combined with pin/cell
//   constraints forces internal clock 'int_clk1' (96) to pulse.
//        However, clock restriction is set to 'domain_clock', which
//        prohibits the following incompatible clocks from pulsing:
//                  'int_clk2' (96)
//                  'int_clk3' (97)
//        As a result, all faults in the above 2 clock domains will
//        be untestable. (P96-1)
```

  In this case, ignoring the violation causes the faults on the incompatible clock domains to be AU.

The summary message issued by the report_drc_rules command is as follows:

```
P96:  #fails=1 handling=error (clock_control forced-on check)
```

P97

Category: Procedure Rules

Contexts Supported: all

Default Handling: Error (cannot be changed with the set_drc_handling command)

repot_drc_rules: Supported

Verifies in flat models that the input port of a loopback is driven by the output port of the loopback.

The input port of the loopback cannot be connected to a clock, a constraint pin, or a used pin. An exception exists for BSCAN patterns, in which the input port of the loopback can be driven separately by another input pin because the output port of the loopback can be a tristate, which enables the input port of the loopback to be driven separately.

The error message describes the detected driver for the input port and is one of the following:

```
// Error: '<clock_name>' is a clock and cannot be an input port of the
// loadboard loopback. (P97-1)

// Error: '<pin_name>' is a constrained pin and cannot be an input port
// of the loadboard loopback. (P97-1)

// Error: '<pin_name>' is a used pin and cannot be an input port of the
// loadboard loopback. (P97-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Core Mapping for ATPG and Scan Pattern Retargeting Rules (R Rules)

Prior to performing core mapping for ATPG or for Scan Pattern Retargeting, Tessent Shell performs rule checks on the design as you switch from setup to analysis mode.

This design rule checking phase reads the Tessent Core Description (TCD) files and core instance information. The tool applies all R rules, except for R7, to the Core mapping for ATPG and Scan Pattern Retargeting functionality. See "Tessent Core Description" on page 4793.

R1 3515
R2 3515
R3 3515
R4 3516
R5 3516
R6 3517
R7 3518
R8 3518
R9 3521
R10 3522
R11 3523
R12 3524
R13 3525
R14 3525
R15 3527
R16 3530
R17 3532
R18 3532
R19 3533
R20 3534
R21 3534
R22 3535
R23 3536
R27 3536
R28 3537

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

R1

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting, EDT Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting, patterns -failure_mapping

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that ports defined in the core description match pins at the boundary of core instance(s).

This rule helps detect when a core description is associated with the wrong design module.

R2

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting, EDT Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Extracts connectivity between the core-level scan pins and top-level ports, including extraction of pipelining and inversion. Then, validates the ability to shift from top-level ports to the scan inputs on the core boundary, and from the scan output pins on the core boundary to top-level ports.

How to Debug a R2 Violation

To debug R2 violations, examine the shift simulation values to determine why the scan from the core to the top level is blocked using the following command.

set_gate_report drc shift

You can use the report_gates command and the trace capability in Tessent Visualizer to analyze the blockage.

You can use the analyze_drc_violation command to debug a given R2 violation graphically.

If your debug efforts locate multiple outputs on the trace path, use the preferred_core_extraction_endpoint port property to specify the output you want the tool to use. Refer to “Port” on page 3683 for more information about this property.

R3

Category: Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan_retargeting

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Verifies that scan input pins of non-identical cores are not shared, unless broadcasting to multiple instances of the same core type.

- When broadcasting to multiple instances of the same core, the pipelining and inversion for each set of equivalent channels must be identical.
- When patterns are loaded from the chip level, if broadcasting to multiple core instances, they must load identical values at the same time.

Verifies that a top-level pin does not drive multiple channels of the same core.

Verifies that the connection from the top-level pin to identical cores connects to the pins of each core in the same order.

R4

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Validates core-level pin constraints against capture-cycle states at the core instance boundary.

The conditions present at the core boundary during core-level pattern generation must be satisfied for each core instance when the core is instantiated in the design. To verify that the same conditions occur, this DRC validates core-level pin constraints against capture-cycle states at the core instance boundary.

Note

This DRC only validates pin constraints that are specified by the add_input_constraints command.

To perform this validation, the tool simulates the top-level test procedures (test_setup, load_unload, and shift) and input constraints, and compares the stable (constant) values during capture to the pin constraint values at the boundary of each core. The core-level input constraints are stored in the core description file.

You can use the analyze_drc_violation command to debug a given R4 violation graphically.

R5

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Default Handling: Warning

report_drc_rules: Supported

Validates that the stable values of sequential elements at the end of test_setup at the core level are met after simulating the top-level test_setup procedure. Those are the sequential elements where the "stable_after_setup" value was non-X at the core level.

If the sequential element is no longer present in the design in the retargeting phase due to black boxing or gray boxing of the core, the sequential element's value cannot be validated and it is omitted.

Because the tool cannot determine whether mismatches between the test_setup at the core and the top level matter, this DRC defaults to warning. For example, the state of a core's TAP instruction register may vary after test_setup at the core versus top level, but this may not matter for the core's scan operation.

You can use the analyze_drc_violation command to debug a given R5 violation graphically.

R6

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Verifies the equivalence of the load_unload and shift procedures at the core and top levels.

This DRC validates simulated top-level load_unload and shift procedures against load_unload and shift procedures at the core level.

Pins that are forced in the test_setup procedure are maintained during the load_unload and shift procedures if they are not specifically constrained in the dofile.

You can use the "report_procedures -core core_name" and the "report_timeplates -core core_name" commands to debug an R6 violation.

Changing the Default Handling

The R6 DRC can be pessimistic because you may have forced a value, either directly or indirectly, at the core level that is not relevant to the operation of the core. In this case, if you choose to change the default handling for R6 from Error to Warning, the implications of doing that depend on whether the violation was due to over-specification (tool pessimism) or whether the flagged mismatch was needed for correct circuit operation. If the value that is no longer forced from the top is not relevant, downgrading R6 has no impact.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

R7

Category: Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Verifies that all primary inputs (PIs) are controlled deterministically. Verification occurs immediately before scan pattern generation or core-level pattern retargeting.

The tool excludes any PI (port) from automatic constraints as a result of applying the is_excluded_from_isolation_constraints attribute (found in the Hierarchical Design Data Model Port attribute table). For these excluded ports, this DRC verifies that the PI has deterministic control and has not been left for ATPG to justify different values in different patterns.

Specifically, the tool verifies that every PI signal satisfies at least one of the following criteria:

- The PI has a pin constraint.
- The PI has an ATPG constraint.
- If NCPs are present and enabled, the PI must be forced or pulsed in all cycles of the internal mode of every enabled NCP.

The tool does not check the following PIs:

- Any internal PI.
- A PI that is blocked by pins or cells whose values are constant during capture.
- A PI that is defined as a pulse-always clock.
- A PI that is defined as a pulse-in-capture clock.

Note

You must fix all R7 DRC violations for pattern retargeting. When pattern retargeting is not a requirement, you can downgrade R7 DRC severity to a warning.

R8

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error

report_drc_rules: Supported

Verifies that the core-level load_unload and shift procedures can be correctly retargeted.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In patterns -scan context, when top-level load_unload and shift procedures are present, R8 additionally checks if core-level procedures can be correctly merged with top-level procedures.

The default setting for this rule is an Error. You can change the setting to be either Warning or Note by using the set_drc_handling command.

You can use the analyze_drc_violation command to debug the R8 violations that perform tracing.

The following checks are performed during test procedure retargeting:

- The test procedure is in the core description file. If this is not true, the following error is reported:

```
// Error: load_unload procedure retargeting failed for
// core instance 'coreInstance' (core 'coreName', mode 'coreMode').
// Core-level procedures are missing.
```

- Top-level clock off-states are consistent with those at the core level. If this is not true, the following error is reported:

```
// Error: load_unload procedure retargeting failed for
// core instance 'coreInstance' (core 'coreName', mode 'coreMode').
// Clock pin 'pinName' is driven by top-level port 'portName'. The
// core-level clock pin has an off-state 'value', but the top-level
// clock port has an off-state 'value'.
```

- The path between the core pin and the top-level pin contains only buffers and inverters. If this is not true, the following error is reported:

```
// Error: load_unload procedure retargeting failed for
// core instance 'coreInstance' (core 'coreName', mode 'coreMode').
// Core Pin 'pinName' cannot be traced to the top level. Tracing
// stopped at a gate 'gateName' (gateId).
```

- The core-level clock pin that should be pulsed is instead constantly forced. If this is true, the following error is reported:

```
// Error: load_unload procedure retargeting failed.
// The core-level clock pin 'pinName' (core 'coreName', mode
// 'coreMode') is now constantly forced to 'value'.
```

- The core-level pin that should be forced is constantly forced to an opposite value. If this is true, the following error is reported:

```
// Error: load_unload procedure retargeting failed.
// The core-level pin 'pinName' (core 'coreName', mode 'coreMode')
// is forced to 'value' in the core-level load_unload procedure,
// but it is now constantly forced to 'value'.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The top-level port that is constrained during load_unload can only be forced to the constraint value. If this is true, the following error is reported:

```
// Error: load_unload procedure retargeting failed for
// core instance 'coreInstance' (core 'coreName', mode 'coreMode').
// Core pin 'pinName' is driven by top-level port 'portName'.
// The core-level pin is forced to 'value' in the core-level
// load_unload procedure, but the top-level port is currently
// constrained to 'value' during load_unload.
```

- The top-level port is forced to a different value. If this is true, the following error is reported:

```
// Error: load_unload procedure retargeting failed in
// cycle 'cycleNumber' 1 of the 'procedureName' procedure.
// Top-level port 'portName' has conflicting requirements from the
// existing top-level procedure and the core-level pin(s) it drives:
//
// Top-Level Core-Level Core Pin  Inverted Core        Mode
// --------- ---------- -------   -------  ----------  ----------
// 'value'   'value'    'corePin' Yes      'coreName'  'modeName'
// 'value'   'value'    'corePin'          'coreName'  'modeName'
```

- A clock pulse definition is missing in the timeplate. If this is true, the following error is reported:

```
// Error: load_unload procedure merging failed.
// Top-level clock port 'portName' which is driving the following
// core-level pin: 'pinName' (core 'coreName', mode 'coreMode')
// has no pulse definition in timeplate 'timeplateName'.
```

- The top-level clock is forced to its on-state. If this is true, the following error is reported:

```
// Error: load_unload procedure merging failed.
// Top-level clock port 'portName' driving the following core-level
// pin: 'pinName' (core 'coreName', mode 'coreMode') is forced to
// its on-state of 'value'.
```

- The top-level clock is pulsed a different number of times in two cores. If this is true, the following error is reported:

```
// Error: load_unload procedure retargeting failed.
// Top-level clock port 'portName' has conflicting pulse
// requirements from the existing top-level procedure and the
// core-level pin(s) it drives:
//
//                Cycles with
// Clock Pulses   Clock Pulse  Core Pin     Core        Mode
// ------------   -----------  --------     ----------  ----------
// 'pulseCount'   'cycleList'  'pinName'    'coreName'  'coreMode'
// 'pulseCount'   'cycleList'  'pinName'    'coreName'  'coreMode'
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The top-level procedures have conflicting end_measure requirements:

```
// Error: load_unload procedure retargeting failed.
// Top-level procedures have conflicting end_measure requirements
// with core-level procedures. (R8-1)
```

- The core-level procedures have conflicting end_measure requirements:

```
// Error: load_unload procedure retargeting failed.
// Conflicting core-level end_measure requirements. (R8-1)
```

R9

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Warning (severity can be changed)

report_drc_rules: Supported

Verifies during core extraction that a core-level capture clock connects to a matching top-level clock.

This DRC only checks clocks on the core boundary; clocks internal to the core are not validated.

If a core-level clock is constrained off, and it is not pulsed in the core-level test_setup, load_unload, shift, or test_end procedures, capture validation is not needed because the R4 DRC validates all core-level input constraints. If the core-level clock is constrained off and is pulsed in any of these procedures, the tool performs the following checks:

- Verifies that the core-level clock connects to a top-level pin (either a port or cut point defined by the add_clocks command on an internal pin).
- Verifies that the offstates of the clock and top-level pin match (taking inversion into account).

If a core-level clock is not constrained off and is a pulse-always clock, the tool does the following:

- Verifies that the core-level clock connects to a top-level pin (either port or cut point defined by the add_clocks command on an internal pin).
- Verifies that the top-level pin is defined as a pulse-always clock.
- If a clock is synchronous (not added with “add_clocks -period”), the tool verifies that the off-states match, taking inversion into account. If a clock was added with “add_clocks -period”, the tool does not compare off-states.
- If a clock is synchronous, the tool verifies that it is not connected to a top-level asynchronous free-running clock.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If a core-level clock is not constrained off, is not a pulse-always clock, and is not the capture clock, the tool does not perform any additional R9 checks.

Note

This DRC considers a clock to be a capture clock if the clock is not blocked, is not an internal primary input, is not constrained during capture, is pulse-always or pulse-in-capture, or if one or more NCPs exists, is pulsed at least once in the active NCPs.

If a core-level clock is not constrained off, is not a pulse-always clock, and is a capture clock, the tool performs the following additional checks:

- Verifies that a core-level clock connects to a top-level pin (either a port or cut point defined by the add_clocks command on an internal pin).
- Verifies that the offstates of a clock and top-level pin match (taking inversion into account).

For pattern retargeting when a core-level pin is a pulse-in-capture clock (that is, added with the "add_clocks -pulse_in_capture" command and switch), then the tool performs the following additional checks:

- If top-level clock is always_pulse issue, the tool issues an R9 violation.
- If top-level clock is [async] free-running, the tool issues an R9 violation.

See "Scan Pattern Retargeting" in the *Tessent Scan and ATPG User's Manual*.

For core reuse when a core-level pin is a pulse-in-capture clock (that is, added with the "add_clocks -pulse_in_capture" command switch), then the tool performs the following additional checks:

- If the top-level clock has no special property (that is, always_capture, async free_running, and so on), then the tool updates the clock, which means the tool maps and applies the core-level property to the top-level clock.
- In other cases the tool issues an R9 and makes no changes to the top-level clock.

See "Core Mapping for ATPG Process Overview" in the *Tessent Scan and ATPG User's Manual*.

R10

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting, EDT Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_drc_rules: Supported

Verifies that top-level ICL is available when any of the core instances being retargeted use IJTAG in their test_setup or test_end procedures. This check also applies if any EDT IP instances have been added. If you have disabled IJTAG, then this DRC is not run by the tool.

To correct this error condition, you should use the extract_icl command to extract top-level ICL.

By default, this violation is an error and can be downgraded to warning or note. If you downgrade this DRC to a lesser violation and the lesser violation occurs, the tool bypasses any remaining IJTAG retargeting.

This DRC reports the following possible error messages:

```
//  Error: IJTAG was used in the test_setup and test_end procedures of
//         core 'piccpu_maxlen16' (mode 'internal')
//         but top-level ICL was not extracted or read.
//         Use the 'extract_icl' command to extract top-level ICL or read
//         the top-level ICL model. (R10-1)
//  Error: There was 1 R10 violation (Top-level ICL not extracted while
//         IJTAG used in cores).

//  Note:  IJTAG was used in the test_setup and test_end procedures of
//         core 'piccpu_maxlen16' (mode 'internal')
//         but top-level ICL was not extracted or read.
//         The PDL will not be automatically retargeted and appended to
//         the top-level procedure. (R10-1)
```

R11

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting, EDT Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Verifies that every core instance that uses IJTAG at the core-level matches an ICL instance at the top-level. This includes design core instances in which IJTAG was used in test_setup or test_end, in addition to any EDT instrument.

By default, this violation is an error, but you can downgrade it to warning or note for non-scan instruments. If you downgrade this DRC to a lesser violation and the lesser violation occurs, the IJTAG retargeting capability skips any core instance with IJTAG that does not have a corresponding ICL instance at the top level. If the IJTAG instrument has a scan interface, it requires ICL, and you cannot ignore the DRC during scan operations. In this case, the tool automatically upgrades the R11 to an error during the scan identification process.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This DRC reports the following possible error messages:

```
//  Error: Current design (mode 'internal', ICL module 'piccpu_maxlen16')
//         is expected to have ICL instance 'tessent_occ_i_clk'
//         in the ICL model, but no such ICL instance was found.
//         Read the missing ICL model and rerun ICL extraction. (R11-1)
//  Error: Core 'piccpu_maxlen16' (mode 'internal', ICL module //
//         'piccpu_maxlen16')
//         is expected to have ICL instance 'piccpu_maxlen16_edt_i'
//         in the ICL model, but no such ICL instance was found.
//         Read the missing ICL model and rerun ICL extraction. (R11-2)
//  Error: Core instance 'chip_int_1/piccpu_2' (core 'piccpu_maxlen16',
//         mode 'internal') uses IJTAG but has no ICL instance
//         corresponding to it in the ICL model.
//         Rerun ICL extraction and provide the missing ICL model.(R11-3)
//  Error: There were 3 R11 violations (Missing ICL instance for core
//         which uses IJTAG).
//  Error: Instrument instance 'piccpu_maxlen16_edt_i'
//        (core 'piccpu_maxlen16_edt', instrument type 'edt')
//         uses IJTAG but has no ICL instance corresponding to it in
//         the ICL model. Rerun ICL extraction and provide the missing
//         ICL model. (R11-1)
```

R12

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Warning (severity can be changed)

report_drc_rules: Supported

Verifies that core-level iCalls are at the end of the test_setup procedure or the start of the test_end procedure.

The tool automatically maps all iCalls located at the end of the test_setup procedure or at the beginning of the test_end procedure; all others are ignored.

By default this violation is a warning and can be changed to error or note. If you downgrade this DRC from error and has a violation, the IJTAG retargeting capability skips all core-level iCalls that have an invalid position.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This DRC reports the following possible error messages:

```
//  Error: Core 'piccpu_maxlen16' (mode 'internal', ICL module
//         'piccpu_maxlen16') calls iProc 'setup'
//         of ICL instance 'piccpu_maxlen16_edt_i' (ICL module
//         'piccpu_maxlen16_edt') in test_setup,
//         but not at the end of the test procedure.
//         It will not be automatically retargeted and appended to the
//         top-level procedure. (R12-1)
//  Error: Core 'piccpu_maxlen16' (mode 'internal', ICL module
//         'piccpu_maxlen16') calls iProc 'my_proc'
//         in test_end, but not at the beginning of the test procedure.
//         It will not be automatically retargeted and appended to the
//         top-level procedure. (R12-2)
//  Error: There were 2 R12 violations (Position of IJTAG in core-level
//         test procedure prohibits automatic retargeting).
```

R13

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting, EDT Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Verifies that the used core-level iProcs are loaded in memory and are registered against the right ICL instances.

By default, this violation is an error and can be downgraded to warning or note. If this DRC is downgraded from an error and has a violation, the IJTAG retargeting functionality skips all core-level iProc calls that have not been read in at the top level.

This DRC reports the following possible error messages:

```
//  Error: Core 'piccpu_maxlen16' (mode 'internal') calls iProc 'setup'
//         in test_setup, but this iProc is not read for ICL module
//         'piccpu_maxlen16_edt'. (R13-1)

//  Error: Core 'piccpu_maxlen16' (mode 'internal') calls iProc 'my_proc'
//         in test_end, but this iProc is not read for ICL module
//         'piccpu_maxlen16'. (R13-2)

//  Error: There were 2 R13 violations (Missing iProc used at core level).

//  Error: Core 'piccpu_maxlen16_edt' (instrument type 'edt') calls iProc
//         'setup', but this iProc was not read for ICL module
//         'piccpu_maxlen16_edt'. (R13-1)
```

R14

Category: EDT Instruments

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Contexts Supported: patterns -scan

Default Handling: Error (severity can be changed to warning)

report_drc_rules: Supported

Validates mapping of static and dynamic control signals.

This DRC validates that static and dynamic EDT IP control signals are properly mapped to the top. Any IJTAG-controlled instruments that do not have a scan interface are also validated.

If the value on the IP boundary is different than what is expected or the signal cannot be traced to the top, the tool issues an R14 violation.

By default, this violation is an error and can be downgraded to warning. When R14 fails and is downgraded to a warning, no automatic procedure updates, clock and constraint adding are performed for the failing pin. You must ensure pin requirements in this case.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This DRC reports the following possible error messages:

```
//  Error: Tracing with the stable_shift simulation context from
//         core-level control input pin
//         'TOP_DE1_rtl_tessent_occ_elo_inst/slow_clock'
//         (core 'TOP_DE1_rtl_tessent_occ', instrument type 'occ')
//         to the top level has failed. (R14-1)

//  Error: Tracing with the stable_capture simulation context from
//         core-level control input pin
//         'TOP_DE1_rtl_tessent_occ_elo_inst/slow_clock'
//         (core 'TOP_DE1_rtl_tessent_occ', instrument type 'occ')
//         to the top level has failed. (R14-1)

//  Error: Core-level control input pin
//         'TOP_DE1_rtl_tessent_occ_elo_inst/capture_cycle_width[1]'
//         (core 'TOP_DE1_rtl_tessent_occ', instrument type 'occ')
//         is not driven to the expected value of '0'
//         or tracing with the stable_shift simulation context to the
//         top level has failed.
//         Simulated pin value is '1'. (R14-2)

//  Error: Tracing with the stable_shift simulation context from
//         core-level control input pin
//         'cba9_gate_tessent_lpct_i/scan_en'
//         (core 'cba9_gate_tessent_lpct', instrument type 'lpct')
//         to the top level has failed.
//         Multiple potential drivers found:
//            'tie_high' (31.0),
//            'CS' (5.0).
//         Use 'set_attribute_value -name constraint_value_during_load_unload'
//         command to constrain ports and guide the tracer. (R14-1)

//  Error: Tracing with the stable_capture simulation context from
//         core-level control input pin
//         'cba9_gate_tessent_lpct_i/scan_en'
//         (core 'cba9_gate_tessent_lpct', instrument type 'lpct')
//         to the top level has failed.
//         Multiple potential drivers found:
//            'tie_high' (31.0),
//            'CS' (5.0).
//         Use the 'add_input_constraints' command to constrain ports
//         and guide the tracer. (R14-1)
```

R15

Category: EDT Instruments

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Validates consistency of static and dynamic signals.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This DRC validates that static and dynamic EDT IP signals have consistent requirements properly mapped to the top.

The R15 rules checks if any top-level ports have the same requirements between multiple instruments and already specified values (clock-off state, constrain value).

This DRC checks the following properties:

- Is clock port off-state correct?
- Is the capture constraint value correct?
- Is the load_unload force value correct?
- Is the shift force value correct?

All channel sharing rules are validated by the existing K15 rule.

By default R15 handling is set to error and cannot be downgraded.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This DRC reports the following possible error messages:

```
//  Error: Conflicting top-level clock 'edt_clock' clock-off values
//         between core instances and user defined clock. (R15-1)
//         The clock-off value defined at the top-level is '1'.
//
//         Core Pin   Instance    Core      Type  Top Level  Core Level
//         ---------  ----------  --------  ----  ---------  ----------
//         edt_clock  blk1_edt_i  blk1_edt  edt       0           0
//         edt_clock  blk2_edt_i  blk2_edt  edt       0           0
//         edt_clock  blk3_edt_i  blk3_edt  edt       0           0
//         edt_clock  blk4_edt_i  blk4_edt  edt       0           0
//         edt_clock  blk5_edt_i  blk5_edt  edt       0           0
//         edt_clock  blk6_edt_i  blk6_edt  edt       0           0
//         edt_clock  blk7_edt_i  blk7_edt  edt       0           0
//
//  Error: Conflicting top-level clock 'edt_clock' clock-off values
//         between core instances. (R15-2)
//         Expected clock-off value is '1'.
//
//         Core Pin   Instance    Core      Type  Top Level  Core Level
//         ---------  ----------  --------  ----  ---------  ----------
//         edt_clock  blk1_edt_i  blk1_edt  edt       1           1
//         edt_clock  blk2_edt_i  blk2_edt  edt       0           0
//         edt_clock  blk3_edt_i  blk3_edt  edt       0           0
//         edt_clock  blk4_edt_i  blk4_edt  edt       0           0
//         edt_clock  blk5_edt_i  blk5_edt  edt       0           0
//         edt_clock  blk6_edt_i  blk6_edt  edt       0           0
//         edt_clock  blk7_edt_i  blk7_edt  edt       0           0
//
//  Error: Conflicting top-level port 'edt_low_power_shift_en' forces
//         requirement in 'load_unload' procedure between core instances.
//         (R15-3)
//         Expected force value is '1'.
//
//     Core Pin                Instance    Core      Type  Top Level  Core Level
//     ----------------------  ----------  --------  ----  ---------  ----------
//     edt_low_power_shift_en  blk1_edt_i  blk1_edt  edt       1           1
//     edt_low_power_shift_en  blk2_edt_i  blk2_edt  edt       0           0
//     edt_low_power_shift_en  blk4_edt_i  blk4_edt  edt       1           1
//     edt_low_power_shift_en  blk7_edt_i  blk7_edt  edt       1           1
//
//  Error: Conflicting top-level port 'edt_update' forces requirement in 'shift'
//         procedure between core instances. (R15-4)
//         Expected force value is '0'.
//
//         Core Pin    Instance    Core      Type  Top Level  Core Level
//         ----------  ----------  --------  ----  ---------  ----------
//        edt_update   blk1_edt_i  blk1_edt  edt       0           0
//        edt_update   blk2_edt_i  blk2_edt  edt       0           0
//        edt_update   blk4_edt_i  blk4_edt  edt       1           0
//        edt_update   blk7_edt_i  blk7_edt  edt       1           0
//
//  Error: Conflicting top-level port 'edt_clock' capture constraint values
//         between core instances and user defined constraint. (R15-5)
//         The capture constraint value defined at the top-level is '1'.
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//          Core Pin   Instance    Core      Type  Top Level  Core Level
//          ---------  ----------  --------  ----  ---------  ----------
//          edt_clock  blk1_edt_i  blk1_edt  edt       0           0
//          edt_clock  blk2_edt_i  blk2_edt  edt       0           0
//          edt_clock  blk3_edt_i  blk3_edt  edt       0           0
//          edt_clock  blk4_edt_i  blk4_edt  edt       0           0
//          edt_clock  blk5_edt_i  blk5_edt  edt       0           0
//          edt_clock  blk6_edt_i  blk6_edt  edt       0           0
//          edt_clock  blk7_edt_i  blk7_edt  edt       0           0
//
//  Error: Conflicting top-level port 'edt_clock' capture constraint values
//         between core instances. (R15-6)
//         Expected capture constraint value is '1'.
//
//          Core Pin   Instance    Core      Type  Top Level  Core Level
//          ---------  ----------  --------  ----  ---------  ----------
//          edt_clock  blk1_edt_i  blk1_edt  edt       1           0
//          edt_clock  blk2_edt_i  blk2_edt  edt       0           0
//          edt_clock  blk3_edt_i  blk3_edt  edt       0           0
//
//  Error: There were 6 R15 violations (Conflicting port requirements).
```

R16

Category: EDT Instruments

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Validates EDT IP core scan chain binding.

This DRC validates if the scan chain binding is correct for the newly added EDT IP cores. This DRC is executed after automatic scan chain binding is executed.

The tool attempts to bind EDT instruments with the following user-defined scan chains:

- Direct access chains
- User added internal chains that are not bound to an EDT block

Direct access chains can be bound with an EDT instrument in bypass or single bypass mode.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This DRC reports the following possible error messages similar to the following:

If corresponding EDT instance scan pins are connected to different chains:

```
//  Error: Core instance 'core_1/piccpu_maxlen16_1_edt_i'
//  (core 'piccpu_maxlen16_1_edt', instrument type 'edt') (R16-1)
//    EDT instance scan pins corresponding to chain index 1 are connected
//    to two different scan chains:
//       EDT scan pin ‘core_1/piccpu_maxlen16_1_edt_i/edt_scan_out[0]’
//        connects to scan chain input ‘core_1/input56’ of ‘chain56’
//       EDT scan pin 'core_1/piccpu_maxlen16_1_edt_i/edt_scan_in[0]’
//        connects to scan chain output ‘core_1/output57’ of ‘chain57’
```

If one side of the chain is bound, and the pin specified for the other side does not exist in the design (so not a connectivity issue):

```
//  Error: Core instance 'core_1/piccpu_maxlen16_1_edt_i'
//  (core 'piccpu_maxlen16_1_edt', instrument type 'edt') (R16-1)
//   One of the scan pins of the following scan chains does not exist in
//   the design:
//    chain56 (the scan output
//    'core_1/piccpu_maxlen16_1_edt_i/edt_scan_out1[1]'
//    does not exist in the design),
//    chain57 (the scan input 'core_1/foo' does not exist in the design).
```

If one side of the chain is bound, and the other side has a valid pin with no binding:

```
//  Error: Core instance 'core_1/piccpu_maxlen16_1_edt_i'
//  (core 'piccpu_maxlen16_1_edt', instrument type 'edt') (R16-1)
//    One of the scan pins of the following scan chains is not connected
//    to the EDT instance:
//      chain56 (no connection found from scan chain outputABC' to the EDT
//      instance pin ‘core_1/piccpu_maxlen16_1_edt_i/edt_scan_out[1]’),
//      chain57 (no connection found from EDT instance pin
//      'core_1/piccpu_maxlen16_1_edt_i/edt_scan_in[3]’ to scan chain
//      input core_1/foo').
```

If neither side is bound, and both scan pins do not exist in the design:

```
//  Error: The scan pins of the following scan chains do not exist in the
//  design. (R16-1)
//    edt_chain0 (input = ‘foo1’, output = ‘bar1’),
//    edt_chain1 (input = ‘foo2’, output = ‘bar2’).
```

If neither side is bound, and SCI exists in the design but SCO does not:

```
//  Error: The scan output pins of the following scan chains do not exist
//  in the design and their scan input pins do not connect to any EDT
//  instance.  (R16-1)
//    edt_chain0 (input = ‘input1’, output = ‘foo’),
//    edt_chain1 (input = ‘input2’, output = ‘bar’).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If neither side is bound, and SCO exists in the design but SCI does not:

```
//  Error: The scan input pins of the following scan chains do not exist
//  in the design and their scan output pins do not connect to any EDT
//  instance.  (R16-1)
//    edt_chain0 (input = 'foo', output = 'output1'),
//    edt_chain1 (input = 'bar', output = 'output2').
```

If neither side is bound, but both SCI and SCO exist in the design:

```
//  Error: The scan pins of the following scan chains do not connect to
//  any EDT instance.  (R16-1)
//    edt_chain0 (input = 'input1', output = 'output1'),
//    edt_chain1 (input = 'input2', output = 'output2').
```

R17

Category: EDT Instruments

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Validates if EDT IP unused channels are tied off.

Accessible input channel number selection requires that unused channels are feeding constant '0' values into the decompressor.

By default R17 handling is set to error and cannot be downgraded.

This DRC reports the following possible error messages:

```
//  Error: Core instance 'core_1/piccpu_maxlen16_1_edt_i' (core
//         'piccpu_maxlen16_1_edt', instrument type 'edt') (R17-1)
//         has inaccessible input channels that are not tied off:
//            core_1/piccpu_maxlen16_1_edt_i/edt_channels_in[2],
//            core_1/piccpu_maxlen16_1_edt_i/edt_channels_in[3].
//  Error: Core instance 'core_1/piccpu_maxlen16_1_edt_i' (core
//         'piccpu_maxlen16_1_edt', instrument type 'edt') (R17-2)
//         has an inaccessible input channel that is not tied off:
//            core_1/piccpu_maxlen16_1_edt_i/edt_channels_in[3].
//  Error: There were 2 R17 violations (Inaccessible input channels
//  not tied off).
```

R18

Category: Tessent OCC

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Verifies after scan chain tracing that the Tessent OCC registers are part of the scan chains.

Applies only when using the Tessent OCC in a mode where it is controllable by scan.

This DRC reports the following possible error messages:

```
//  Error: OCC configuration registers of core instance
//  'top_gates_tessent_occ_NX1_inst' (core 'top_gates_tessent_occ',
//  instrument type 'occ') are not part of active scan chains. (R18-1)
//    The following OCC configuration registers exist in the design but
//    are not part of any active scan chain.
//      'top_gates_tessent_occ_NX1_inst/occ_control/ShiftReg/FF_reg_0_',
//      'top_gates_tessent_occ_NX1_inst/occ_control/ShiftReg/FF_reg_1_',
//      'top_gates_tessent_occ_NX1_inst/occ_control/ShiftReg/FF_reg_2_',
//      'top_gates_tessent_occ_NX1_inst/occ_control/ShiftReg/FF_reg_3_'.
//    Please make sure all configuration registers of OCCs that are active
//    in this mode are stitched into scan chains that are active in this
//    mode and all EDT instances active in this mode were added using the
//    'add_core_instances' command.
//  Error: There was 1 R18 violation (OCC configuration registers are not
//  part of any active scan chains).

//  Error: OCC configuration register of core instance
//  'top_gates_tessent_occ_NX1_inst' (core 'top_gates_tessent_occ',
//  instrument type 'occ') does not exist in the design. (R18-1)
//    Please make sure the hierarchy inside the OCC IP is preserved during
//    synthesis, and that design object names are not changed.
//      'top_gates_tessent_occ_NX1_inst/occ_control/ShiftReg/KUKU[0]'.
//  Error: There was 1 R18 violation (OCC configuration registers are not
//  part of any active scan chains).
```

The tool reports the second error message when the design requires register mapping instead of instance mapping. You can fix the error by mapping the registers with the add_rtl_to_gates_mapping command. Use the optional arguments "-register_name_prefix *register_name_prefix*" and "-register_name_suffix *register_name_suffix*" to specify names other than the defaults. See "-register_name_prefix *register_name_prefix*" on page 352 for more information.

R19

Category: Tessent OCC

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Verifies that Tessent OCC internal nodes exist in the design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This DRC reports the following possible error message:

```
// Error: Core '<OCC IP instance>' (core '<OCC IP core name>',
// instrument type 'occ') (R19-1)
// The following OCC internal paths do not exist in the design.
// /top_gates_tessent_occ_NX1_inst/name_path1,
// /top_gates_tessent_occ_NX1_inst/name_path2.
```

Tip

If you see R19 DRC messages, check that your TCD file was not manually edited and that Tessent persistent buffers have been preserved.

R20

Category: EDT Instruments

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Validates that if an iCall is retargeted from one level to the next, then this iCall is for an iProc and not an iTopProc.

Detects if an iTopProc was used at the core level. It is not triggered for core instances added with -current_design switch.

By default R20 handling is set to error and can be downgraded to a warning or a note. If the DRC is downgraded, those iCalls are not mapped.

This DRC reports the following possible error messages:

```
//  Error: Core 'piccpu_maxlen16' (mode 'internal') calls iTopProc
//         'my_proc' in the test_end procedure.
//         iTopProc can only be defined for the top level and therefore
//         cannot be retargeted from a lower-level instance. (R20-1)
//  Error: There was 1 R20 violation (iTopProc used at core level).
```

R21

Category: EDT Instruments

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Verifies that the gate-level register instrument names correspond to those found in the RTL.

By default, R21 handling is set to error. It can be downgraded to a warning or a note.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This DRC reports the following possible error message:

```
//  Error: Core instance 'I' (core 'C', instrument type 'T') (R21-1)
//         The following debug registers were not found in the design:
//          'R1', 'R2'.
//  Error: There was 1 R21 violation (The instrument's debug registers not
//  found in the design).
```

where I is the core instance name, C is the core name, T is the instrument type, and R<*n*> are register names.

If this error occurs because the flops are part of a multibit flop tray, use one of the following approaches:

- Resynthesize the IP and prevent use of multibit flop trays.
- Downgrade the error to a warning. In this case, testbench monitoring related to the affected registers is unavailable, which could increase difficulty in debugging simulation problems.

R22

Category: EDT Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Warning (severity can be changed)

report_drc_rules: Supported

Verifies instrument IJTAG requirements for instruments without a scan interface.

This DRC can be changed to error or note. R22 typically applies to scan instruments like the EDT controller where static control signals like edt_bypass are declared as DataIn ports.

This DRC reports the following possible warnings and error messages:

```
// Error: Core 'piccpu_maxlen16_1_edt' (instrument type 'edt') is an
// IJTAG instrument with no scan interface, but top-level ICL was not
// extracted or read.
// Use the 'extract_icl' command to extract top-level ICL or read the
// top-level ICL model. (R22-1)
// Error: There was 1 R22 violation (Top-level ICL not extracted while
// IJTAG instruments with no scan interfaces present in the design).

// Warning: Core 'piccpu_maxlen16_1_edt' (instrument type 'edt') is
// an IJTAG instrument with no scan interface, but top-level ICL was not
// extracted or read.
// The PDL will not be automatically appended to the top-level
// procedure. (R22-1)
// Warning: There was 1 R22 violation (Top-level ICL not extracted
// while IJTAG instruments with no scan interfaces present in the design).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

R23

Category: EDT Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed to a Warning or Note)

report_drc_rules: Supported

Verifies scan instrument IJTAG requirements.

This DRC can be downgraded to a Warning or Note.

This DRC reports the following possible error messages:

```
// Error: Core 'piccpu_maxlen16_1_edt' (instrument type 'edt') is an
// IJTAG instrument with scan interface, but top-level ICL was not
// extracted or read.
// Use the 'extract_icl' command to extract top-level ICL or read the
// top-level ICL model. (R23-1)
// Error: There was 1 R23 violation (Top-level ICL not extracted
// while IJTAG instruments with scan interfaces present in the design).
```

R27

Category: OCC Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Verifies that the active OCCs are not driven by the outputs of other active OCCs during capture.

Cascading of OCCs can lead to incorrect capture clock sequences and cause mismatches. Active OCCs in the parent mode driving active child OCCs do not cause a R27 violation. This DRC is run during the ATPG and pattern retargeting step.

Note

If the lower OCC is a standard OCC built with the upstream_parent_occ property set to “allow”, then you must use “set_core_instance_parameters -parameter_values {active_upstream_parent_occ on}” to disable the local synchronizer of the lower OCC and ensure that an OCC activated in “parent” mode is present in its fan-in. For more information, see “OCC” on page 4356.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This DRC reports the following error message:

```
// Error: Capture clock 'corec_i1/corec_rtl2_tessent_occ_clka_inst/fast_clock'
//       (20442.1) of OCC instance 'corec_i1/corec_rtl2_tessent_occ_clka_inst'
//       (core 'corec_rtl2_tessent_occ') is driven by OCC instance
//       'coreb_i1/corea_i1/corea_rtl2_tessent_occ_clka_inst'
//       (core 'corea_rtl2_tessent_occ') output clock 'coreb_i1/corea_i1/
//        corea_rtl2_tessent_occ_clka_inst/clock_out' (19140.0).
//        An active OCC cannot be driven by another active OCC unless the driven
//        OCC is a child OCC and the driving OCC is operating in parent mode.
//        This may lead to incorrect OCC capture clock sequences and result in
//        mismatches. (R27-1)
```

R28

Category: OCC Instruments

Contexts Supported: patterns -scan, patterns -scan_retargeting, and patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Verifies that, in fast_capture mode, the fast_clock ports of OCC instances are driven by a pulse-always, pulse-in-capture, or an asynchronous clock.

This DRC performs the following checks:

1. The OCC fast clock port must be driven by a port or a pseudo-port.

   Example:

   ```
   //  Warning: Fast clock pin
   //  'coreb_rtl2_tessent_occ_clka_inst/fast_clock' (1990.1)
   //  of OCC instance 'coreb_rtl2_tessent_occ_clka_inst'
   //  core 'coreb_rtl2_tessent_occ') is not driven by a clock.
   //  This may result in incorrect patterns. (R28-1)
   ```

2. The OCC fast clock port driver, which is a port or pseudo-port, must be defined as a clock.

   ```
   //  Warning: Fast clock pin
   //  'coreb_rtl2_tessent_occ_clkc_inst/fast_clock'(1992.1)
   //  of OCC instance 'coreb_rtl2_tessent_occ_clkc_inst'
   //  (core 'coreb_rtl2_tessent_occ') is driven by port 'clkc' (3.0)
   //  which is not defined as a clock.
   //  This may result in incorrect patterns. (R28-2)
   ```

3. The OCC fast clock driver must be a clock that is pulse-always, pulse-in-capture, or an asynchronous clock.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Warning: Fast clock pin
//  'coreb_rtl2_tessent_occ_clkb_inst/fast_clock' (1991.1)
//  of OCC instance 'coreb_rtl2_tessent_occ_clkb_inst'
//  (core 'coreb_rtl2_tessent_occ') is driven by clock 'clkb' (2.0)
//  which is not pulse-always, pulse-in-capture, or asynchronous
//  (defined with a period).
//  This may result in incorrect patterns. (R28-3)
```

Use the set_drc_handling command to change the handling of this DRC to error or note.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Scannability Rules (S Rules)

Scan substitution and stitching is done with respect to library cells and pre-existing scan segments.

For library cells, a simple cell may just contain a single memory element (_dff primitive in the Tessent cell library) that exists in both the non-scan version and the scan equivalent. In this case, S-rule checking must be performed on that memory element. A more complex library cell may contain some memory elements that do not become part of the scan path after the non-scan model has been replaced with the scan model. S-rules are only run on those memory elements that become part of the scan path. The scan model for a complex library cell may contain multiple scan input and scan output pins. In this case, the tool identifies the clocks that control these embedded scan segments, and target only those clocks during S-rule analysis. Finally, you can specify pre-stitched scan segments on arbitrary Verilog modules. In this case, S-rules are once again performed with respect to the clock and set/reset pins that you specified as part of the add_scan_segments command. The specific memory elements (dff primitives) that become part of the scan path after scan chain stitching is complete are referred to as scan path memory elements.

For each sequential element in the design, the tool performs two main checks. The first check, S1, ensures that when all defined clocks — including sets and resets — are at their off-states, the sequential elements remain stable and inactive. The second check, S2, ensures that for each defined clock, sequential elements can capture data when all the other defined clocks are off. These scannability checks determine if the tool can turn off all set and reset lines, and turn on and off all clock inputs of sequential cells from the design's primary input pins. Without this controllability, the tool does not allow a sequential element to pass scannability checks, which is a requirement to be considered for scan identification.

This checking is similar to the C1 (S1) and C7 (S2) rules checks, which the tool performs to determine the stability of defined scan chains. The C1 and C7 rules perform these same checks on sequential elements that are already converted to scan. For more information on C1 and C7, refer to “Clock Rules (C Rules)” on page 2240.

Once these scannability rules pass, the elements are considered “scannable” and the DRC process treats the non-scan elements as though they were scan elements for the remainder of the DRC rules.

The default handling for S1 and S2 rules is “error”. This identifies the flops that cannot be used for scan insertion due to clock controllability issues and aborts the system mode transition. This enables you to debug your design at this stage instead of discovering it later during ATPG. You can change the default handling to “warning” with the set_drc_handling command, in which case the flops remain non-scan and may lead to poor ATPG results.

The default handling for the S3 rule is “warning” which causes the tool to add the necessary pin constraints to enable activating the clock during scan shifting. You can change the handling to

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

"error", but this does not abort the system mode transition. Instead, the tool inserts the flops into scan chains but does not add extra pin constraints.

Note

You should define any existing scan enable pins prior to running S-rules. This enables the tool to force these signals into the active state prior to running the DRC. This is necessary to avoid false S-rule violations when the design already contains clock gating logic that fixes clock controllability issues when the defined scan enable signal(s) are active.

Tip

In Tessent Visualizer, you can press Ctrl + R to execute a report_gates command on selected gates. For more information, see "How to Report Gate Data" on page 3173 on page 2215.

S1 3540
S2 3542
S3 3542
S4 3543
S5 3544
S6 3545
S7 3546
S8 3547

S1

Category: Scannability

Contexts Supported: dft -scan, dft -test_points

Default Handling: Error

report_drc_rules: Supported

Scannability rule S1 checks all the clock inputs (including sets and resets) of each scannable non-scan memory element to ensure that these inputs can be disabled. This rule ensures that non-scan elements that may be converted to scan can be controlled to hold their current data. You can use "report_drc_rules s1" to get a listing of all the violations. This report includes the pin-path and gate-id of the flop with the violation (for scan segments it reports the segment name instead).

To debug an S1 DRC violation with Tessent Visualizer, use the command "analyze_drc_violation <violation_number>". This runs the DRC analysis for the specific violation and displays the failing design elements in the Flat Schematic tab. In the following example, the failure is due to the clock pin "CP" being set to X. To pass the S1 DRC, the clock must be set to its off state of 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

analyze_drc_violation S1-2

report_drc_rules s1-2

```
//  Warning: Unstable nonscan cell /AA/OUT_2_reg(32) when all clocks are
//  off. (S1-2)
```

report_gates 32

```
// /AA/OUT_2_reg (32) TIEX
// "I0" I (0) 10-
// "I1" I (0) 27-
// CP I (X) 22-/AA/I_1/out
// "I3" I (X) 13-
// "OUT" O (X) 15-
```

Scannability rule S1 is a modified version of the C1 clock rule, in that it performs the same type of checking on nonscan sequential elements that C1 performs on scan elements. For additional information on the C1 clock rule, refer to "C1" on page 3227.

The S1 rule violation may be a result of a type-B clock gater driving leading-edge flops or scan elements driven by cascaded type-B and type-A clock gaters. For details, refer to Type-B Scan Chain Tracing Failures (T3 Violations) and Driving LE Flops With Type-B Clock Gaters in the *Tessent Scan and ATPG User's Manual*.

You can specify how these potential scan cells are handled by setting S1 to one of the following handling options:

- **Error** — Identifies cells with constant data inputs, and stops after identifying all the DRC violations.
- **Warning** — Identifies cells, reports violations, and marks these cells as non-scan.

Note

Clocks that are defined as pulse-always cannot be disabled during testing. Therefore, flops driven by pulse-always clocks fail the S1-rule checking.

Note that Tessent Scan does not use pulse-always clocks as test-clocks to repair the clock signals of the flops that fail with any S-rules.

Here are some examples of S1 DRC violation error messages:

- For flop based violation:

```
//  Error: Unstable nonscan cell because clock input
//  '/dff1_inst/CLK' (552.3) is set to X when all clocks are off.
//  (S1-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- For subchain based violation:

```
//  Error: Unstable nonscan cell because clock input
//  (subchain: 'lff_subchain_1/lff2_u2/SO2', pin: 'CLK') is driven
//  by a pulse-always clock. (S1-122)
```

S2

Category: Scannability

Contexts Supported: dft -scan, dft -test_points

Default Handling: Error

report_drc_rules: Supported

Scannability rule S2 checks all clock inputs (not including sets and resets) of each scan path memory element to see whether they can capture data. This rule ensures that a non-scan cell can capture data using one of the defined clocks. To be converted to scan, a non-scan cell must be able to capture data when a single clock is on. You can use “report_drc_rules s2” to get a listing of all the violations. This report includes the pin-path and gate-id of the flop with the violation (for scan segments it reports the segment name instead)

Scannability rule S2 is a modified version of the C7 clock rule, in that it performs the same type of checking on non-scan sequential elements that C7 performs on scan elements. For additional information on the C7 clock rule, refer to “C7” on page 3248.

You can specify how these potential scan cells are handled by setting S2 to one of the following handling options:

- **Error** — Identifies cells with constant data inputs, and stops after identifying all the DRC violations.
- **Warning** — Identifies cells, reports violations, and marks these cells as non-scan.

Here is an example of an S2 DRC violation error message:

```
//  Error: Clock capture ability check failed at clock input '/dff1_inst/CLK' (553.3) (S2-1)
```

S3

Category: Scannability

Contexts Supported: dft -scan, dft -test_points

Default Handling: Warning

report_drc_rules: Supported

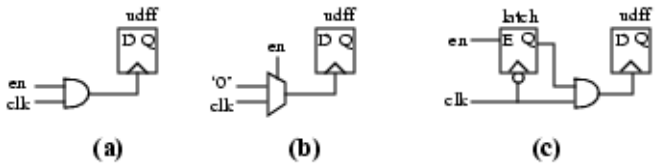
All non-clock primary input pins in the clock cone of scan path memory elements must be constrained. If these pins are unconstrained and made scannable, trace violations may occur.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Non-clock primary input pins in the clock cone of non-scan memory elements should be constrained in the ATPG dofile or load_unload procedure to correctly sensitize the clock path.

Figure 10-54 illustrates S3 violations where the pin 'clk' is defined as the shift clock with an off state value of '0'. The non-clock pin 'en' causes the violation if it is left un-constrained.

Figure 10-54. Example S3 Rule Violation



The total number of S3 violations are reported in the session transcript during the DRC rule checking. Individual S3 violations can be reported with the report_drc_rules and report_scan_elements commands.

The default DRC handling of the S3 rule is "Warning," which means that memory elements with S3 violations are kept in the pool of scannable candidates. The handling can be changed to "Error" using the set_drc_handling command, in which case the memory elements with S3 violations are excluded from the pool of scannable candidates automatically.

If you want to include some of those memory elements back in scan insertion, you can issue delete_nonscan_instances commands on them. Similarly, if the S3 rule handling is "Warning" (the default), you can obtain the list of the violations using the report_scan_elements or report_drc_rules commands and issue add_nonscan_instances commands on a subset of the list explicitly.

Although the handling of the S3 DRC violation can be changed to "Error," the effect of the violation is different from that of the S1 and S2 rules. The tool does not insert test logic on the clock pins to correct an S3 violation even if the user turns on the test logic insertion using the set_test_logic command.

Here is an example of an S3 DRC violation error message:

```
//  Warning: Un-initialized non-clock input detected for clock port aa/udff2/CLK
//           Primary input pin "d" must be set to 0 (S3-1)
```

S4

Category: Scannability

Contexts Supported: dft -scan, dft -test_points

Default Handling: Note

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Scannability rule S4 identifies and determines the handling of non-scan memory elements that satisfy certain conditions.

These conditions are:

- Meet requirements for use as scan elements.
- Maintain constant output values during the shift and capture cycles.
- Are in the clock and set/reset paths of other flip-flops/latches, RAMs, and ROMs.
- Contribute to the data lines of the memory elements in the clock and set/reset paths of other flip-flops/latches, RAMs, and ROMs.

Note

Flip-flop/latch elements that maintain constant output values during shift and capture cycles can potentially disturb the clock and set/reset signals of other flip-flops/latches, RAMs, and ROMs if their constant output values are not maintained.

If non-scan cells that maintain constant output values and are associated with clock and set/reset signals are converted to scan cells, they can prevent the clock and set/reset signal paths from being sensitized properly. When this occurs, more test logic may be required to sensitize the clock path.

You can specify how these non-scan cells are handled by setting S4 to one of the following handling options:

- **Note** — Reports the non-scan cells to the log file and leaves them as non-scan cells. Default handling.
- **Warning** — Reports the non-scan cells to the log file and converts them to scan cells. The resulting scan cells have an unknown value (TIEX) for the output during the remainder of the DRC.
- **Ignore** — Enables non-scan cells to be converted to scan cells; no reporting is done.

You can report the S4 results with the report_drc_rules command.

Here is an example of an S4 DRC violation error message:

```
//  Note: Non-scan cell /dff1(14) that maintains a constant value during
//  capture and shift phases will remain non-scan.  (S4-1)
```

S5

Category: Scannability

Contexts Supported: dft -scan, dft -test_points

Default Handling: Error

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_drc_rules: Supported

Scannability rule S5 identifies clock sources that drive set/reset ports as well as clock ports.

When a clock source (a top-level pin or an internally defined clock net) drives the set/reset port of one or more flip-flops while also driving the clock port of one or more flip-flops, the set/reset port driven by this signal is reported as an S5 violation.

You can optionally fix S5 violations by adding test logic at the set/reset ports driven by the clock. The tool then treats the set/reset ports as uncontrollable and inserts the same test logic that is used to fix the set/reset ports of flip-flops with an S1 violation.

You can specify how these potential scan cells are handled by setting S5 to one of the following handling options:

- **Error —** Identifies cells with constant data inputs, and stops after identifying all the DRC violations.
- **Warning —** Identifies cells, reports violations, and marks these cells as non-scan.

Here is an example of an S5 DRC violation error message:

```
//  Error: Set/reset port (subchain: '/my_inst/TEST__SOUT[4]', pin: 'R0')
//  is driven by clock '/test_TEST__SHIFT_CLK1' (36)
//  This signal also drives clock port
//  (subchain: '/my_inst/TEST__SOUT[0]', pin: 'TEST__SHIFT_CLK[1]') which
//  must pulse during shift (S5-1)
```

S6

Category: Scannability

Contexts Supported: dft -scan, dft -test_points

Default Handling: Error (in dft -scan context); Note (in dft -test_points context)

report_drc_rules: Supported

Scannability rule S6 identifies both subchains and pre-existing scan cells (that are not part of declared scan chains or subchains) that have driven scan-in ports and reports them as an S6 violation.

Subchain instances with driven scan-in ports are deleted. That is, they are no longer considered subchains.

This scannability rule is omitted for existing scan segments that have the “mux_si_functional_source” attribute set.

By default, an S6 violation is reported as an Error, the system mode remains as setup, and the following error message displays:

```
//  Error: There were <int> scannability rule S6 fails (driven scan-in pin
//   of a sub chain or of a scan cell in unspecified scan chain).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can optionally choose to change the handling of this rule to Warning. In this case, the tool allows the system mode to change to analysis, but deletes subchain instances that violate the S6 rule and ignores scan cells that violate the S6 rule when stitching into scan chains during the scan chain generation process. The following warning message displays:

```
// Warning: There were <int> scannability rule S6 fails (driven scan-in
//  pin of a sub chain or of a scan cell in unspecified scan chain).
```

You can optionally choose to change the handling of this rule to Note. The severity changes from Warning to Note, but the handling remains the same (except when you are in the stand-alone dft -test_points context). The following message displays:

```
//  Note: There were <int> scannability rule S6 fails (driven scan-in
//  pin of a sub chain or of a scan cell in unspecified scan chain).
```

In the stand-alone dft -test_points context, S6 violations for scan elements (but not scan segments) are completely suppressed and the tool considers the memory element controllable and observable during shift.

Note

The S6 violations require manual intervention later in the flow.

You can use the report_drc_rules command to list the details of an S6 rule violation.

When scan cells produce S6 violations, you can optionally fix them by explicitly declaring the scan chains or subchains that the identified scan cells belong to using the add_scan_chains command. The tool uses this information to validate the traceability of declared scan chains or subchains when a test procedure file containing load_unload and shift procedures is present.

You can optionally declare the scan cells violating the S6 rule to be non-scan instances using the add_nonscan_instances command. In this case, the tool ignores these cells when stitching scan chains during the scan chains generation process.

S7

Category: Scannability

Contexts Supported: dft -scan, dft -test_points

Default Handling: Note

report_drc_rules: Supported

Scannability rule S7 identifies potential scan cells with a constant data input.

During scan insertion, memory elements with input signals tied to 0 or 1 are treated as scannable cells, which improves stuck-at fault coverage. However, these paths may not be optimized for timing, and are not used in functional mode. By default, all cells with constant input values are identified but are not excluded from scan insertion unless the constant value is

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

in the clock control logic (that is, the output of the flop is in the fanin cone of a clock or set/reset port of another flop).

The data inputs are checked for a constant 1 or 0 value. Flops failing the S1 or S2 rules are treated as non-scannable cells and are not reported as part of the S7 DRC report.

Tied values do not propagate through sequential elements unless a test_setup procedure is used to pulse the relevant clocks.

You can specify how these potential scan cells are handled by setting S7 to one of the following handling options:

- **Error** — Identifies cells with constant data inputs, and stops after identifying all DRC violations.
- **Warning** — Identifies cells, reports violations, and marks these cells as non-scan.
- **Note** — Identifies cells, reports violations, but allows these cells to become scan cells. This is the default DRC handling of rule S7. Maintains backward compatibility.
- **Ignore** — Does not check for constant data input.

For each DRC violation, the instance name of the scan cell, the affected data-input pin, and the constant value that was identified are reported. You may use the analyze_drc_violation command to graphically display the failing gate along with the sensitization values that led to the violation. This can be used to trace backward from the data input pin of the flop and identify the source of the constant value that triggered this violation.

S8

Category: Scannability

Contexts Supported: dft -scan, dft -test_points

Default Handling: Error

report_drc_rules: Supported

Scannability rule S8 identifies the OCC circuits that control the clocks for all of the scan elements in any scan mode, and then it verifies that the OCC control scan segments that are needed to operate the clocks for the scan elements within in each scan mode are also part of the population for that mode.

When an OCC segment is omitted from a scan mode population that also includes some of scan elements clocked by its OCC, it is reported as an S8 violation. By default, an S8 violation is reported as an Error, and the following error message displays:

```
//  Error: There was 1 S8 violation (OCC scan segment not included in a
//  scan mode's population that contains scan elements clocked by
//  its OCC).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can optionally choose to change the handling of this rule to Warning using the set_drc_handling command. In this case, the following warning message displays:

set_drc_handling s8 warn
analyze_scan_chains
...
...
...

```
//  Warning: There was 1 S8 violation (OCC scan segment not included in
//  a scan mode's population that contains scan elements clocked by
//  its OCC)
```

...

If the OCC segment is not part of the population because it marked as is_non_scannable, then this is reported in the error message:

```
//  Error: OCC scan segment '/corea_rtl_tessent_sib_sti_inst/ltest_so'
//  is not included in the population of the unwrapped scan mode
//  'unwrapped' which contains scan elements clocked by its OCC.
//  Note that the OCC scan segment
//  '/corea_rtl_tessent_sib_sti_inst/ltest_so' is non scannable for this
//  reason: user_specified. (S8-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Scan-Wrapper Rules (SW Rules)

Scan-wrapper DRC rules identify issues created during wrapper analysis, including identification of pre-existing dedicated wrapper cells and placement of new dedicated wrapper cells.

SW1 3549
SW2 3550
SW3 3550
SW4 3550
SW5 3551
SW6 3552
SW7 3552

SW1

Category: Scan-Wrapper

Contexts Supported: dft -scan

Default Handling: Warning

report_drc_rules: Supported

Scan-wrapper rule SW1 checks that the associated scan element of existing dedicated wrapper cells is usable.

All messages start with:

```
Ignoring identified pre-existing dedicated wrapper cell 'instanceName'.
```

SW1 messages by sub-type:

- Nonscan cell:

  ```
  It cannot be used as a dedicated wrapper cell because it is
  identified as non-scannable.
  ```

- Non-usable scan element child of parent scan element:

  ```
  It cannot be used as a dedicated wrapper cell because it belongs to
  a parent_type 'parent_name'.
  ```

- Multibit cell:

  ```
  It cannot be used as a dedicated wrapper cell because it is part of
  a multibit scan element.
  ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SW2

Category: Scan-Wrapper

Contexts Supported: dft -scan

Default Handling: Warning

report_drc_rules: Supported

Scan-wrapper rule SW2 checks that you do not specify to add a new dedicated wrapper cell to a port that has a pre-existing dedicated wrapper cell.

The occurrence message is:

```
Ignoring user specifications to add a new dedicated wrapper cell to
primary input 'in1' as it has a pre-existing dedicated wrapper cell
(ts_diw_301sffp1_i).
```

SW3

Category: Scan-Wrapper

Contexts Supported: dft -scan

Default Handling: Warning

report_drc_rules: Supported

Scan-wrapper rule SW3 checks that an existing dedicated wrapper cell is associated with at least one primary input or output..

The occurrence message is:

```
Tool identified an input dedicated wrapper cell 'ts_diw_301sffp1_i' that
is not associated with any primary input. This may cause additional
wrapper cells to be added to the associated port if it is not excluded
from wrapper analysis.
```

SW4

Category: Scan-Wrapper

Contexts Supported: dft -scan

Default Handling: Warning

report_drc_rules: Supported

Scan-wrapper rule SW4 checks that an existing dedicated wrapper cell covers all branches of the primary input (or output) that it wraps.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

An example SW4 message is:

```
Tool identified pre-existing dedicated wrapper cell 'ts_diw_301sffp1_i'
for primary input 'in1' does not cover all the fan-out branches. No new
dedicated wrapper cells will be placed on this inputOrOutput and it will
remain partially unwrapped.
```

SW5

Category: Scan-Wrapper

Contexts Supported: dft -scan

Default Handling: Warning

report_drc_rules: Supported

Scan-wrapper rule SW5 checks that the pre-existing dedicated wrapper cell found in the design matches the specification found in the relevant TCD file.

SW5 messages by sub-type:

- SPECIFIED_PORT_NOT_FOUND:

  ```
  Could not find primary input or primary output 'my_port' specified
  in the TCD file with at least one pre-existing dedicated wrapper
  cell.
  This specification will be ignored.
  ```

- SPECIFIED_PORT_HAS_NO_CELLS:

  ```
  Could not find any pre-existing dedicated wrapper cells for primary
  input 'in1' specified in the TCD file with at least one pre-existing
  dedicated wrapper cell.
  This specification will be ignored.
  ```

- SPECIFIED_CELL_NOT_FOUND:

  ```
  Could not find the instance 'ts_diw_301sffp1_i' specified in the TCD
  file that corresponds to a pre-existing dedicated wrapper cell for
  primary input 'in1'.
  This specification will be ignored.
  The following pre-existing dedicated wrapper cell instances were
  identified for this inputOrOutput: identifiedInstances
  ```

- SPECIFIED_BEHAVIOR_DOES_NOT_MATCH:

  ```
  The specified behavior 'shift' for 'ts_diw_301sffp1_i' does not
  match the identified behavior 'hold' for the pre-existing dedicated
  wrapper cell located at primary input 'in1'.
  This cell will be used during wrapper analysis with the identified
  behavior.
  ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- SPECIFIED_CELL_DOES_NOT_MATCH:

```
The identified input dedicated wrapper cell 'ts_diw_301sffp1_i'
could not be associated with the specified port 'in1'.
This may cause additional wrapper cells to be added to the
associated port if it is not excluded from wrapper analysis.
```

SW6

Category: Scan-Wrapper

Contexts Supported: dft -scan

Default Handling: Warning

report_drc_rules: Supported

Scan-wrapper rule SW6 checks that each pre-existing dedicated wrapper cell found in the design is specified in the relevant TCD file.

This is an example SW6 message:

```
Tool found pre-existing  dedicated wrapper cell 'ts_diw_301sffp1_i' at
primary input 'in1'.
The cell is not specified in the TCD file.
This input is assumed to be wrapped and no new dedicated wrapper cell will
be added.
```

SW7

Category: Scan-Wrapper

Contexts Supported: dft -scan

Default Handling: Error

report_drc_rules: Supported

Scan-wrapper rule SW7 checks that there is a valid location (or locations) for the tool to place a dedicated wrapper cell (or cells), such that it complies with all user specifications and tool requirements.

SW7 messages by sub-type:

- -cluster_isolation:

```
Could not find a valid location for a dedicated wrapper
cell for primary inputOrOutput ‘ioName’. The dedicated wrapper cell
cannot be placed directly on the port due to user specifications.

It is not possible to cover all branches of the port
with dedicated wrapper cells that each only drive scan elements with
the same cluster_name attribute (as required by the
'set_dedicated_wrapper_cell_options -isolate_clusters' command).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- -below_instances:

```
Could not find a valid location for a dedicated wrapper
cell for primary inputOrOutput ‘ioName’. The dedicated wrapper cell
cannot be placed directly on the port due to user specifications.

It is not possible to cover all branches of the port
with dedicated wrapper cells that are below any of the following
instances specified by the ‘set_dedicated_wrapper_cell_options
-below_instances’ command: <list of instances specified by the user
for the port or the default>
```

- POWER_ISOLATION:

```
Could not find a valid location for a dedicated wrapper cell for
primary input 'in1'.
The dedicated wrapper cell cannot be placed directly on the port due
to user specifications.
It is not possible to cover all branches of the port with dedicated
wrapper cells placed after power isolation cells.
Placement failed at or near 'my_comb_gate'.
```

- TESSENT_INSTRUMENT:

```
Could not find a valid location for a dedicated wrapper cell for
primary input 'in1'.
It is not possible to cover all branches of the port that require
dedicated wrapper cells without blocking branches that drive tessent
instruments (which must be controlled during both intest and
extest).
Placement failed at or near 'my_comb_gate'.
```

For all sub-types, the tool adds the following text in special cases:

- If it is a set/reset port:

```
This port is a set/reset port. All dedicated wrapper cells on this
port must be located before the hardware that is required to disable
set/reset values during shift.
The location of the hardware is specified by the DFT
async_set_reset_dynamic_disable signal.
```

- If the severity is set to Warning (that is, not Error):

```
No dedicated wrapper cell will be placed on this branch, which may
result in reduced coverage.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SSN ICL Rules (SSN_ICL Rules)

SSN ICL rules only apply to designs that incorporate SSN. They detect violations based on the ICL description of the SSN. SSN ICL rules do not detect problems in the design or inconsistencies between the design and the ICL.

SSN_ICL2 3554
SSN_ICL3 3555

SSN_ICL2

Category: SSN ICL

Contexts Supported: all contexts

Default Handling: Error

report_drc_rules: Supported

Ensures that the SSN scan hosts (SSHs) receive their stimulus data unaltered (without inversion) from the SSN datapath inputs, and that the response data from the SSHs is observed unaltered at the SSN datapath outputs.

There must not be any net inversions on the SSN datapath between any two SSHs, between the datapath inputs and the first SSH, or between the last SSH and the datapath outputs. Such a net inversion would result in alteration of the stimulus data received by the SSHs and of the response data observed at the datapath outputs. SSH_ICL2 identifies invalid inversion based on the ICL description of the SSN and reports all inversions that contribute to an invalid net inversion and their locations in the ICL files. It also reports the associated design pin or design instance, if available. If multiple inversions contribute to the invalid net inversion, the message contains multiple groups with the properties of the source of the individual inversions and the sink of the individual inversions.

The error message is:

```
// Error: There is a net inversion on one bit of the SSN bus between the
//        Scan Host ICL instance 'I' and the datapath ICL output
//        port 'O' in the Streaming Scan Network.
//        The details of the path in question are as follows
//        (input -> output):
//           Inversion 1 source
//             ICL instance output pin: P1
//             design pin:              P2
//             ICL file name:           F1
//             ICL file line number:    L1
//           Inversion 1 sink
//             ICL primary output port: P3
//             design port:             P4
//             ICL file name:           F2
//             ICL file line number:    L2
//            (SSN_ICL2-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

where I is an ICL instance name, O is an output port name, P<*n*> are port names, F<*n*> are filenames, and L<*n*> are line numbers. F<*n*> may be "extracted ICL has not been written to a file", in which case the ICL file line number line is omitted for that source or sink description.

Note

This check uses only the ICL description of the SSN. It does not detect problems in the design or inconsistencies between the design and the ICL. These problems can result from situations such as broken synthesis after the ICL has already been extracted from the RTL.

SSN_ICL3

Category: SSN ICL

Contexts Supported: all contexts

Default Handling: Error

report_drc_rules: Supported

Ensures that bus bits between SSN building blocks are connected in the correct order (that is, bits 0 to 0, 1 to 1, and so forth). The bits must not swap their order between the SSN building blocks (for example, bit 0 on one connecting to bit 3 on another).

The tool checks the connectivity between the following types of SSN building blocks:

- Scan host (SSH)
- Bus frequency divider (BFD)
- Bus frequency multiplier (BFM)

You can then connect the checked elements with each other directly or through pipelines, FIFOs, or SSN datapath multiplexers between them.

The error message is:

```
//  Error: The Streaming Scan Network contains two building blocks which
//         are connected by an SSN bus with inconsistent bit order.
//         (SSN_ICL3-1)
//         The details of one of the individual connections with
//         mismatching bus indices are shown below. The SSN bus, which
//         this connection is part of, contains N connections with
//         mismatching bus indices in total.
//           Source:
//             Type                 : T1
//             Bus output pin       : P1
//             Bus output pin (ICL): P1ICL
//           Sink:
//             Type                 : T2
//             Bus input pin        : P2
//             Bus input pin (ICL) : P2ICL
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

where N is an integer count, T<*n*> are source/sink types. P<*n*> are pins, and P<*n*>ICL are pins in the ICL.

Note

This check uses only the ICL description of the SSN. It does not detect problems in the design or inconsistencies between the design and the ICL. These problems can result from situations such as broken synthesis after the ICL has already been extracted from the RTL. You can detect opens, shorts, and connection errors in the datapath that can be found only in the design and not the ICL by generating and simulating the SSN continuity patterns. For more information about SSN continuity patterns, refer to the section "SSN Debug on the Tester" in the *Tessent Shell User's Manual*.

Examples

```
//  Error: The Streaming Scan Network contains two building blocks which
//         are connected by an SSN bus with inconsistent bit order.
//         (SSN_ICL3-1)
//         The details of one of the individual connections with
//         mismatching bus indices are shown below.
//         The SSN bus, which this connection is part of, contains 4
//         connections with mismatching bus indices in total.
//           Source:
//             Type                : SSN scan host
//             Bus output pin      : corea_i2/
//       corea_rtl2_tessent_ssn_scan_host_1_inst/bus_data_out[3]
//             Bus output pin (ICL):
//       corea_i2.corea_rtl2_tessent_ssn_scan_host_1_inst.bus_data_out[3]
//           Sink:
//             Type                : SSN scan host
//             Bus input pin       : corea_i1/
//       corea_rtl2_tessent_ssn_scan_host_1_inst/bus_data_in[0]
//             Bus input pin (ICL) :
//       corea_i1.corea_rtl2_tessent_ssn_scan_host_1_inst.bus_data_in[0]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SSN Retargeting Rules (SSN_R Rules)

SSN retargeting rules only apply to designs that incorporate SSN.

The following subsections describe each of the SSN retargeting rules and the special handling you can set for them.

SSN_R1 3557
SSN_R2 3558
SSN_R3 3558
SSN_R4 3559
SSN_R5 3562
SSN_R6 3564
SSN_R7 3565
SSN_R8 3567
SSN_R9 3570
SSN_R10 3571
SSN_R11 3573
SSN_R12 3576
SSN_R13 3578
SSN_R15 3579
SSN_R16 3580
SSN_R17 3581
SSN_R18 3582
SSN_R19 3584
SSN_R20 3585
SSN_R21 3586
SSN_R23 3586

SSN_R1

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Default Handling: Error

report_drc_rules: Supported

This DRC verifies for each SSH ICL module that the Tessent Core Description for that instrument is in memory.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following lists the checks and violations that may occur:

- Unable to find a matching design instance for SSH ICL instance XXX. No persistent buffers corresponding to the boundary of this instance were found either.
- Cannot find the TCD for SSH ICL module XXX in memory or in an opened TSDB.
- The TCD of core XXX matching SSH ICL module YYY is not of type SSH.
- The ungrouped SSH design instance matching SSH ICL instance XXX is missing one or more persistent buffers.

If the TCD for the instrument is not in memory, the tool attempts to auto-load the TCD into memory. If this fails, then the tool issues an error. The default is an error, but you can downgrade the error to a warning, a note, or an ignore. If downgraded, the tool ignores in subsequent steps any SSH ICL module without a TCD.

SSN_R2

Category: SSN

Contexts Supported: patterns -scan_retargeting

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Ensures the ICL of current design is available when its TCD includes SSH instances.

When doing retargeting in non-extraction flow, the SSH ICL instances are inferred from the current design TCD file. This rule ensures the ICL of current design is available when current design TCD includes SSH instances.

The following lists the check and violation that may occur:

- No ICL found for the current design 'XXX'. Read the current design ICL.

The error message is:

```
//  Error: No ICL found for the current design 'T'. Read the current
//  design ICL. (SSN_R2-1)
```

Where T is the design name.

SSN_R3

Category: SSN

Contexts Supported: patterns -scan_retargeting

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Ensures the SSH instances in TCD of the current design map to SSH ICL instances.

When doing retargeting in the non-extraction flow, the SSH instances are inferred from the current design TCD file. This rule ensure the tool can map each SSH instance inferred from TCD to its ICL instance.

The following lists the check and violation that may occur:

- Unable to find a matching ICL instance for the SSH ICL instance 'XXX' derived from the TCD of the current design. Read the correct current design ICL and/or TCD file(s).

The error message is:

```
//  Error: Unable to find a matching ICL instance for the SSH ICL instance
//  'I' derived from the TCD of the current design. Read the correct
//  current design ICL and/or TCD file(s). (SSN_R3-1)
```

Where I is the SSH ICL instance name.

Examples

```
Error: Unable to find a matching ICL instance for the SSH ICL instance
'coreA_i2.coreA_ijtag_ssn_edt_occ_insertion_tessent_ssn_scan_host_1_inst'
derived from the TCD of the current design. Read the correct current
design ICL and/or TCD file(s). (SSN_R3-2)

Error: Unable to find a matching ICL instance for the SSH ICL instance
'coreA_i3.coreA_ijtag_ssn_edt_occ_insertion_tessent_ssn_scan_host_1_inst'
derived from the TCD of the current design. Read the correct current
design ICL and/or TCD file(s). (SSN_R3-3)

Error: Unable to find a matching ICL instance for the SSH ICL instance
'coreA_i4.coreA_ijtag_ssn_edt_occ_insertion_tessent_ssn_scan_host_1_inst'
derived from the TCD of the current design. Read the correct current
design ICL and/or TCD file(s). (SSN_R3-4)

Error: There were 4 SSN_R3 violations (SSH instances in TCD of the current
design do not map to SSH ICL instances).
```

SSN_R4

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Checks that the connections between the EDT or uncompressed scan chains connect only to a single SSH.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following lists the checks and the violations that may occur:

- If any channel of an EDT instance connects to an SSH instance, all channels of that instance must connect to that SSH instance. There cannot be a mix of SSH and non-SSH connections to the EDT instance.

  The error message is:

  ```
  //  Error: SSN is active but EDT instance 'edt_inst'
  //         is not correctly connected to a single SSH
  //         instance. (SSN_R4-1)
  ```

  Where ‘edt_inst’ is the EDT instance name.

- For uncompressed chains, the scan chain should be connected to a single SSH instance.

  The error message is:

  ```
  //  Error: SSN is active but scan chain 'chain_1'
  //         is not correctly connected to a single SSH
  //         instance. (SSN_R4-1)
  ```

  Where ‘chain_1’ is a scan chain name.

Examples

Example 1

The following is an example of an EDT instance connected to two SSH instances:

```
// Error: SSN is active but EDT instance 'edt_inst' is not correctly
//        connected to a single SSH instance. (SSN_R4-1)
//
//        EDT Input Channel Pin  : edt_inst/edt_channels_in[0]
//        SSH Output Pin         : ssn_2_inst/to_scan_in[1]
//
//        EDT Input Channel Pin  : edt_inst/edt_channels_in[1]
//        SSH Output Pin         : ssn_1_inst/to_scan_in[1]
//
//        EDT Output Channel Pin : edt_inst/edt_channels_out[0]
//        SSH Input Pin          : ssn_1_inst/from_scan_out[0]
//
//        EDT Output Channel Pin : edt_inst/edt_channels_out[1]
//        SSH Input Pin          : ssn_1_inst/from_scan_out[1]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following is an example of an EDT instance not connected to a single SSH instance. Some of the input and output channels connect to top-level ports:

```
// Error: SSN is active but EDT instance 'edt_inst' is not correctly
//        connected to a single SSH instance. (SSN_R4-1)
//
//        EDT Input Channel Pin  : edt_inst/edt_channels_in[0]
//        Port                   : YZ_ch_in[0]
//
//        EDT Input Channel Pin  : edt_inst/edt_channels_in[1]
//        Port                   : YZ_ch_in[1]
//
//        EDT Input Channel Pin  : edt_inst/edt_channels_in[2]
//        SSH Output Pin         : ssn_inst/to_scan_in[2]
//
//        EDT Input Channel Pin  : edt_inst/edt_channels_in[3]
//        SSH Output Pin         : ssn_inst/to_scan_in[3]
//
//        EDT Output Channel Pin : edt_inst/edt_channels_out[0]
//        SSH Input Pin          : ssn_inst/from_scan_out[0]
//
//        EDT Output Channel Pin : edt_inst/edt_channels_out[1]
//        SSH Input Pin          : ssn_inst/from_scan_out[1]
//
//        EDT Output Channel Pin : edt_inst/edt_channels_out[2]
//        Port                   : YZ_ch_out
//
```

Example 3

The following is an example of an EDT instance inside a core instance not connected to a single SSH instance:

```
// Error: SSN is active but EDT instance 'core_1/edt_inst' instantiated
//        inside core instance 'core_1' is connected to top level ports
//        instead of an SSH instance. (SSN_R4-1)
//
//        Core Instance Input Pin  : core_1/edt_channels_in[0]
//        Port                     : core_1_edt_channels_in[0]
//
//        Core Instance Input Pin  : core_1/edt_channels_in[1]
//        Port                     : core_1_edt_channels_in[1]
//
//        Core Instance Output Pin : core_1/edt_channels_out[0]
//        Port                     : core_1_edt_channels_out[0]
//
//        Core Instance Output Pin : core_1/edt_channels_out[1]
//        Port                     : core_1_edt_channels_out[1]
//
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 4

The following is an example of an uncompressed scan chain input not connected to the same SSH instance as the scan chain output:

```
// Error: SSN is active but scan chain 'chain_1' is not
//        correctly connected to a single SSH instance. (SSN_R4-1)
//
//        Scan Chain Input Pin  : /chain_1/ts_si[0]
//        SSH Output Pin        : Not connected to SSH instance
//        Scan Chain Output Pin : ssn_inst/tessent_persistent_cell_from_scan_out_0_and/A0
//        SSH Input Pin         : ssn_inst/from_scan_out[0]
//
```

SSN_R5

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Default Handling: Warning (can be changed to Error, Note, or Ignore)

report_drc_rules: Supported

Verifies the presence of a pulse_always clock with internal capture SSN. An example of a pulse_always clock is a free-running PLL fast clock that is a direct clock source to a flip-flop. When this DRC is not an error, the tool simulates all pulse_always clocks in test_setup/test_end, and as X-sources during load_unload and capture. All pulse_always clocks are still pulsed in the pattern file and simulation testbench.

The following lists the checks and the violations that may occur:

- During SSN internal capture mode:
  - The warning message is:

```
// Warning: The following pulse-always clocks will be
//          modeled as X sources during load_unload
//          and capture because SSN in internal
//          capture mode is present: (SSN_R5-1)
// 'pa_clk1'
// 'pa_clk2'
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The error message is:

```
// Error: Use of pulse-always clocks with SSN in internal
//        capture mode ("set_current_mode<name> -type internal")
//        may cause pattern mismatches for the reasons listed
//        below: (SSN_R5-1)
//           - The period of the clocks may vary between phases
//             of test (e.g., test_setup vs. shift) since each
//             typically operates its clocks at a different
//             frequency.
//           - The number of clock pulses observed by ATPG does
//             not match the number of clock pulses applied by
//             the tester since ATPG cycles do not match tester
//             cycles.
//           - The tester cycles match the SSN shift clock rate,
//             which is typically faster than the internal
//             capture clocks.
//        The following clock(s) were defined as pulse-always
//        clocks:
//        'pa_clk1'
//        'pa_clk2'
//        Downgrade the DRC if you considered all of the above
//        cases and would like the tool to model pulse-always
//        clocks as X sources during load_unload and capture.
//        They will continue to be handled normally during DRC
//        simulation of test_setup/test_end.
//        Note that they will be treated as X's by DRC and ATPG,
//        but the clocks will still be pulsed in the pattern
//        file and simulation testbench.
//        Alternatively, add those clocks as asynchronous
//        free-running clocks by using the "-period" switch
//        of "add_clocks".
// Error: There was 1 SSN_R5 violation (Presence of
//        pulse-always clocks with SSN in internal capture mode).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- During SSN ATPG retargeting mode:
  - The error message is:

```
// Error: Use of pulse-always clocks with SSN in internal
//        capture mode may cause pattern mismatches for the
//        reasons listed below: (SSN_R5-1)
//           - The period of the clocks may vary between phases
//             of test (e.g., test_setup vs. shift) since each
//             typically operates its clocks at a different
//             frequency.
//           - The tester cycles match the SSN shift clock rate,
//             which is typically faster than the internal
//             capture clocks.
//        The following clock(s) were defined as pulse-always
//        clocks:
//        'pa_clk1'
//        'pa_clk2'
//        Downgrade the DRC if you considered all of the above
//        cases and would like the tool to model pulse-always
//        clocks as X sources during load_unload and capture.
//        They will continue to be handled normally during DRC
//        simulation of test_setup/test_end.
//        Note that they will be treated as X's by DRC,
//        but the clocks will still be pulsed in the pattern
//        file and simulation testbench.
//        Alternatively, add those clocks as asynchronous
//        free-running clocks by using the "-period" switch
//        of "add_clocks".
// Error: There was 1 SSN_R5 violation (Presence of
//       pulse-always clocks with SSN in internal capture mode).
```

SSN_R6

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Checks that SSH instances have a corresponding PDL setup iProc in memory.

The following lists the check and the violation that may occur:

- Check if all active SSH instances have their corresponding PDL setup proc in memory.

  The error message is:

```
// Error: Unable to find the 'setup' iProc in memory for active SSH
//             ICL instance 'ssn_1_inst' (ICL module 'ssn_1').
//             Please source the PDL file that includes this
//             iProc. (SSN_R6-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Where ‘ssn_1_inst’ is the SSH instance name.

SSN_R7

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Checks the edge timing between the SSH outputs and the compressed EDT channel inputs, the uncompressed scan chain inputs, or the cores during shift.

The following lists the checks per chain group and the violations that may occur:

- The capture edge of all SSH to_scan_in connections, either compressed channel inputs or uncompressed chain inputs, must be clocked by a leading-edge clock.

  The error message is:

  ```
  // Error: The connection from an SSH, to either a decompressor or
  //        uncompressed chains, should be clocked by a leading
  //        edge clock. (SSN_R7-1)
  //        The 'to_scan_in' port of the SSH instance 'S'
  //        is not driving leading edge elements.
  ```

  Where 'S' is the SSH instance name.

- The capture edge of all SSH to_scan_in connections to core instances must be clocked by a leading-edge clock.

  The error message is:

  ```
  //  Error: The connection from an SSH, to core instance 'C',
  //         should be clocked by a leading edge clock. (SSN_R7-1)
  //         The 'to_scan_in' port of the SSH instance 'S' chain
  //         group 'G' is not driving leading edge elements.
  ```

  Where 'C' is the core instance name, 'S' is the SSH instance name, and 'G' is the chain group name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following is an example of the rule violation where the SSH outputs are connected directly to the EDT inputs:

```
// Error: The connection from an SSH, to either a decompressor or
//        uncompressed chains, should be clocked by a leading
//        edge clock. (SSN_R7-1)
//        The 'to_scan_in' port of the SSH instance 'ssh_inst' is
//        not driving leading edge elements.
//
//        SSH Output Pin           : ssh_inst/to_scan_in[0]
//        EDT Input Channel Pin    : edt_inst/edt_channels_in[0]
//        EDT Input Channel Timing : LE
//
//        SSH Output Pin           : ssh_inst/to_scan_in[1]
//        EDT Input Channel Pin    : edt_inst/edt_channels_in[1]
//        EDT Input Channel Timing : LE
//
//        SSH Output Pin           : ssh_inst/to_scan_in[2]
//        EDT Input Channel Pin    : edt_inst/edt_channels_in[2]
//        EDT Input Channel Timing : LE
//
//        SSH Output Pin           : ssh_inst/to_scan_in[3]
//        EDT Input Channel Pin    : edt_inst/edt_channels_in[3]
//        EDT Input Channel Timing : TE
```

Example 2

The following is an example where there are one or more pipeline stages between the SSH outputs and the EDT inputs:

```
// Error: The connection from an SSH, to either a decompressor or
//        uncompressed chains, should be clocked by a leading
//        edge clock. (SSN_R7-1)
//        The 'to_scan_in' port of the SSH instance 'ssh_inst' is
//        not driving leading edge elements.
//
//        SSH Output Pin                           : ssh_inst/to_scan_in[0]
//        EDT First Channel Input Pipeline Instance : edt_chan_in_0_pipe_inst_0
//        EDT First Channel Input Pipeline Timing   : LE
//
//        SSH Output Pin                           : ssh_inst/to_scan_in[1]
//        EDT First Channel Input Pipeline Instance : edt_chan_in_1_pipe_inst_0
//        EDT First Channel Input Pipeline Timing   : LE
//
//        SSH Output Pin                           : ssh_inst/to_scan_in[2]
//        EDT First Channel Input Pipeline Instance : edt_chan_in_2_pipe_inst_0
//        EDT First Channel Input Pipeline Timing   : LE
//
//        SSH Output Pin                           : ssh_inst/to_scan_in[3]
//        EDT First Channel Input Pipeline Instance : edt_chan_in_3_pipe_inst_0
//        EDT First Channel Input Pipeline Timing   : TE
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

The following is an example where there is a pipeline between the SSH and the core input:

```
//  Error: The connection from an SSH, to core instance 'corea', should be
//         clocked by a leading edge clock. (SSN_R7-1)
//         The 'to_scan_in' port of the SSH instance 'ssh' chain group
//         'edt1' is not driving leading edge elements.
//
//         SSH Output Pin                      :  ssh/to_scan_in_edt1[0]
//         Core Input Pin                      :  corea/my_edt_channel_in[0]
//         Core Input Timing                   :  LE
//
//         SSH Output Pin                      :  ssh/to_scan_in_edt1[1]
//         Core Input Pin                      :  corea/my_edt_channel_in[1]
//         Core Input Timing                   :  LE
//
//         SSH Output Pin                      :  ssh/to_scan_in_edt1[2]
//         Core Input Pin                      :  corea/my_edt_channel_in[2]
//         Core Input Timing                   :  LE
//
//         SSH Output Pin                      :  ssh/to_scan_in_edt1[3]
//         Core Input Pin                      :  corea/my_edt_channel_in[3]
//         Core Input Timing                   :  LE
//
//         SSH Output Pin                      :  ssh/to_scan_in_edt1[4]
//         Core Input First Pipeline Instance  :  Delay1
//         Core Input First Pipeline Timing    :  TE
//
//         SSH Output Pin                      :  ssh/to_scan_in_edt1[5]
//         Core Input Pin                      :  corea/to_scan_in[0]
//         Core Input Timing                   :  LE
//
//         SSH Output Pin                      :  ssh/to_scan_in_edt1[6]
//         Core Input Pin                      :  corea/to_scan_in[1]
//         Core Input Timing                   :  LE
```

SSN_R8

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Checks the edge timing between the compressed EDT channel outputs or the uncompressed scan chain outputs and the SSH inputs during shift.

The following lists the checks per chain group and the violations that may occur:

- The SSH must have the necessary hardware to enable chain groups to strobe on the leading edge of the scan clock in the case of mixed leading- and trailing-edge clocks clocking the scan chains or channels driving the SSH.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The error message is:

```
//  Error: The 'from_scan_out' port of the SSH instance 'S'
//         is driven by a mixture of leading and
//         trailing edge clocks. (SSN_R8-1)
//         Cannot configure the SSH to strobe on the leading edge
//         since the hardware does not support it.
//         Regenerate the SSH, and ensure that the
//         support_from_scan_out_le_strobing property of the SSH
//         in the DFT Spec is not set to 'off'.
```

Where 'S' is the SSH instance name.

- The SSH must have the necessary hardware to enable chain groups to strobe on the leading edge of the scan clock in the case of leading-edge clocks clocking the scan chains or channels driving the SSH.

  The error message is:

```
//  Error: The 'from_scan_out' port of the SSH instance 'S'
//         is driven by a leading edge clock. (SSN_R8-1)
//         Cannot configure the SSH to strobe on the leading edge
//         since the hardware does not support it.
//         Regenerate the SSH, and ensure that the
//         support_from_scan_out_le_strobing
//         property of the SSH in the DFT Spec is not set to 'off'.
```

  Where 'S' is the SSH instance name.

- The SSH must have the necessary hardware to enable chain groups to strobe on the leading edge of the scan clock in the case where the from_scan_out port of the SSH is driven by a combination of leading- and trailing-edge clocks.

  The error message is:

```
//  Error: The 'from_scan_out' port of the SSH instance 'S' chain
//         group 'C' is driven by a mixture of leading and
//         trailing edge clocks. (SSN_R8-1)
//         Cannot configure the SSH to strobe on the leading edge
//         since the hardware does not support it.
//         Regenerate the SSH, and ensure that the
//         support_from_scan_out_le_strobing property of the SSH in
//         the DFT Spec is not set to 'off'.
```

  Where 'S' is the SSH instance name and 'C' is the chain group name.

- The SSH should strobe all chains or channels on the trailing edge if all chains or channels are clocked on the trailing edge.

  There will be no error message in this case.

Note

In this case, it is optional to turn off the support_from_scan_out_le_strobing property of the SSH. See “ScanHost” on page 4484 for details.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following is an example where both leading- and trailing-edge clocks are present in the connections from the EDT to the SSH and the SSH hardware does not have the LE strobing capability:

```
//  Error: The 'from_scan_out' port of the SSH instance 'ssh_inst' is
//         driven by a mixture of leading
//         and trailing edge clocks. (SSN_R8-1)
//         Cannot configure the SSH to strobe on the leading edge since
//         the hardware does not support it.
//         Regenerate the SSH, and ensure that the
//         support_from_scan_out_le_strobing property of the SSH
//         in the DFT Spec is not set to 'off'.
//
// SSH Input Pin                            :  ssh_inst/from_scan_out[0]
// EDT Channel Output Last Pipeline Instance:  edt_ch_out_0_lup_inst_last
// EDT Channel Output Last Pipeline Timing  :  TE
//
// SSH Input Pin                            :  ssh_inst/from_scan_out[1]
// EDT Channel Output Last Pipeline Instance:  edt_ch_out_1_lup_inst_last
// EDT Channel Output Last Pipeline Timing  :  TE
//
// SSH Input Pin                            :  ssh_inst/from_scan_out[2]
// EDT Channel Output Last Pipeline Instance:  edt_ch_out_2_lup_inst_last
// EDT Channel Output Last Pipeline Timing  :  LE
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following is an example where both leading- and trailing-edge clocks are present in the connections from core outputs to the SSH and the SSH hardware does not have the LE strobing capability:

```
//  Error: The 'from_scan_out' port of the SSH instance 'ssh' chain group 'edt1'
//         is driven by a mixture of leading and trailing edge clocks. (SSN_R8-1)
//         Cannot configure the SSH to strobe on the leading edge since the
//         hardware does not support it.
//         Regenerate the SSH, and ensure that the
//         support_from_scan_out_le_strobing property of the SSH in the DFT Spec
//         is not set to 'off'.
//
//         SSH Input Pin                       :  ssh/from_scan_out_edt1[6]
//         Core Output Pin                     :  corea/from_scan_out[0]
//         Core Output Timing                  :  TE
//
//         SSH Input Pin                       :  ssh/from_scan_out_edt1[5]
//         Core Output Pin                     :  corea/from_scan_out[1]
//         Core Output Timing                  :  TE
//
//         SSH Input Pin                       :  ssh/from_scan_out_edt1[0]
//         Core Output Last Pipeline Instance  :  Delay2
//         Core Output Last Pipeline Timing    :  LE
//
//         SSH Input Pin                       :  ssh/from_scan_out_edt1[1]
//         Core Output Last Pipeline Instance  :  Delay3
//         Core Output Last Pipeline Timing    :  LE
```

SSN_R9

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Checks the edge timing between the SSH inputs and the compressed EDT channel outputs, the uncompressed scan chain outputs, or core outputs during shift.

The following lists the checks per chain group and violations that may occur:

- If the SSH from_scan_out connections are clocked by a mixture of leading- and trailing-edge clocks, check if the SSH LE strobing capability is supported. See “ScanHost” on page 4484 for details on support_from_scan_out_le_strobing.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The warning messages are:

```
// Warning: The 'from_scan_out' port of the SSH instance 'S'
//          is driven by a mixture of leading and trailing edge
//          clocks.(SSN_R9-1)
//          The SSH strobing for this chain group will be done on
//          the leading edge which will only provide half a clock
//          period for the loop delay.
```

and:

```
// Warning: The 'from_scan_out' port of the SSH instance 'S' chain
//          group 'C' is driven by a mixture of leading and
//          trailing edge clocks. (SSN_R9-1)
//          The SSH strobing for this chain group will be done on
//          the leading edge which will only provide half a clock
//          period for the loop delay.
```

Where 'S' is the SSH instance name and 'C' is the chain group name.

Examples

Example 1

The following is an example where both leading- and trailing-edge clocks are present in the connections from the EDT to the SSH and the SSH hardware supports the LE strobing capability:

```
//  Warning: The 'from_scan_out' port of the SSH instance 'ssh_inst'
//           is driven by a mixture of leading and trailing
//           edge clocks. (SSN_R9-1)
//           The SSH strobing for this chain group will be done on the
//           leading edge which will only provide half a clock
//           period for the loop delay.
//
// SSH Input Pin                              : ssh_inst/from_scan_out[0]
// EDT Channel Output Last Pipeline Instance : edt_ch_out_0_lup_inst_last
// EDT Channel Output Last Pipeline Timing   : TE
//
// SSH Input Pin                              : ssh_inst/from_scan_out[1]
// EDT Channel Output Last Pipeline Instance : edt_ch_out_1_lup_inst_last
// EDT Channel Output Last Pipeline Timing   : TE
//
// SSH Input Pin                              : ssh_inst/from_scan_out[2]
// EDT Channel Output Last Pipeline Instance : edt_ch_out_2_lup_inst_last
// EDT Channel Output Last Pipeline Timing   : LE
```

SSN_R10

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Checks that the values of the calculated parameters of an active SSH do not exceed the corresponding IJTAG register sizes.

The following lists the check per IJTAG register and violation that may occur:

- If the calculated parameter value is larger than the corresponding IJTAG register size.

  The error message is:

  ```
  //  Error: The following parameter of the SSH core
  //         'tessent_ssn_scan_host_1' exceeds the maximum
  //         allowed value: (SSN_R10-1)
  ```

  Where ‘tessent_ssn_scan_host_1’ is the SSH core name.

Examples

Example 1

The following is an example where multiple parameters have a value larger than the permitted value:

```
//  Error: The following parameter of the SSH core
//         'tessent_ssn_scan_host_1' exceeds the maximum
//         allowed value: (SSN_R10-1)
//
// Parameter Name  Parameter Value  Parameter Max Value  ScanHost DFT Specification Property
// --------------  ---------------  -------------------  -----------------------------------
// bits_per_packet        2                  1                   max_packet_size
// initial_offset         3                  1                   max_initial_offset
//
// The affected active SSH instance is 'tessent_ssn_scan_host_1_inst'.
//
// Regenerate the hardware and set the DFT Specification property to a
// sufficiently large value.
```

Example 2

The following is an example where multiple parameters have a value larger than the permitted value for multiple SSH instances:

```
//  Error: The following parameter of the SSH core
//         'tessent_ssn_scan_host_1' exceeds the maximum
//         allowed value: (SSN_R10-1)
//
// Parameter Name  Parameter Value  Parameter Max Value  ScanHost DFT Specification Property
// --------------  ---------------  -------------------  -----------------------------------
// initial_offset        15                 7                   max_initial_offset
// packet_size           10                 1                   max_packet_size
//
// One of the affected active SSH instances is 'corea_i1/tessent_ssn_scan_host_1_inst'.
// To list all instances of this SSH core, issue 'report_core_instances -core
// { tessent_ssn_scan_host_1 }'.
//
// Regenerate the hardware and set the DFT Specification property to a
// sufficiently large value.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

In this case, the “Parameter Value” column reports the maximum value among all instances.

Example 3

The following is an example where one parameter has a value larger than the permitted value, and the SSH instance is ungrouped:

```
//  Error: The following parameter of the SSH core
//         'tessent_ssn_scan_host_1' exceeds the maximum
//         allowed value: (SSN_R10-1)
//
// Parameter Name  Parameter Value  Parameter Max Value  ScanHost DFT Specification Property
// --------------  ---------------  -------------------  -----------------------------------
// initial_offset        31                 15                   max_initial_offset
//
// The affected active SSH instance is 'tessent_ssn_scan_host_1_inst'(ungrouped).
//
// Regenerate the hardware and set the DFT Specification property to a
// sufficiently large value.
```

SSN_R11

Category: SSN

Contexts Supported: patterns -scan, patterns -scan-diagnosis, patterns -scan_retargeting

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Checks if SSH registers are large enough to satisfy desired timing requirements.

The following lists the checks and violations that may occur:

- Unable to satisfy capture clock period for SSH instance 'XXX'. The register that controls the clock division is too small to generate a PPP capture clock period from a TTT SSN bus clock. You can either decrease the capture clock period via 'set_capture_timing_options' if your design timing supports it, or you can regenerate the SSH hardware to support this capture clock period by setting the 'max_capture_to_shift_clock_period_ratio' property of the DftSpecification to at least *N*.
- Unable to satisfy the requested number of capture clock pulses coming from SSH 'XXX'. The register that controls the number of capture clock pulses is too small to generate *N* capture clock pulses. You can decrease the number of capture clock pulses via the core instance parameter 'capture_packets' to a maximum of *M*, or you can regenerate the SSH hardware to support this number of capture clock pulses setting the 'max_capture_clock_pulses' property of the DFT Specification to at least *L*.
- Unable to satisfy the requested scan_en setup and hold time for SSH instance 'XXX'. The register that controls the number of scan_en setup and hold cycles is too small. The number of scan_en setup extra cycles is *N* and the number of scan_en hold extra cycles

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is *M*. You can decrease the number of scan_en setup and/or hold extra cycles via: 'set_load_unload_timing_options -scan_en_setup_extra_cycles' and 'set_load_unload_timing_options -scan_en_hold_extra_cycles', or you can regenerate the SSH hardware to support this number of scan_en setup and hold cycles setting the 'max_scan_en_mcp' property of the DFT Specification to at least *L*.

- Unable to satisfy the requested edt_update setup and hold time for SSH instance 'XXX'. The register that controls the number of edt_update setup and hold cycles is too small. The number of edt_update setup extra cycles is *N* and the number of edt_update hold extra cycles is *M*. You can decrease the number of edt_update setup and/or hold extra cycles via: 'set_load_unload_timing_options -edt_update_setup_extra_cycles' and 'set_load_unload_timing_options -edt_update_hold_extra_cycles', or you can regenerate the SSH hardware to support this number of edt_update setup and hold cycles setting the 'max_edt_update_mcp' property of the DFT Specification to at least *L*.
- Unable to satisfy the requested scan_en setup and hold time for SSH instance 'XXX'. The register that controls the number of scan_en setup and hold cycles is too small. The number of scan_en setup extra cycles is *N* and the number of scan_en hold extra cycles is *M*. You can decrease the number of scan_en setup and/or hold extra cycles via: 'set_load_unload_timing_options -scan_en_setup_extra_cycles' and 'set_load_unload_timing_options -scan_en_hold_extra_cycles'. Supporting the current scan_en setup and hold cycles would require specifying the 'max_scan_en_mcp' property of the DFT Specification to at least *L*, but the largest allowed is *K*.
- Unable to satisfy the requested minimum capture window size in bus clock cycles of *N* for SSH instance 'XXX'. The following equation is used to calculate the minimum capture window size in bus clock cycles: (<scan_en_transition_packets> + (<capture_packets> * (<packets_per_capture_clock_minus_one> + 1))) * <bus clock cycles per packet> The bus clock cycles per packet for this SSH is *L*. The maximum capture window size in bus clock cycles for this SSH is *M*. You can decrease the number of minimum capture window size in bus clock cycles using the 'set_capture_timing_options -min_capture_window_in_ssn_bus_cycles' command, or you can regenerate the SSH hardware with larger 'max_scan_en_mcp', 'max_capture_clock_pulses' and/or 'max_capture_to_shift_clock_period_ratio' property values.

Examples

```
//  Error: Unable to satisfy capture clock period for SSH instance 'ssh'.
//  The register that controls the clock division is too small to
//  generate a 5000ns capture clock period from a 10ns SSN bus clock.
//  You can either decrease the capture clock period via
//  'set_capture_timing_options' if your design timing supports it,
//  or you can regenerate the SSH hardware to support this capture
//  clock period by setting the 'max_capture_to_shift_clock_period_ratio'
//  property of the DFT Specification to at least 8. (SSN_R11-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Error: Unable to satisfy the requested number of capture clock pulses
//  coming from SSH instance 'ssh'.
//  The register that controls the number of capture clock pulses is too
//  small to generate 10 capture clock pulses.
//  You can decrease the number of capture clock pulses via the core
//  instance parameter 'capture_packets' to a maximum of 7,
//  or you can regenerate the SSH hardware to support this number of
//  capture clock pulses setting the 'max_capture_clock_pulses' property
//  of the DFT Specification to at least 3. (SSN_R11-1)

//  Error: Unable to satisfy the requested scan_en setup and hold time
//  for SSH instance 'ssh'.
//  The register that controls the number of scan_en setup and hold cycles
//  is too small.
//  The number of scan_en setup extra cycles is 88 and the number of
//  scan_en hold extra cycles is 1.
//  You can decrease the number of scan_en setup and/or hold extra cycles
//  via: 'set_load_unload_timing_options -scan_en_setup_extra_cycles' and
//  'set_load_unload_timing_options -scan_en_hold_extra_cycles',
//  or you can regenerate the SSH hardware to support this number of
//  scan_en setup and hold cycles setting the 'max_scan_en_mcp' property
//  of the DFT Specification to at least 8. (SSN_R11-1)

//  Error: Unable to satisfy the requested edt_update setup and hold time
//  for SSH instance 'ssh'.
//  The register that controls the number of edt_update setup and hold
//  cycles is too small.
//  The number of edt_update setup extra cycles is 77 and the number of
//  edt_update hold extra cycles is 1.
//  You can decrease the number of edt_update setup and/or hold extra
//  cycles via:
//  'set_load_unload_timing_options -edt_update_setup_extra_cycles' and
//  'set_load_unload_timing_options -edt_update_hold_extra_cycles',
//  or you can regenerate the SSH hardware to support this number of
//  edt_update setup and hold cycles setting the 'max_edt_update_mcp'
//  property of the DFT Specification to at least 8. (SSN_R11-1)

//  Error: Unable to satisfy the requested scan_en setup and hold time for
//  SSH instance 'ssh'.
//  The register that controls the number of scan_en setup and hold
//  cycles is too small.
//  The number of scan_en setup extra cycles is 10000 and the number
//  of scan_en hold extra cycles is 10000.
//  You can decrease the number of scan_en setup and/or hold extra
//  cycles via:
//  'set_load_unload_timing_options -scan_en_setup_extra_cycles' and
//  'set_load_unload_timing_options -scan_en_hold_extra_cycles'.
//  Supporting the current scan_en setup and hold cycles would require
//  specifying the 'max_scan_en_mcp' property of the DFT Specification to
//  at least 16, but the largest allowed is 8. (SSN_R11-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Error: Unable to satisfy the requested minimum capture window size in
// bus clock cycles of 500 for SSH instance 'ssh'.
// The following equation is used to calculate the minimum capture window
// size in bus clock cycles:
// (<scan_en_transition_packets> + (<capture_packets> *
// (<packets_per_capture_clock_minus_one> + 1))) *
// <bus clock cycles per packet>
// The bus clock cycles per packet for this SSH is 7.
// The maximum capture window size in bus clock cycles for this SSH is
// 497.
// You can decrease the number of minimum capture window size in bus clock
// cycles using the 'set_capture_timing_options
// -min_capture_window_in_ssn_bus_cycles' command,
// or you can regenerate the SSH hardware with larger 'max_scan_en_mcp',
// 'max_capture_clock_pulses' and/or
// 'max_capture_to_shift_clock_period_ratio' property values. (SSN_R11-1)
```

SSN_R12

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Checks that the edt_clock and edt_update connections of the EDT connect to the correct SSH instance.

The following lists the checks and the violations that may occur:

- If the channels of an EDT controller connect to an SSH instance, the edt_clock of the EDT controller should connect to the same SSH instance.
  - If the edt_clock of the EDT instance is connected to another SSH instance, the error message is:

    ```
    //  Error: The EDT instance clock 'edt_inst_1/edt_clock'
    //         is not driven by the same SSH instance that drives its
    //         channels.
    //         It is expected to be driven by
    //         'ssh_inst_1/edt_clock', but it is instead driven by
    //         'ssh_inst_2/edt_clock' which is the output of
    //         another SSH instance. (SSN_R12-1)
    ```

    Where ‘edt_inst_1’ is the EDT instance name, and ‘ssh_inst_1’ and ‘ssh_inst_2’ are SSH instance names.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- If the EDT and ScanHost instances are ungrouped, the error message is:

```
//  Error: The EDT instance clock
//         '{\edt_inst_1/edt_clock_buf /Y}' is not driven by
//         the same SSH instance that drives its channels.
//         It is expected to be driven by
//         '{\ssh_inst_1/edt_clock_buf /Y}', but it is instead
//         driven by '{\ssh_inst_2/edt_clock_buf /Y}' which
//         is the output of another SSH instance. (SSN_R12-1)
```

- If the edt_clock of the EDT instance is connected to a top-level port, the error message is:

```
//  Error: The EDT instance clock 'edt_inst_1/edt_clock'
//         is not driven by the same SSH instance that drives
//         its channels.
//         It is expected to be driven by
//         'ssh_inst_1/edt_clock', but it is instead
//         driven by port 'TOP_EDT_CLOCK'. (SSN_R12-1)
```

- If the edt_clock of the EDT instance is undriven, the error message is:

```
//  Error: The EDT instance clock 'edt_inst_1/edt_clock'
//         is not driven by the same SSH instance that drives
//         its channels.
//         It is expected to be driven by 'ssh_inst_1/edt_clock',
//         but it is floating. (SSN_R12-1)
```

- If the channels of an EDT controller connect to an SSH instance, the edt_update of the EDT controller should connect to the same SSH instance.
  - If the edt_update of the EDT instance is connected to another SSH instance, the error message is:

```
//  Error: The EDT instance update 'edt_inst_1/edt_update'
//         is not driven by the same SSH instance that drives
//         its channels.
//         It is expected to be driven by
//         'ssh_inst_1/edt_update', but it is instead driven by
//         'ssh_inst_2/edt_update' which is the output of
//         another SSH instance. (SSN_R12-1)
```

Where ‘edt_inst_1’ is the EDT instance name, and ‘ssh_inst_1’ and ‘ssh_inst_2’ are SSH instance names.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- If the EDT controller and ScanHost node instances are ungrouped, the error message is:

```
//  Error: The EDT instance update
//         '{\edt_inst_1/edt_update_buf /Y}' is not driven by
//         the same SSH instance that drives its channels.
//         It is expected to be driven by
//         '{\ssh_inst_1/edt_update_buf /Y}', but it is instead
//         driven by '{\ssh_inst_2/edt_update_buf /Y}' which is
//         the output of another SSH instance. (SSN_R12-1)
```

- If the edt_update of the EDT controller is connected to a top-level port, the error message is:

```
//  Error: The EDT instance update 'edt_inst_1/edt_update'
//         is not driven by the same SSH instance that drives
//         its channels.
//         It is expected to be driven by
//         'ssh_inst_1/edt_update', but it is instead
//         driven by port 'TOP_EDT_UPDATE'. (SSN_R12-1)
```

SSN_R13

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Default Handling: Error (can be changed to Warning or Note)

report_drc_rules: Supported

Checks that the edt_clock and edt_update connections of the EDT connect to the correct SSH instance.

The following lists the checks and the violations that may occur:

- If the channels of an EDT controller connect to an SSH instance, the edt_clock of the EDT controller should connect to the same SSH instance.
  - If the edt_clock of the EDT instance is connected to a pseudo port, the error message is:

```
//  Error: The EDT instance clock 'edt_inst_1/edt_clock'
//         is not driven by the same SSH instance that drives
//         its channels.
//         It is expected to be driven by
//         'ssh_inst_1/edt_clock', but it is instead driven
//         by pseudo_port 'occ_inst_1_clock_buf_Y'.
//         (SSN_R13-1)
```

    Where ‘edt_inst_1’ is the EDT instance name, and ‘ssh_inst_1’ is the SSH instance name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- If the channels of an EDT controller connect to an SSH instance, the edt_update of the EDT controller should connect to the same SSH instance.
  - If the edt_update of the EDT controller is connected to a pseudo port, the error message is:

```
//  Error: The EDT instance update 'edt_inst_1/edt_update'
//         is not driven by the same SSH instance that drives
//         its channels.
//         It is expected to be driven by
//         'ssh_inst_1/edt_update', but it is instead driven
//         by pseudo_port 'ssh_inst_1_cell_update_out_mux_Y'.
//         (SSN_R13-1)
```

    Where ‘edt_inst_1’ is the EDT instance name, and ‘ssh_inst_1’ is the SSH instance name.

SSN_R15

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Verifies the number of output scan channels does not exceed the output_chain_count_in_on_chip_compare_mode value of a Scanhost when the on-chip compare is enabled.

The SSN_R15 DRC verifies that the number of output scan channels does not exceed the output_chain_count_in_on_chip_compare_mode property value of a ScanHost wrapper when the on-chip compare is enabled.

Note

Unlike most DRCs, the tool checks this rule when you issue the write_patterns command and not during system mode transition.

Examples

Example 1

```
//  Error: The SSH instance
//  'core_g1_xtea_piccpu_gate2_tessent_ssn_scan_host_piccpu_inst' has
//  on-chip compare enabled but is observing too many scan channels (10).
//  The maximum number of scan channels that can be observed is 5.
//  Regenerate the SSH, specifying the
//  output_chain_count_in_on_chip_compare_mode property of the SSH in the
//  DFT Spec to at least (10). (SSN_R15-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

This example illustrates the error produced by an EDT dual configuration:

```
//  Error: The SSH instance
//  'core_g1_xtea_piccpu_gate2_tessent_ssn_scan_host_xtea_inst' has
//  on-chip compare enabled but is observing too many scan channels (20).
//  The maximum number of scan channels that can be observed is 3.
//  Regenerate the SSH, specifying the
//  output_chain_count_in_on_chip_compare_mode property of the SSH in the
//  DFT Spec to at least (20).//  Alternatively, use the high compression
configuration of EDT by
//  specifying the '-parameter_values' option:
//  set_core_instance_parameters -instances
//  {xtea_1/xtea_rtl_tessent_edt_c10_inst} -parameter_values
//  {edt_configuration high_compression} (SSN_R15-1)
```

SSN_R16

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Verifies that pipeline stages for EDT and for uncompressed chains driving or driven by an SSH are clocked during shift by the shift_capture_clock, edt_clock, or test_clock clocks of this SSH.

To list all affected channel pipelines for every violation of the rule, issue this command:

report_drc_rules SSN_R16 -verbose

To list all affected channel pipelines for violation *n* of the rule, issue this command:

report_drc_rules SSN_R16-*n* -verbose

where *n* is a single violation instance, such as 1, 2, and so on.

When the pipeline is driven by an SSH that has two clocks, the error message refers only to the shift_capture_clock. If driven by the edt_clock, there is no error.

Examples

Example 1

This example shows a pipeline that is driven by another SSH instance.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  Error: Multiple channel pipelines are expected to be clocked during
//  shift by 'top_ssn_scan_host_1_inst/shift_capture_clock',
//  but they are instead clocked by
//   'top_ssn_scan_host_2_inst/shift_capture_clock'
//  which is the output of another SSH instance. (SSN_R16-1)
//  One of those pipelines is:
//   'coreC_i1/coreC_edt_c1_ext_chan_in_1_lockup_inst_last'.
//  To list all affected channel pipelines, issue 'report_drc_rules'
//  with the '-verbose' switch.


report_drc_rules SSN_R16 -verbose
//  Error: Multiple channel pipelines are expected to be clocked during
//  shift by 'top_ssn_scan_host_1_inst/shift_capture_clock',
//  but they are instead clocked by
//   'top_ssn_scan_host_2_inst/shift_capture_clock'
//  which is the output of another SSH instance. (SSN_R16-1)
//  Those pipelines are:
//   'coreC_i1/coreC_edt_c1_ext_chan_in_1_lockup_inst_last',
//   'coreC_i1/coreC_edt_c1_ext_chan_in_0_pipe_inst_2',
//   'coreC_i1/coreC_edt_c1_ext_chan_in_0_lockup_inst_last'.
```

Example2

This examples shows a pipeline that is driven by a top-level port.

```
//  Error: Multiple channel pipelines are expected to be clocked during
//  shift by 'coreb_rtl2_tessent_ssn_scan_host_a_inst/test_clock',
//  but they are instead clocked by port 'YZ'. (SSN_R16-1)
//  One of those pipelines is:
//  corea/corea_rtl1_tessent_edt_c1_ext_chan_in_0_pipe_inst_0'.
//  To list all affected channel pipelines, issue 'report_drc_rules'
//  with the '-verbose' switch.


//  command: report_drc_rules ssn_r16 -verbose
//  Error: Multiple channel pipelines are expected to be clocked during
//  shift by 'coreb_rtl2_tessent_ssn_scan_host_a_inst/test_clock',
//  but they are instead clocked by port 'YZ'. (SSN_R16-1)
//  Those pipelines are:
//  'corea/corea_rtl1_tessent_edt_c1_ext_chan_in_0_pipe_inst_0',
//  'corea/corea_rtl1_tessent_edt_c1_ext_chan_in_0_lockup_inst_last',
//  'corea/corea_rtl1_tessent_edt_c1_ext_chan_out_0_pipe_inst_0',
//  'corea/corea_rtl1_tessent_edt_c1_ext_chan_out_0_lockup_inst_last'.
```

SSN_R17

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting

Default Handling: Error (can be changed with the set_drc_handling command to Warning or Note)

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Verifies that pipeline stages for EDT and for uncompressed chains driving or driven by an SSH are not clocked during shift by pseudo_ports that are not SSH outputs.

To list all affected channel pipelines for every violation of the rule, issue this command:

report_drc_rules SSN_R17 -verbose

To list all affected channel pipelines for violation *n* of the rule, issue this command:

report_drc_rules SSN_R17-*n* -verbose

where *n* is a single violation instance, such as 1, 2, and so on.

When the pipeline is driven by an SSH that has two clocks, the error message refers only to the shift_capture_clock. If driven by the edt_clock, there is no error.

Examples

This example shows a pipeline that is driven by a top pseudo port.

```
//  Error: Multiple channel pipelines are expected to be driven by
//  'coreb_rtl2_tessent_ssn_scan_host_a_inst/test_clock',
//  but they are instead driven by pseudo_port 'YZ_PPORT'. (SSN_R17-1)
//  One of those pipelines is:
//  'corea/corea_rtl1_tessent_edt_c1_ext_chan_in_0_pipe_inst_0'.
//  To list all affected channel pipelines, issue
//  'report_drc_rules' with the '-verbose' switch.


//  command: report_drc_rules ssn_r17 -verbose
//  Error: Multiple channel pipelines are expected to be driven by
//  'coreb_rtl2_tessent_ssn_scan_host_a_inst/test_clock',
//  but they are instead driven by pseudo_port 'YZ_PPORT'. (SSN_R17-1)
//  Those pipelines are:
//   'corea/corea_rtl1_tessent_edt_c1_ext_chan_in_0_pipe_inst_0',
//   'corea/corea_rtl1_tessent_edt_c1_ext_chan_in_0_lockup_inst_last',
//   'corea/corea_rtl1_tessent_edt_c1_ext_chan_out_0_pipe_inst_0',
//   'corea/corea_rtl1_tessent_edt_c1_ext_chan_out_0_lockup_inst_last'.
```

SSN_R18

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Verifies that scan cells for EDT and uncompressed chains driving or driven by an SSH are clocked by the same SSH during shift.

To list all affected scan cells for every violation of the rule, run this command:

report_drc_rules SSN_R18 -verbose

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

To list all affected scan cells for violation *n* of the rule, run this command:

report_drc_rules SSN_R18-*n* -verbose

where *n* is a single violation, such as 1, 2, and so on.

Examples

The following example shows a scan cell driven by another SSH instance:

```
//  Error: Multiple scan cells are expected to be clocked during shift by
//  'coreb_rtl2_tessent_ssn_scan_host_a_inst/test_clock',
//         but they are instead clocked by
//  'coreb_rtl2_tessent_ssn_scan_host_b_inst/shift_capture_clock' which
//  is the output of another SSH instance. (SSN_R18-1)
//         One of those scan cells is:
//             '{corea/\ffb1_reg[9] }'.
//         To list all affected scan cells, issue 'report_drc_rules' with
//         the '-verbose' switch.
```

report_drc_rules ssn_r18-1 -verbose

```
//  Error: Multiple scan cells are expected to be clocked during shift by
//  'coreb_rtl2_tessent_ssn_scan_host_a_inst/test_clock',
//         but they are instead clocked by
//  'coreb_rtl2_tessent_ssn_scan_host_b_inst/shift_capture_clock' which
//  is the output of another SSH instance. (SSN_R18-1)
//         Those scan cells are:
//             '{corea/\ffb1_reg[9] }',
//             '{corea/\ffb1_reg[10] }',
//             '{corea/\ffb1_reg[11] }',
//             '{corea/\ffb1_reg[12] }',
//             '{corea/\ffb1_reg[13] }',
//             '{corea/\ffb1_reg[14] }',
//             '{corea/\ffb1_reg[15] }',
//             '{corea/\ffb1_reg[16] }',
//             'corea/ts_0_diw_501sffp1_i',
//             '{corea/\ffa1_reg[1] }',
//             '{corea/\ffa1_reg[2] }',
//             '{corea/\ffb4_reg[21] }',
//             '{corea/\ffb4_reg[20] }',
//             '{corea/\ffb4_reg[19] }',
//             '{corea/\ffb4_reg[18] }',
//             '{corea/\ffb1_reg[29] }',
//             '{corea/\ffb1_reg[30] }',
//             '{corea/\ffb1_reg[31] }',
//             'corea/ts_0_pipeline_fsffp_clkc29_lo0l5_extsi522_sea0_i'.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following example shows a scan cell driven by a top-level port:

```
//  Error: Multiple scan cells are expected to be clocked during shift by
//  'coreb_rtl2_tessent_ssn_scan_host_a_inst/test_clock',
//         but they are instead clocked by port 'YZ'. (SSN_R18-1)
//         One of those scan cells is:
//             '{corea/\ffb1_reg[9] }'.
//         To list all affected scan cells, issue 'report_drc_rules' with
//         the '-verbose' switch.
```

command: report_drc_rules ssn_r18-1 -verbose

```
//  Error: Multiple scan cells are expected to be clocked during shift by
//  'coreb_rtl2_tessent_ssn_scan_host_a_inst/test_clock',
//         but they are instead clocked by port 'YZ'. (SSN_R18-1)
//         Those scan cells are:
//             '{corea/\ffb1_reg[9] }',
//             '{corea/\ffb1_reg[10] }',
//             '{corea/\ffb1_reg[11] }',
//             '{corea/\ffb1_reg[12] }',
//             '{corea/\ffb1_reg[13] }',
//             '{corea/\ffb1_reg[14] }',
//             '{corea/\ffb1_reg[15] }',
//             '{corea/\ffb1_reg[16] }',
//             'corea/ts_0_diw_501sffp1_i',
//             '{corea/\ffa1_reg[1] }',
//             '{corea/\ffa1_reg[2] }',
//             '{corea/\ffb4_reg[21] }',
//             '{corea/\ffb4_reg[20] }',
//             '{corea/\ffb4_reg[19] }',
//             '{corea/\ffb4_reg[18] }',
//             '{corea/\ffb1_reg[29] }',
//             '{corea/\ffb1_reg[30] }',
//             '{corea/\ffb1_reg[31] }',
//             'corea/ts_0_pipeline_fsffp_clkc29_lo0l5_extsi522_sea0_i'.
```

SSN_R19

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (can be changed with the set_drc_handling command to Warning or Note)

report_drc_rules: Supported

Verifies that scan cells for EDT and uncompressed chains driving or driven by an SSH are not clocked during shift by pseudo_ports that are not SSH outputs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example shows a scan cell driven by a top-level pseudo_port:

```
//  Error: Multiple scan cells are expected to be clocked during shift by
//  'coreb_rtl2_tessent_ssn_scan_host_a_inst/test_clock',
//    but they are instead clocked by pseudo_port 'YZ_PPORT'. (SSN_R19-1)
//         One of those scan cells is:
//             '{corea/\ffb1_reg[9] }'.
//         To list all affected scan cells, issue 'report_drc_rules' with
//         the '-verbose' switch.
```

report_drc_rules ssn_r19-1 -verbose

```
//  Error: Multiple scan cells are expected to be clocked during shift by
//  'coreb_rtl2_tessent_ssn_scan_host_a_inst/test_clock',
//         but they are instead clocked by pseudo_port 'YZ_PPORT'.
//         (SSN_R19-1)
//         Those scan cells are:
//             '{corea/\ffb1_reg[9] }',
//             '{corea/\ffb1_reg[10] }',
//             '{corea/\ffb1_reg[11] }',
//             '{corea/\ffb1_reg[12] }',
//             '{corea/\ffb1_reg[13] }',
//             '{corea/\ffb1_reg[14] }',
//             '{corea/\ffb1_reg[15] }',
//             '{corea/\ffb1_reg[16] }',
//             'corea/ts_0_diw_501sffp1_i',
//             '{corea/\ffa1_reg[1] }',
//             '{corea/\ffa1_reg[2] }',
//             '{corea/\ffb4_reg[21] }',
//             '{corea/\ffb4_reg[20] }',
//             '{corea/\ffb4_reg[19] }',
//             '{corea/\ffb4_reg[18] }',
//             '{corea/\ffb1_reg[29] }',
//             '{corea/\ffb1_reg[30] }',
//             '{corea/\ffb1_reg[31] }',
//             'corea/ts_0_pipeline_fsffp_clkc29_lo0l5_extsi522_sea0_i'.
```

SSN_R20

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Verifies during internal capture that all PIs are constrained, or defined as pulse_always or asynchronous clocks.

When mode type is internal, this rule checks that the top-level PIs (this does not apply to pseudo_ports) are one of the following:

- capture constrained

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- pulse_always clocks
- asynchronous clocks

Example

```
//  Error: 'clk' is not a constrained port, pulse_always clock, or
//  asynchronous clock.
//  The current design input ports must have predictable values and cannot
//  be left for ATPG to specify in internal mode. (SSN_R20-1)
```

SSN_R21

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Verifies for internal mode that top-level ports (does not apply to pseudo_ports) load_unload and capture constraints do not have conflicting values.

```
//  Error: 'clk' is constrained to 1 during load_unload and to 0 during
//  capture.
//  The current design input ports should not be constrained to
//  conflicting binary values in internal mode. (SSN_R21-1)
```

SSN_R23

Category: SSN

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Verifies that SSN bus clocks are not defined as pulse-always.

The following is the error message:

```
// Error: The SSN bus clock '<clk>' is specified as pulse-always.
// (SSN_R23-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Scan Chain Trace Rules (T Rules)

Using the information in the test procedure files, the rule checker traces the scan chains to identify the scan cells and all memory elements associated with the scan cells. It then classifies the scannable memory elements as either MASTER, SLAVE, SHADOW, COPY, or EXTRA.

Violations of scan chain trace rules are error or warning conditions, and you cannot change the handling of these rules, except for T18. You can set rule T18 to ignore.

In the DFT context, you can also define scan chain segments which are stitched into larger chains during scan insertion. Some of these scan segments are targeted during scan chain tracing even when a user-provided load/unload procedure does not exist.

Scan segments are typically defined with respect to input and output pins for a specific instance within the design. The tool injects values at these control locations to facilitate scan chain tracing. When tracing fails, the simulation values can be viewed directly in the visualizer using the analyze_drc_violation command. However, when tracing passes, these simulation values are not preserved. You can use the scan_segment_tracing_clock_on and scan_segment_tracing_clock_off simulation contexts to access these values for debugging purposes. For more information, see "set_current_simulation_context" on page 2434.

The following subsections describe each of the trace rules.

Tip

In Tessent Visualizer, you can press Ctrl + R to execute a report_gates command on selected gates.

T1 3588
T2 3588
T3 3589
T4 3593
T5 3597
T6 3599
T8 3600
T9 3600
T10 3601
T11 3601
T12 3601
T13 3602
T14 3602
T15 3602
T16 3603

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

T17 3603
T18 3604
T19 3604
T20 3604
T21 3605
T22 3605
T23 3606
T24 3606
T25 3607
T26 3608
T27 3609

T1

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

All defined scan chains must contain at least one scan cell. Correct this error condition by deleting the indicated scan chains.

The error message is:

```
No scan cells identified in scan chain C. (T1-1)
```

C is the scan chain name.

T2

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

A scannable memory element can reside in more than one scan chain unless you have run the "set_drc_handling -ALLOW_COMmon_scan_cells_in_multiple_chain OFf" command. However, there are some restrictions on scan cell sharing, which result in a T2 violation when not observed:

- Scan cell sharing is not allowed for compressed chains driven by a decompressor.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The shared scan cells must be at the logical scan cell boundary. A library model with multiple scan cells cannot be partially shared. In other words, you cannot have the merged point within the state elements of a logical scan cell.
- The shared scan cells must not end at a master segment of a master/slave combination.

You can display the complete paths of the scan chains by using the set_trace_report command by turning trace reporting on and then repeating the rules checking. If you use all scan chains, you may need to make netlist modifications to correct this error condition.

The T2 DRC flags a violation when the split point drives scan cells with different clock edges. The split point and the driven point need to be at the same clock edge, for example leading edge to leading edge, trailing edge to leading edge, or trailing edge to trailing; never leading edge to trailing edge.

The T2 rule has three possible error messages, in which N is an instance name, and G is its gate ID number:

- You have run the "set_drc_handling -ALLOW_COMmon_scan_cells_in_multiple_chain OFf" command, and the scan cell exists in more than one scan chain:

  ```
  N (G) already used in chain trace. (T2-1)
  ```

- Two scan chains share a merge point that is within a single library model:

  ```
  Chain N and N split at N (G) which is not a logical cell boundary.
  (T2-1)
  ```

- The scan cell exists in more than one scan chain, and at least one of the chains is compressed:

  ```
  Scan cell N (G) is shared by compressed chains N and N. (T2-1)
  ```

T3

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

The shift procedure must create a sensitizable path from the scan chain output back to the scan chain input. An improperly sensitized gate in the scan path causes an error condition, which the tool reports as a T3 DRC violation.

The occurrence message is

```
Scan chain S blocked at gate N (G) after tracing C cells. (T3-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

S is the scan chain name, N is the instance name, G is its gate ID number, and C is the number of scan cells traced in the scan chain.

The summary message is:

```
T3:  #fails=N handling=error (scan path blocked)
```

N is the number of T3 rule violations.

How to Debug T3 Violations in Tessent Visualizer

You can debug T3 violations in Tessent Visualizer. When the tool encounters the error, it reports a message similar to the following:

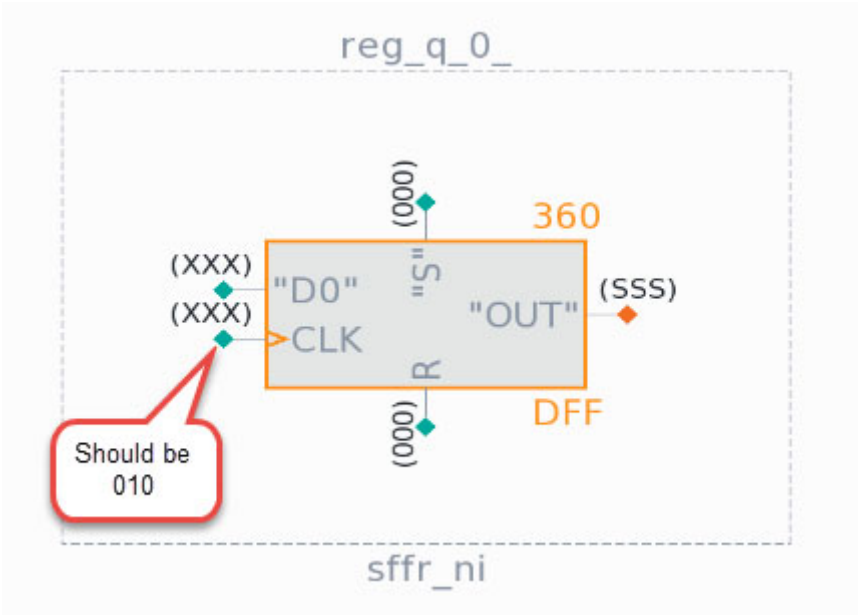
```
//  Error: Scan chain chain1 blocked at gate /datao/reg_q_0_ (360) after
tracing 0 cells.  (T3-1)
//  Error: Rules checking unsuccessful, cannot exit SETUP mode.
```

You can analyze the DRC error in Tessent Visualizer using the DRC Browser or by issuing this command:

analyze_drc_violation T3-1

Tessent Visualizer starts and displays /datao/reg_q_0_ in the Flat Schematic window as shown in Figure 10-55.

Figure 10-55. T3 Violation



The clock “CLK” should be 010, not XXX.

Trace backwards by clicking the pin for “CLK”. Open the Tracer table and, using the Columns and Filters Editor, add the “Gate data (end)” column. In the “Filter” field of the new column,

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

add "='(XXX)'" (without the double quotation marks). Continue tracing the path with the XXX value to find the source of the problem.

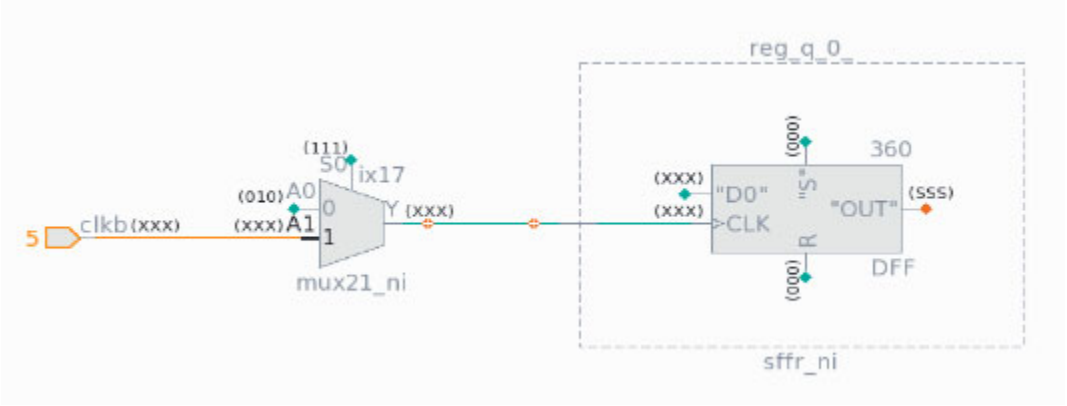
Note

When a T3 violation occurs, and you use the analyze_drc_violation command, the gate report is set to trace and the report shows an SSS on some paths. The S distinguishes scan path "don't cares" from non-scan path "don't cares", which are shown as X.

The same happens when you use the "set_gate_report drc shift" command.

The result of tracing backwards from "CLK" is shown in Figure 10-56.

Figure 10-56. Tracing T3 Violation



The "CLK" pin is driven by "clkb", which is not pulsed.

The solution is to modify the test procedure to pulse "clkb" during shift. Then issue the following commands:

```
add_clocks clkb
set_system_mode analysis
```

How to Debug T3 Violations Without Tessent Visualizer

Correct this error condition by accessing the simulated values of all time periods of the **shift** procedure. To do this, set the gate reporting to trace, and use the report_gates command for the gate ID number displayed in the error message. This can help you to identify where the blockage occurs; tracing back from the inputs helps you identify how to correct the problem.

The error message is:

```
//  Error: Scan chain chain1 blocked at gate /reg2/df2 (39) after tracing
1 cells.  (T3-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The message tells you that the scan chain, “chain1,” is blocked at the gate /datao/reg_q_0. The tool stays in SETUP mode. To proceed to debug, set up the tool to be in trace mode. If you have not already done so, issue the following commands:

set_trace_report on
set_gate_level primitive
set_gate_report trace

Re-run the DRC rule checks by going back into ANALYSIS mode and then report the gate that is identified as having a T3 error.

set_system_mode analysis
report_gate /reg2/df2

```
//  /reg2/df2 (39)  DFF
//      "S"    I  (000)  10-
//      "R"    I  (000)  9-
//      "C0"   I  (XXX)  26-
//      "D0"   I  (XXX)  23-
//      "OUT"  O  (SSS)  13-  14-
```

When you run report_gates with trace on, you expect the scan clock to show “010”, the constrained pins to be at their constrained values, scan enable to be active, and all sets and resets to be inactive. Scan out and scan in (on scan elements) should all be “X”. On all the combinational elements, in the scan chain, you should see “X” on the output and “X” on one of the inputs. When you have “X” on two inputs, the chain is blocked.

In this example, the blocked gate is a flop. Pins “S” and “R” are both inactive, which is okay, but the clock “C0” and data “D0” pins have a problem. “C0” is not shifting, and since this is a flop, it should show (010). Do a backwards trace to find out what drives the clock. In this example, we set the gate level to be design.

set_gate_level design
report_gate /reg2/df2

```
//  /reg2  dfsc
//      ck     I  (XXX)  /uu1/out
//      d0     I  (XXX)  /gate2/out
//      d1     I  (XXX)  /reg3/qbar
//      sc     I  (111)  /scan_en
//      q      O  (XXX)  /gate3/a
//      qbar   O  (SSS)  /reg1/d1
```

The clock, “ck”, is driven by /uu1/out, so report that gate.

report_gates /uu1

```
//  /uu1  mux21
//      d1     I  (XXX)  /reg1/q
//      d2     I  (010)  /clock
//      sel    I  (XXX)  /ten_reg/q
//      out    O  (XXX)  /reg2/ck  /reg3/ck
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Gate /uu1 is a mux with an unknown driver. One of the pins, “d2”, looks like the clock, so the “sel” signal should have been “111”, but it is “XXX”. Trace /ten_reg.

report_gates /ten_reg

```
//  /ten_reg  dff
//      ck     I  (XXX)  /in3
//      d      I  (111)  /gate5/out
//      q      O  (XXX)  /uu1/sel
//      qbar   O  (XXX)
```

The gate /ten_reg is a dflop that controls the mux. In this design, this control signal is actually an internally generated “test_enable” signal. The flop needs to be initialized to hold a “111” on the output. Modify test_setup sequence as needed to put the device into a state so there is a sensitizable path from the scan chain output back to the scan chain input.

T4

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

A memory element in the scan path must have an active clock during some time period of the shift procedure. Correct this error condition by accessing the simulated values of all time periods of the shift procedure. You do this by setting the gate reporting to trace and using the report_gates command for the gate ID number displayed in the error message. This can help you identify where the problem occurred; tracing back from the inputs helps you identify how to correct the problem.

The error message is:

```
Clock inputs of N (G) never set active during shift procedure. (T4-1)
```

N is the instance name and G is its gate ID number.

The summary message is:

```
T4:  #fails=F handling=error (scan cell without active clock event during
shift)
```

F is the number of T4 violations.

How to Debug T4 Violations in Tessent Visualizer

You can debug T4 violations in Tessent Visualizer. When the tool encounters the error, it reports a message, for example

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

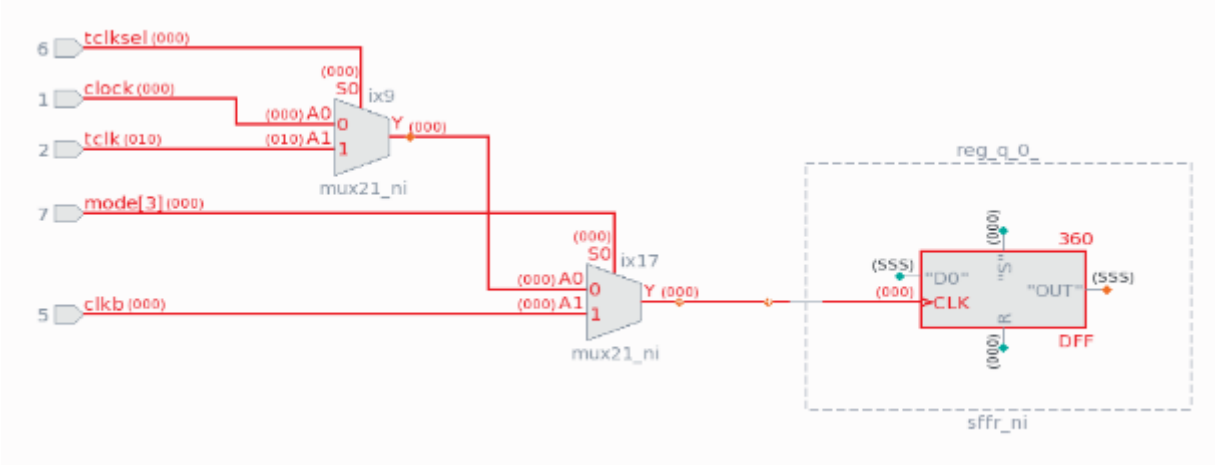
```
//  Error: Clock inputs of /datao/reg_q_0_ (360) never set active during
//  shift procedure.  (T4-1)
```

Because T4 and T3 violations occur in pairs, you can also analyze T3-1. Use the DRC Browser or issue this command:

analyze_drc_violation T4-1

Tessent Visualizer starts and displays the instance with the violation. Figure 10-57 shows the signal driving CLK of /reg_q_0_ violates the T4 trace rule and the traceback to the source of the problem. CLK must be active at some point.

Figure 10-57. T4 Violation



Note

When a T4 violation occurs, and you use the analyze_drc_violation command, the gate report is set to trace and the report shows an SSS on some paths. The S distinguishes scan path “don’t cares” from non-scan path “don’t cares,” which are shown as X.

The same happens when you have used the “set_gate_report drc shift” command.

The traceback shows that the control signal for MUX /ix17 is set to select input A0. A0 is driven by MUX /ix9, whose control signal, tclksel, is selecting A0 as well. However, the input A1 has the pulse that is required to fix the problem CLK. So you need to have tclksel go high.

You need to find when to pulse the offending clock high by forcing tclksel high at some point. You can determine when by setting the gate report to see the stable values of all the signals. This can be done with the command:

set_gate_report drc state_stability

Figure 10-58 shows the offending /reg_q_0_ instance and the states of the signals. CLK is fine until the last shift, when it is 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-58. T4 Violation Showing States

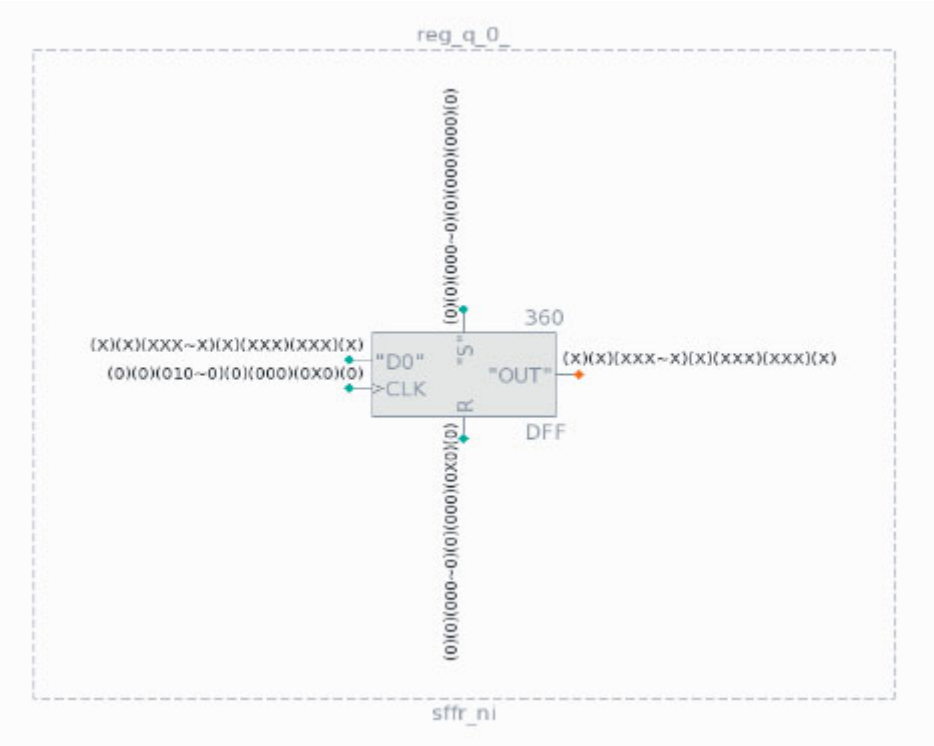
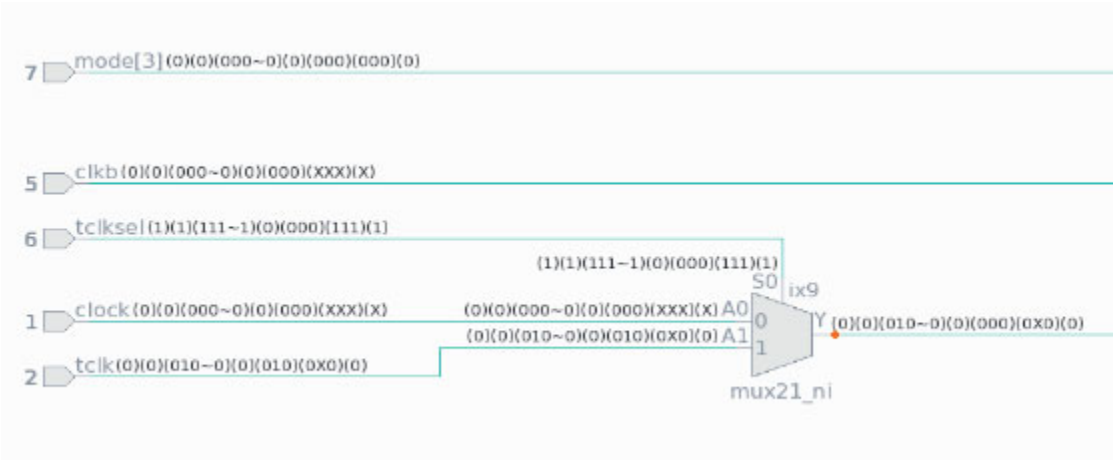


Figure 10-59 shows the result of tracing back to tclksel. The tclksel signal is 000 in the last shift and needs to be forced to 1 in the load_unload procedure.

Figure 10-59. T4 Violation Traced Back Showing States



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The load_unload procedure in the test procedure file should look like this:

```
procedure load_unload =
    scan_group grp1 ;
    timeplate gen_tp2 ;
    cycle =
        force tclksel 1 ;
        force clear 0 ;
        force scan_en 1 ;
        force tclk 0 ;
        force clka 0 ;
        force clkb 0 ;
        force clkr 0 ;
        force clock 0 ;
    end ;
    apply shift 12;
    cycle =
        force tclksel 1 ;
    end ;
    apply shift 1;
 end;
```

How to Debug T4 Violations Without Tessent Visualizer

You correct T4 violations by making certain that there is an active clock during some time period of the shift procedure. Correct this error condition by accessing the simulated values of all the time periods of the shift procedure.

For simple cases set the gate reporting style to trace:

set_gate_report trace

Use the report_gates command to help you know where to trace back on the blocking path and then sensitize the path correctly.

For more complex cases set the gate report style to state_stability:

set_gate_report drc state_stability

Go back into analysis mode:

set_system_mode analysis

Use the report_gates command to help you know where to trace back on the blocking path.

Make the necessary changes to sensitize the path correctly.

There could be several causes for T4 violations, these are the most common issues

- The shift clock is defined incorrectly in the shift procedure.
- A missing or incorrect pin constraint.
- The TAP controller is in the wrong state.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

T5

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

During the shift procedure, you must never place an X value on a clock input or an active (X or 1) value on a set or reset input of a memory element in the scan path. The T5 DRC reports such issues.

The error message is:

```
T input of N (G) set to V. (T5-1)
```

T is the type of input (clock, set, or reset), N is the instance name, G is its gate ID number, and V is the invalid state.

The summary message is:

```
T5:  #fails=F handling=error (scan cell with unexpected clock event)
```

F is the number of fails for this DRC violation.

How to Debug T5 Violations in Tessent Visualizer

You can debug T5 violations in Tessent Visualizer. When the tool encounters the error, it reports a message:

```
//  Error: clock input of /datao/reg_q_0_ (360) set to X.  (T5-1)
//  Error: Scan chain chain1 blocked at gate /datao/reg_q_0_ (360)
//  after tracing 0 cells.  (T3-1)
```

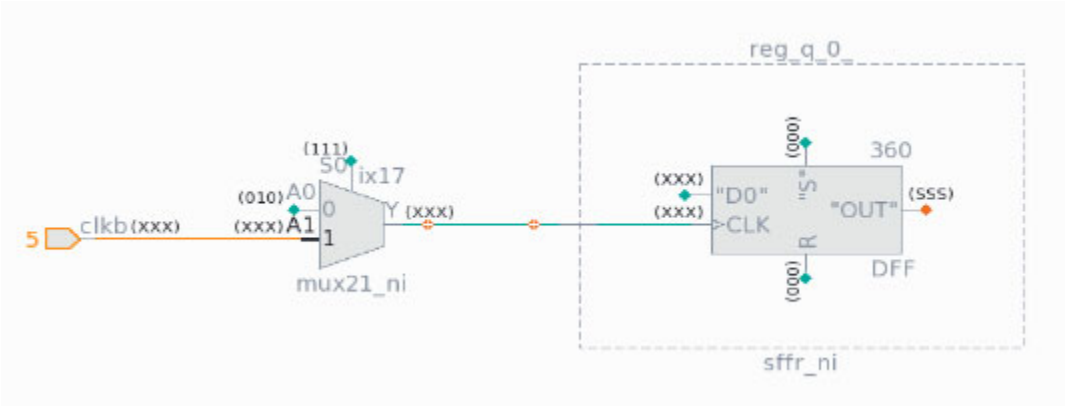
T5 and T3 DRC violations occur in pairs. Resolving one corrects the other. To analyze the T5 DRC violation, use the DRC Browser or this command:

analyze_drc_violation T5-1

Tessent Visualizer opens displaying the instance with the T5 (and T3) violation as shown in Figure 10-60

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-60. T5 DRC Violation



The clock input, CLK, of the /datao/reg_q_0_ is set to XXX. That is the cause of both the T5 and T3 violations. The S0 pin of the MUX appears to be constrained to 1, which selects the A1 input of the MUX. The desired input is A0, which has a pulse 010.

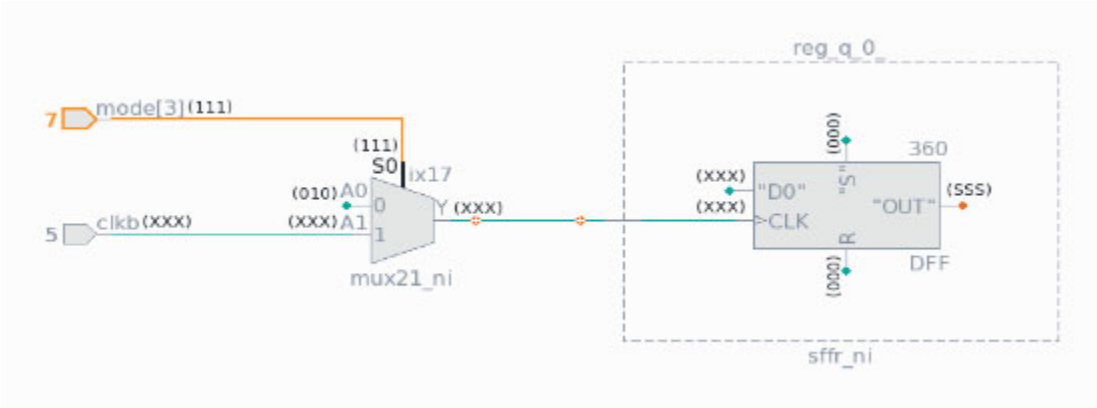
Note

When a T5 violation occurs, and you use the analyze_drc_violation command, the gate report is set to trace and the report shows an SSS on some paths. The S distinguishes scan path “don’t cares” from non-scan path “don’t cares”, which are shown as X.

The same happens when you have used the “set_gate_report drc shift” command.

Figure 10-61, shows S0 is driven by a primary input, mode[3].

Figure 10-61. T5 DRC Violation Traced Back



Use the command report_input_constraints to verify mode[3] is constrained. The solution for this would be to constrain the input S0 to 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This can be done temporarily with these commands:

delete_input_contraint mode[3]
add_input_constraint mode[3] -C0
set_system_mode analysis

You could also make certain the primary input is correctly constrained in a dofile.

How to Debug T5 DRC Violations Without Tessent Visualizer

You correct T5 violations by making certain that during the shift procedure, you must never place an X value on a clock input or an active (X or 1) value on a set or reset input of a memory element in the scan path. Correct this error condition by accessing the simulated values of all the time periods of the shift procedure. You do this by setting the gate reporting to trace and using the report_gates command for the gate ID number displayed in the error message. This can help you identify where the problem occurred; tracing back from the indicated input helps you identify how to correct the problem.

Set the gate reporting style to trace:

set_gate_report trace

Use the report_gates command to trace back on the blocking path to find the path that corrected.

T6

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

During any time period of the shift procedure, a memory element in the scan path must never have more than one clock input turned on.

Correct this error condition by accessing the simulated values of all time periods of the shift procedure. You do this by setting the gate reporting to trace and using the report_gates command for the gate ID number displayed in the error message. This can help you identify where the problem occurred; tracing back from the indicated input helps you identify how to correct the problem.

The error message is:

```
Multiple clock inputs of N (G) set active. (T6-1)
```

N is the instance name and G is the gate ID number.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

T8

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The measure_sco statement in the shift procedure must follow the successful observation of the scan cells.

To guarantee the observation is successful, the time of the measure_sco statement must meet the following conditions:

- It must not occur after exercising the first clock on the memory element closest to the scan chain output.
- It cannot be at time 0 if you capture the data into the last memory element at the end of the shift procedure.
- The states on all inputs at the measure_sco time must be the same as at the end of the shift procedure.

To correct the error condition, you must modify the shift procedure to meet the above conditions for the measure_sco statement.

The error message is:

```
Invalid measure_sco time. (T8-1)
```

T9

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The traced scan chain input pin must be the same as the scan chain input pin specified with the add_scan_chains command. You can correct this error condition by redefining the scan chain input to be the traced pin.

The error message is:

```
Chain input P1 doesn't match entered value P2. (T9-1)
```

P1 is the name of the traced scan chain input pin, and P2 is the name of the entered scan chain input pin.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

T10

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The time of the force_sci statement in the shift procedure must occur before a clock input of the memory element (closest to the scan chain input) turns on. Correct this error condition by changing the time of the force_sci statement to a value less than or equal to the indicated time.

The error message is:

```
Force_sci must occur on or before time T. (T10-1)
```

T is the maximum time.

T11

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

A clock input of the memory element (closest to the scan chain input) must not turn on during the shift procedure prior to the time of the force_sci statement. Correct this error condition by changing the times of the force_sci statement or force statements.

The error message is:

```
Incorrect propagation of force_sci value to scan cell. (T11-1)
```

T12

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

If a scan cell contains a SLAVE element, the MASTER element is not directly observable.

The error message is:

```
MASTER not observable, a master_observe procedure is required by scan
group G. (T12-1)
```

G is the scan group.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When the handling is set to other than error, the tool automatically makes the necessary MASTER unobservable to prevent a potential simulation mismatch. This applies only to the MASTER of the scan cell containing a SLAVE. The scan cell without a SLAVE retains its original observability. Due to the loss of observability on some MASTERs, test coverage may be reduced. To correct this violation, you can define a master_observe procedure to propagate the MASTER value to the SLAVE.

T13

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

If you define and try to use a master_observe procedure, there must be at least one scan cell that contains a SLAVE. If there is no such cell, Correct this error condition by deleting the master_observe procedure.

The error message is:

```
Master_observe procedure defined but not used. (T13-1)
```

T14

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

If you define and try to use a shadow_control procedure, there must be at least one identified SHADOW memory element. Correct this error condition by either changing or deleting the shadow_control procedure.

The error message is:

```
No SHADOWs identified using shadow_control procedure. (T14-1)
```

T15

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If you define and try to use a shadow_observe procedure, the procedure must observe at least one SHADOW. You can correct this error condition by changing or deleting the shadow_observe procedure.

The error message is:

```
No observable SHADOWs identified using shadow_observe  procedure. (T15-1)
```

T16

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

When clocks and write control lines are off and pin constraints are set, the gate that connects to the input of a reconvergent pulse generator sink gate (PGS) in the long path must be at the non-controlling value of the PGS gate.

To correct this error condition, access the simulated values by setting the gate reporting to error_pattern and using the report_gates command. You can also avoid this error by setting the pulse generators to off, but that results in no pulse generator support.

The error message is:

```
Input of pulse generator N (G) not at correct value when clocks are off.
(T16-1)
```

N is the pulse generator instance name, and G is its gate ID number.

T17

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Reconvergent pulse generator sink gates (PGS) cannot connect to any of the following: primary outputs, non-clock inputs of scan memory elements, ROM gates, non-write inputs of RAMs, or transparent latches. To avoid this error, set the pulse generators to off, however note that this results in no pulse generator support.

The error message is:

```
Pulse generator N1 (G1) connected to T N2. (T17-1)
```

N1 is the pulse generator instance name, and G1 is its gate ID number.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

T18

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Not Supported

The maximum number of traced cells in the longest scan chain of a group must equal the entered number of repetitions in the apply shift statement in the load_unload procedure. The default setting for this DRC is "Ignore."

You can enable this check with the set_drc_handling command.

You can correct this condition by changing the repetition number on the apply shift statement. This rules violation has no adverse effects because the tool recalculates the actual number of necessary shifts based on the number of scan cells it encounters.

The warning message is:

```
Traced number of shifts (N1) doesn't match entered value (N2).  (T18-1)
```

N1 is the traced number of shifts, and N2 is the entered number of shifts.

T19

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported

If one scan cell has a SLAVE, then all scan cells must have a SLAVE. You must correct this warning by changing the netlist of the scan chains.

The warning message is:

```
N scan cells do not have a SLAVE when some do. (T19-1)
```

N is the number of non-slave scan cells.

T20

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The number of shifts specified using the set_number_shifts command must be at least equal to the length of the longest scan chain.

To correct this error set a valid value using the set_number_shifts command.

The error message is:

```
Entered number of shifts N is too small. (T20-1)
```

N is the entered number of shifts, and T20 is the rule ID number.

T21

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The number of independent shift applications in the load_unload procedure must be less than the scan chain length. Correct this error by removing a sufficient number of independent shift applications from the load_unload procedure or by deleting the short scan chain.

The error message is:

```
Number of independent shifts N must be less than scan chain length L.
(T21-1)
```

N is the number of independent shift applications and L is the scan chain length.

T22

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

If the rules checker traces a scan cell during the application of an independent shift, it must also trace that cell during the application of its associated general shift. Correct this error by changing the sensitization for either the independent or general shift, so that they are sensitizing the same scan cells.

The error message is:

```
N (G) was not used in general chain trace. (T22-1)
```

N is the scan cell instance name, G is its gate ID number, and T22 is the rule ID number.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

T23

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The chain length calculated for an independent shift must be the same as that calculated for its associated general shift. Correct this error by changing the sensitization for either the independent or general shift, so that they are sensitizing the same scan cells.

The error message is:

```
Chain length (L1) using independent shift not equal to chain  length (L2).
(T23-1)
```

L1 is the independent shift chain length, L2 is the general shift chain length, and T23 is the rule ID number.

T24

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

When two adjacent memory elements (source and sink) in a scan chain path are clocked by different shift clocks, the sink must not capture data from the source at the same time the source changes its value. Failure to satisfy this rule can result in unwanted shoot-through during scan chain shifting when clock skew exists between the different shift clocks. This can lead to mismatches and potentially bad silicon.

The error message is:

```
A lockup latch may be required between the following two consecutive
sequential elements of chain 'S':  (T24-1)
         Source sequential elements:        'N1' (G1)
           Cell position from chain output: M1
           Clock source:                    'C1' (CG1)
           Clock off-state:                 S1
           Clock polarity:                  P1
         Destination sequential elements:   'N2' (G2)
           Cell position from chain output: M2
           Clock source:                    'C2' (CG2)
           Clock off-state:                 S2
           Clock polarity:                  P2
```

N1 and N2 are the instance names of the adjacent elements clocked by different clocks. G1 and G2 are the corresponding gate IDs of the elements, CG1 and CG2 are the gate IDs of the clocks,

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

S1 and S2 are the clock off-states, and P1 and P2 are the clock polarity values. M1 and M2 are the cell IDs of the scan cells in which N1 and N2, respectively, are located. S is the name of the scan chain.

Note

If N1 or N2 is not within a scan cell, the corresponding M1 or M2 is the cell ID of the scan cell that is driven by N1 or N2.

Analyze T24 errors by using the DRC Browser in Tessent Visualizer or issuing the analyze_drc_violation command on the command line. The tool displays the corresponding scan cells that fail this check in the schematic window. You can correct the violation by inserting a lockup latch between the adjacent sequential elements reported in the violation message.

T25

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

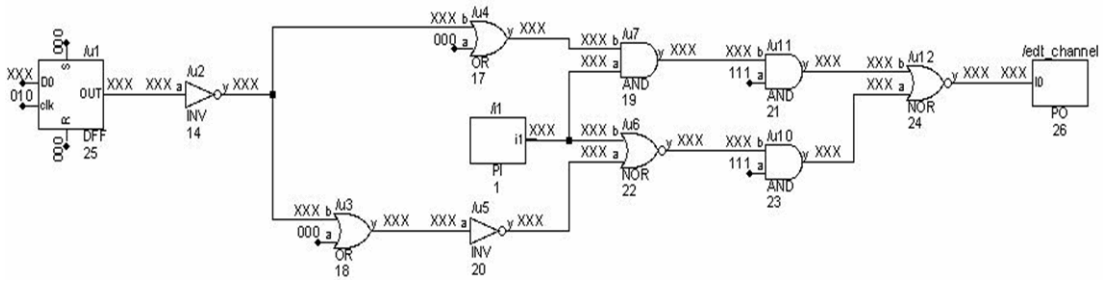
Default Handling: Warning

report_drc_rules: Supported

Scan chain tracing must be independent of whether or not a gate with an X value is initialized to a known value (either 0 or 1).

Figure 10-62 is an example of a T25 violation.

Figure 10-62. Example of T25 Violation



In Figure 10-62, the scan path is from DFF gate 25 to PO gate 26. If PI gate 1 is initialized to 1, then the actual scan path is the upper path from gate 25 to gate 26. If PI gate 1 is initialized to 0, then the actual scan path is the lower path from gate 25 to gate 26. In either case, the scan

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

shifting is successful; however, a simulation mismatch could occur in Verilog if PI gate 1 is not initialized (remains an X value). The warning message is:

```
There is a T25 violation. Scan chain tracing has assumed that some gates
with X values are initialized to any non-X values. This is not a problem
in silicon but may cause Verilog simulation mismatches. For more
conservative scan chain tracing, please use the command: "set_drc_handling
-scan_chain_tracing conservative".
```

You can display the T25 violation using the "report_drc_rules T25" command as shown in the following example:

report_drc_rules T25

```
Warning: Scan chain tracing between scan output and scan cell 0
(from gate 24 back to gate 14) in scan chain 0 assumes that some gates
with X values are initialized to any non-X values. (T25-1)
```

You can debug T25 violations by displaying the gate value in scan shift. To do this, use the set_gate_report command to set gate reporting to drc shift and use the report_gates command with the gate ID number displayed in the message.

The default handling for this rule violation is warning. If you do not want scan chain tracing to pass with a T25 violation, use the "set_drc_handling -scan_chain_tracing conservative" command.

T26

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

The shift procedure must create a sensitizable path through bidirectional pins at the scan chain output or EDT channel output. The improperly sensitized bidirectional pin causes an error condition.

For scan chain output, the tool issues the following error message:

```
Scan chain tracing failed at bidirectional scan chain output pin N in
chain "C". The pin is not forced to Z during measure_sco. (T26-1)
```

For EDT channel output, the tool issues the following error message:

```
Scan chain tracing failed at bidirectional channel output pin N in block
"B". The pin is not forced to Z during measure_sco. (T26-2)
```

N is the pin name, C is the corresponding chain name, and B is the corresponding EDT block name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can avoid this rule violation by adding a force to Z for the reported pins at the beginning of the load_unload procedure. Alternatively, you can force all bidirectional pins with a ‘force _ALL_BIDI Z’ at the beginning of the load_unload procedure. The _ALL_BIDI is a macro that selects all primary bidirectional pins.

You can also define the pin as an output pin instead of as an inout pin in your Verilog file. By doing this, then there is no need to force this pin to Z anymore.

T27

Category: Trace

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Multiple scan chain definitions must not share the same scan input and scan output pins. Correct this error condition either by deleting the extra identical scan chain definitions or by changing incorrect scan input and scan output pins for affected scan chains.

The error message is:

```
The following scan chain definitions are identical because they share the
same scan input 'P1' and the scame scan output 'P2':  (T27-1)
   'C1'
   'C2'
```

P1 is the name of the scan chain input pin, and P2 is the name of the scan chain output pin. C1 and C2 are the names of the scan chain definitions sharing those pins.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Power-Aware Rules (V Rules)

Power-aware designs rely on DRC rules that are designed specifically for that application.

You must load power data using the read_cpf or read_upf commands before using the report_drc_rules command for the V DRC rules.

When using the power-aware DRC rules, the following conditions apply:

- **Rule V1 to V7** — Checked after reading a power data file.
- **Rule V8 to V21** — Checked when switching from setup mode to a non-Setup mode.

The tool displays a summary of the V rules when you load power data into the tool without any parsing errors as shown here:

```
read_upf my_design.upf
report_drc_rules -summary

…
T26: #fails=0  handling=error  (bidirectional chain / channel outputs not
forced to Z during chain tracing)
V1: #fails=0  handling=ignore/verbose (no power modes are defined)
V2: #fails=0  handling=ignore/verbose (no default power modes are defined)
V3: #fails=0  handling=error  (multiple default power modes are defined)
V4: #fails=0  handling=error  (no default power domains are defined)
V5: #fails=0  handling=error  (multiple default power domains are defined)
V6: #fails=0  handling=warning (no restore_edge or default_restore_edge is
defined)
V7: #fails=0  handling=warning  (power mode is not reachable)
V8: #fails=0  handling=error  (scan data path is in a power-off domain)
V9: #fails=0  handling=error  (scan control path is in a power-off domain)
V10: #fails=0  handling=error  (power start/end/shutoff condition is X
during the scan shift period)
V11: #fails=0  handling=error  (active power mode has changed during the
scan shift period)
V12: #fails=0  handling=warning  (active power mode may change during the
capture cycle)
V13: #fails=0  handling=warning  (path crosses power domains without a
level-shifter cell)
V14: #fails=0  handling=warning  (path crosses power domains without an
isolation cell)
V15: #fails=0  handling=note  (scan chain contains both retention and
regular cells)
V16: #fails=0  handling=note  (scan chain contains both always-on and
other types of cells)
V17: #fails=0  handling=note  (EDT channel contains both retention and
regular cells)
V18: #fails=0  handling=warning  (power control signals are in a domain
that could be powered-off)
V19: #fails=0  handling=warning  (retention save signal is not off during
shift)
V20: #fails=0  handling=note  (non-scan retention cells are identified in
the design)
V21: #fails=0  handling=error  (power data assertions are detected)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

V1 3611
V2 3612
V3 3612
V4 3612
V5 3613
V6 3613
V7 3613
V8 3614
V9 3614
V10 3615
V11 3616
V12 3617
V13 3618
V14 3619
V15 3619
V16 3620
V17 3620
V18 3620
V19 3621
V20 3621
V21 3622

V1

Category: Power Aware

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

No power modes are defined when there are multiple power domains. You cannot change the handling.

The violation message is:

```
At least one power mode is required when multiple power domains are
defined (V1).
```

Because power modes in UPF are implicitly learned through power switches control logic, the V1 rule is ignored in UPF format.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

V2

Category: Power Aware

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

No default power modes are defined when there are multiple power modes. You cannot change the handling.

The violation message is:

```
A default power mode is required when multiple power modes are defined
(V2).
```

The V2 rule is ignored for UPF format.

V3

Category: Power Aware

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Multiple default power modes are defined. You cannot change the handling.

The violation message is:

```
Multiple default power modes are defined (V3).
```

V4

Category: Power Aware

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

No default power domain is defined when there are multiple power domains. You cannot change the handling.

The violation message is:

```
A default power domain is required when multiple power domains are defined
(V4).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

V5

Category: Power Aware

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Multiple default power domains are defined. You cannot change the handling.

The violation message is:

```
Multiple default power domains are defined (V5).
```

V6

Category: Power Aware

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Either restore_edge or default_restore_edge must be defined if the design contains any retention cells; if this rule is violated, the state retention rule is ignored and no retention value is restored.

Note that missing both save_edge and default_save_edge is permitted on CPF as long as restore_edge or default_restore_edge is defined. In this case, the inversion of restore_edge (or default_restore_edge when restore_edge is missing) is used as save_edge.

The violation message is:

```
No restore_edge is defined for the state retention rule name. (V6-n).
```

V7

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

A defined power mode is not reachable from the default power mode with the defined power transition rules.

The violation message is:

```
Power mode name1 is not reachable from the default power mode name2 using
any defined power transition rule (V7-n).
```

The V7 rule is ignored for UPF format.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

V8

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

The scan tracing should not propagate to a scan cell in a power domain that is in the shutoff state.

Note

Scan insertion in the dft -scan context supports this rule if the design contains pre-existing scan chains and a valid test procedure file. This rule checks the existing scan chains, but it does not affect the insertion of new chains.

Note that the number of power OFF gates in a given scan chain can be large, so only one V8 violation is recorded per scan chain and the first scan path gate from the sco pin is reported. Violation of the V8 rule can be due to at least one of the following reasons:

- The power mode definition is incorrect.
- The shutoff condition of the power domain is defined incorrectly.
- The test procedures (for example, load_unload or shift) for the scan chain operation are defined incorrectly.
- The inserted scan logic of the design does not consider the power mode properly.

The violation message is:

```
The scan path of chain chain-name is in power off domain domain-name (the
first power off scan path gate is N1 (G1) after tracing C cells ) (V8-n)
```

When you issue the analyze_drc_violation command on an occurrence of this violation, Tessent Visualizer displays the scan cell in the Flat Schematic window with a callout box attached containing the following message:

```
Power Domain: <name>, Power Mode (CPF) or Power State: <name>
```

V9

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

All control logic for the scan operation must be in a power ON domain during the entire scan shift operation.

Note

Scan insertion in the dft -scan context supports this rule if the design contains pre-existing scan chains and a valid test procedure file. This rule checks the existing scan chains, but it does not affect the insertion of new chains.

Some popular control logic examples are listed below:

- If EDT logic exists in the design, it should stay at power ON mode during the scan test when the compression mode is used; otherwise the scan operation fails.
- If the scan_enable signal is derived from a JTAG controller, the JTAG controller should stay at power ON mode during the scan test.
- The shift clock control logic should remain at power ON mode during the scan test. The tool performs the check by monitoring all test procedures except test_setup and ensures that the power domains where the scan control logic are located remain at the power ON mode after test_setup procedure.

The violation message is:

```
The scan chain chain-name is blocked at N1 (G1) after tracing C cells
due to the control gate N2 (G2) in power off domain domain_name (V9-n).
```

When you issue the analyze_drc_violation command on an occurrence of this violation, Tessent Visualizer displays the control gates in the Flat Schematic window with a callout box attached containing the following message:

```
Power Domain: <name>, Power Mode (CPF) or Power State: <name>
```

V10

Category: Power Aware

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

The expression used by the mode_transition rule and shutoff condition should not be X during the scan shift period. The tool performs the rule by checking all events to ensure that no expression can be X during the scan shift operation. The expressions to be checked include the shutoff condition of every power domain and the start and end conditions of every mode_transition rule.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Scan insertion in the dft -scan context supports this rule if the design contains pre-existing scan chains and a valid test procedure file. This rule checks the existing scan chains, but it does not affect the insertion of new chains.

Note

These errors prevent the tool from performing power aware good-machine circuit simulation, which impacts test point analysis.

The violation messages are:

Case 1 — When shutoff condition of a power domain is X:

```
The shutoff condition of power domain name is X at time t of procedure
procedure-name (V10-n).
```

Case 2 — When the start condition of a mode_transition rule is X:

```
The start condition of mode transition rule name is X at time t of
procedure procedure-name (V10-n).
```

Case 3 — When the end condition of a mode_transition rule is X:

```
The end condition of mode transition rule name is X at time t of procedure
procedure-name (V10-n).
```

The handling of X is different based on the handling of the V10 DRC rule. When the handling of the rule is set to warning, the condition of X is treated as HI as if the condition is satisfied. When the handling is Note, the condition of X is treated as LO as if the condition is not satisfied. Note, the violation of this rule may cause the design to be in the wrong power mode and thereby cause a simulation mismatch.

V11

Category: Power Aware

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

During scan chain shift operation, the design should not change power modes because this can cause some power domains to be switched between power OFF mode and power ON mode. This occurs because changing active power domains during the shift period may cause the scan chain configuration to be changed. The tool performs the rule by checking the shutoff expressions of all power domains to ensure they remain false during the scan shift period.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Scan insertion in the dft -scan context supports this rule if the design contains pre-existing scan chains and a valid test procedure file. This rule checks the existing scan chains, but it does not affect the insertion of new chains.

Note

These errors prevent the tool from performing power aware good-machine circuit simulation, which will impact test point analysis.

The violation messages are:

Case 1 — When mode transition rules are defined:

```
Active power mode changed from mode1 to mode2 at time t of procedure
procedure_name (V11-n).
```

Case 2 — When mode transition rules are not defined:

```
The power domain name changed from state-1 to state-2 at time t of
procedure procedure_name (V11-n).
```

When the handling of the violation is not error, the scan shift operation may be unreliable and this can result in a simulation mismatch. A mismatch may not be seen in the parallel testbench and this can make the debugging process more complicated.

V12

Category: Power Aware

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

ATPG should not create patterns that can turn off an active power domain during the capture cycle; turning off an active power domain during the capture period changes the scan chain configuration. The tool performs the rule by checking on active power domains with shutoff expressions and ensuring they cannot be true during the capture period.

The violation messages are:

Case 1 — When mode transition rules are defined:

```
The power mode may change from name1 to name1 during the capture cycle
(V12-n).
```

Case 2 — When mode transition rules are not defined:

```
The power domain name may change state during the capture cycle (V12-n).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When the shutoff condition is true and the handling of the violation is not error, the capture value of the scan cells in the shutoff power domain remains the same as if the power domain is still active. This may cause some test patterns to fail in silicon that may not be detected during the logic simulation phase.

V13

Category: Power Aware

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

A path should not cross different power domains without a level-shifter cell if the power domains can operate in different operation voltages.

The following situations can cause a V13 DRC violation:

- An instance is driven by another instance of a different voltage power domain.
- The scan_in pin or the EDT decompressor to the first scan cell crosses different voltage domains.
- The last scan cell to the scan_out pin or EDT compactor crosses different voltage domains.
- The scan cell (i+1) to scan cell i crosses different voltage domains.

The violation messages are:

Case 1 — When violation is on a non-scan path:

```
Path from gate N1 (G1) to gate N2 (G2) crosses different voltage domains
without a level-shifter cell (V13-n).
```

Case 2 — When violation is on a scan path:

```
Scan path from gate N1 (G1) of cell (i+1) to gate N2 (G2) of cell i of scan
chain chain-name crosses different voltage domains without a level-shifter
cell (V13-n).
```

When you issue the analyze_drc_violation command on an occurrence of this violation, Tessent Visualizer displays the path between the two gates reported in the message in the Flat Schematic window and displays the simulation data as a DRC in the Flat Schematic window. If possible, a callout box for the DRC is attached to the source and the sink containing the power domain names.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

V14

Category: Power Aware

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

If a path crosses two power domains and the driving power domain contains the shutoff condition, the path must include an isolation cell. Similar to rule V13, there are four situations to be considered for this rule.

The violation messages are:

Case 1 — When violation is on a non-scan path:

```
Path from gate N1 (G1) to gate N2 (G2) crosses different power domains
without an isolation cell (V14-n).
```

Case 2 — When violation is on a scan path:

```
Scan path from gate N1 (G1) of cell (i+1) to gate N2 (G2) of cell i of scan
chain chain-name crosses different power domains without an isolation cell
(V14-n).
```

When you issue the analyze_drc_violation command on an occurrence of this violation, Tessent Visualizer displays the path between the two gates reported in the message in the Flat Schematic window and displays the simulation data as a DRC in the Flat Schematic window. If possible, a callout box for the DRC is attached to the source and the sink containing the power domain names.

V15

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

A scan chain should not include both retention cells and regular scan cells. A violation of this rule makes retention test less efficient.

Note

Scan insertion in the dft -scan context supports this rule if the design contains pre-existing scan chains and a valid test procedure file. This rule checks the existing scan chains, but it does not affect the insertion of new chains.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The violation message is:

```
Scan chain chain-name contains both retention cells and regular cells
(V15-n).
```

This violation is not supported in Tessent Visualizer.

V16

Category: Power Aware

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

Typically, always-on cells are used to program power mode and retention mode. To make the control of the power mode more efficient, a scan chain should not include both always-on cells and other types of scan cells.

The violation message is:

```
Scan chain chain-name contains both always-on cells and other type of
cells (V16-n).
```

This violation is not supported in Tessent Visualizer.

V17

Category: Power Aware

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

An EDT scan channel should try to prevent the mix of scan chain with retention cells and, regular scan chain without retention cells. This rule prevents the retention cells from being masked by regular cells when the circuit is restored from the retention mode.

The violation message is:

```
Scan channel channel-name contains both retention scan chain and regular
scan chain (V17-n).
```

This violation is not supported in Tessent Visualizer.

V18

Category: Power Aware

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

All power control signals, including retention save, retention restore, isolation_enable, and power-domain shutoff conditions should be from always-on power domains that cannot be turned off.

By default, the V18 violation handling is Warning. The tool starts with an all power-off state, except for the always-on power domains.

The violation message is:

```
Shutoff condition 'condition_expression' for domain domain_name is in non-
always-on power domain (V18-n).
```

When the V18 violation handling is set to Note, the tool assumes it is in an all power-on state at the beginning of test_setup.

V19

Category: Power Aware

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Retention save signals should be off during the scan shift period to prevent the loading value from being overridden by the retention saved state.

The violation message is:

```
Save signal 'condition_expression' for retention rule rule_name is not off
during the scan shift period (V19-n).
```

V20

Category: Power Aware

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

Only scanned retention cells are supported by the retention test to test the retention functionality. So this rule checks if there are any non-scan retention cells.

The violation message is:

```
Retention cell cell_name is a non-scan cell (V20-n).
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This violation is not supported in Tessent Visualizer.

V21

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

The design should never enter any illegal power configurations specified in CPF command assert_illegal_domain_configurations during any test cycle.

The violation message is:

```
The active power mode violates assert_illegal_domain_configurations name
at time t of procedure proc_name (V21-n).
```

Note

CPF and UPF power data parsers support the Tcl scripting mechanism. The violation of the Tcl format results in an error. Note, the format of the error message is not described in this document.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Timing Rules (W Rules)

These rules apply to timeplates and the mapping of timeplates to procedures. This includes making sure that the order of events in a procedure is not changed after leaving setup mode.

For timing rules specific to the enhanced procedure file, see "Test Procedure File" in the *Tessent Shell User's Manual*.

W1 3624
W2 3624
W3 3625
W4 3625
W5 3625
W6 3626
W7 3626
W8 3627
W9 3627
W10 3627
W11 3628
W12 3628
W13 3629
W14 3629
W15 3629
W18 3630
W19 3630
W20 3631
W21 3631
W22 3632
W23 3632
W24 3632
W25 3633
W26 3633
W27 3634
W28 3634
W29 3635
W30 3635
W31 3636

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

W32 3636
W33 3637
W34 3638
W35 3638
W36 3639
W37 3639
W38 3640
W39 3641
W41 3642
W42 3642
W43 3643

W1

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

This message can only occur when not in setup mode.

After leaving setup mode, it is possible to load an enhanced procedure file using the read_procfile command. It is also possible for the enhanced procedure file to load a new procedure which overwrites a procedure already loaded. This is fine as long as the event order in the new procedure matches that of the old procedure.

If the new procedure has additional events, the tool issues this error message:

```
New procedure P has more events than existing procedure. (W1)
```

P is the name of the new procedure.

W2

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Similar to rule W1, but this rule is violated if the new procedure has less events than the old procedure.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The error message is:

```
New procedure P has fewer events than existing procedure. (W2)
```

P is the name of the new procedure.

W3

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Similar to rule W1, this rule is violated if the new procedure has a different event order than the old procedure.

The error message is:

```
New procedure P has a different event order then existing procedure. (W3)
```

P is the name of the new procedure.

W4

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

This message is similar to rule W1. Instead of an entire new procedure being loaded, the enhanced procedure file only specifies that a new timeplate is applied to an existing procedure. This is done by specifying a new procedure in the enhanced procedure file with only a timeplate reference statement.

If that new timeplate causes the event order in the old procedure to change, the tool issues this error message:

```
New timeplate T changes event order in procedure. (W4)
```

T is the name of the new timeplate.

W5

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

A new non-scan procedure has an event statement that either occurs in the wrong order, or is not allowed in that type of procedure.

All non-scan procedures must conform to the event order that is stated for them in "W5" in the *Tessent Shell User's Manual*. An example of this would be a capture procedure that has a "measure_po" statement before the "force_pi" statement.

The error message is:

```
Procedure P has an illegal event statement or event order S. (W5)
```

P is the procedure name and S is the illegal statement or event order.

W6

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

If one shift procedure has the measure_sco statement occurring after the pulse of the shift clock, then all shift procedures must place the measure_sco statement after the pulse of the shift clock, and all load_unload procedures must have a measure_sco statement occurring at the end of the cycle right before the apply shift statement. Placing the measure_sco statement after the pulse statement enables end measure mode which affects the way the Vector Interfaces code writes out parallel load scan patterns.

See the "Creating Test Procedure Files for End Measure Mode" section of the *Tessent Shell User's Manual*.

The error message is:

```
Shift procedure P [does | does not] use end measure, while previous ones
[do | do not]. (W6)
```

P is the procedure name.

W7

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

No timing information has been specified for the named procedure. Either the procedure needs to reference a timeplate, or time values must be associated with the event statements.

The error message is:

```
No timeplate or times specified for procedure P. (W7)
```

P is the procedure name.

W8

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

A scan procedure (shift, load_unload, …) has been read in by the read_procfile command, but no scan_group statement is in the procedure.

The error message is:

```
No scan group specified for procedure P. (W8)
```

P is the procedure name.

W9

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

The named procedure has no event statements, no apply statements, and no timeplate references.

The error message is:

```
No events in procedure P. (W9)
```

P is the procedure name.

W10

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_drc_rules: Supported

This warning is issued during DRC checking. It indicates that none of the procedure files specified in the add_scan_groups commands have any procedures in them.

The warning message is:

```
No test procedures have been loaded. (W10)
```

W11

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

A cycle in a procedure that needs to be split into two cycles in order to use the timeplate that is specified for that procedure.

This could occur, for example, if the test_setup procedure contains a pulse statement on a specific clock pin, and yet the timeplate used referenced by this procedure does not contain a pulse statement for the clock. The tool issues a P53 rule message and then a W11 rule message to indicate that the cycle (where the pulse statement is) was split into two cycles in order to create the clock pulse.

The warning message is:

```
Cycle S being split in procedure P. (W11)
```

S in the cycle and P is the procedure name.

W12

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

A timeplate was loaded that is missing a required statement. For example, a timeplate must have a force_pi statement. If a required clock pulse statement is missing from the loaded timeplate, the tool issues the following P53 rule message.

The warning message is:

```
No S statement in timeplate T. (W12)
```

S is the statement name and T is the timeplate.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

W13

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The event order for the name procedure has changed while parsing an procedure file, but since the tool is still in setup mode, this change of event order is acceptable. This could happen if two procedure files in two different add_scan_groups commands both have the same procedure (test_setup, for example), but one specifies a different event order than the other. The last one that was loaded is used.

The warning message is:

```
Event order has changed in procedure P. (W13)
```

P is the procedure name.

W14

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Not Supported

This message indicates that a procedure has been replaced by a new procedure with the same name, same type, and same scan group if applicable. This message indicates that the replacement did not violate any rules. Both procedures have the exact same event order, but they might use different timeplates.

The warning message is:

```
Procedure P replaces same procedure from file S. (W14)
```

P is the procedure name and S is the filename.

W15

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

End measure mode is enabled but an illegal force or pulse statement occurs between the measure_sco statement and the end of the procedure in a shift procedure, or the illegal statement

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

occurs between the measure_sco statement and the apply shift statement in a load_unload procedure.

See the "Creating Test Procedure Files for End Measure Mode" section of the *Tessent Shell User's Manual*.

The error message is:

```
Statement S cannot follow measure_sco in end measure procedure P. (W15)
```

S is the illegal statement and P is the procedure name.

W18

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (can be changed to Note or Ignore)

report_drc_rules: Supported

Many ASIC vendors require that shift procedures only be one cycle in length, and it is very unusual to encounter a multiple cycle shift procedure. This warning alerts the user that something might not be specified correctly in the shift procedure, such as forcing the same pin with more than one value, or pulsing a clock twice.

The warning message is:

```
Shift procedure has more than one cycle, please check if shift clocks are
defined properly. (W18)
```

W19

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

This message occurs when using the off-state option improperly. The off-state option is only supported when using the Enhanced Vector Interfaces. Also, the pin referenced with the off-state option cannot be used in a capture or clock_sequential procedure. These are considered non-scan procedures.

Any timeplate that uses the off-state statement is treated as a complex timeplate and is not allowed to be used in a non-scan procedure. Complex timeplates are most useful in shift procedures where a non-clock pin needs to be pulsed while still maintaining a single cycle in the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

shift procedure. If you attempt to pulse a signal specified with an off-state option in a capture or clock_sequential non-scan procedure, the tool displays the following error message:

```
Timeplate gen_tp1 with complex waveforms cannot be used in non-scan
procedure capture.  (W19)
```

W20

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This error occurs if, in a capture procedure, a force_pi event does not occur before the first clock pulse.

The error message is:

```
A force_pi statement does not exist before the first clock pulse in
Procedure P. (W20)
```

P is the procedure name.

To correct this violation, add a force_pi statement to the beginning cycle of the procedure, before the first clock pulse. For a named capture procedure with internal and external modes, add a force_pi statement to the beginning cycle in each mode.

W21

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

In a named capture procedure, you can only have a measure_po statement in the last cycle of the internal mode. This error occurs if a measure_po statement is found in a cycle where it is not permitted.

The error message is:

```
Procedure P has one or more measure_po statements which do not occur in
the cycle before the last clock pulse. (W21)
```

P is the procedure name.

To correct this violation, remove the measure_po statements from the cycles where they are not permitted.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

W22

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

In a named capture procedure, you can only have a measure_po statement in the last "cyclized" ATPG cycle of the internal mode—before the last clock pulse. This warning alerts you that the tool did not find a measure_po statement in this location. A capture procedure with this warning does not use any PO as a capture point.

The warning message is:

```
Procedure P has no measure_po statement in the cycle before the last clock
pulse. (W22)
```

P is the procedure name.

To view the cyclized information for the procedure, use the report_capture_procedures command.

W23

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

In a named capture procedure, the total period of the external mode must match the total period of the internal mode. This warning alerts you that the external mode's period is longer than the internal mode's period.

The warning message is:

```
The total period for the external mode is longer than the total period for
internal mode in procedure P. (W23)
```

P is the procedure name.

To eliminate this warning, modify the timeplate(s) so that the external mode period matches the internal mode period.

W24

Category: Timing

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

In a named capture procedure, the total period of the external mode must match the total period of the internal mode. This error occurs if the internal mode's period is longer than the external mode's period.

The error message is:

```
The total period for the internal mode is longer than the total period for
external mode in procedure P. (W24)
```

P is the procedure name.

To correct this violation, modify the timeplate(s) so that the external mode period matches the internal mode period.

W25

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Event times in the external and internal modes of a named capture procedure should match. This warning alerts you that an event in the external mode does not have a matching event in the internal mode.

The warning message is:

```
// Warning: External event pulse S at time T needs to have a matching
// internal event in procedure P or the signal needs to be disconnected
// from internal sequential elements or observe points. (W25-1)
```

Where S is the signal name, T is the time of the event, and P is the procedure name.

In Tessent Diagnosis, these violations if uncorrected can cause pattern verification errors in the tool. See the *Tessent Diagnosis User's Manual* for more information.

W26

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Event times in the external and internal modes of a named capture procedure must match. This error occurs if an event (other than an internal pin event) in the internal mode does not have a matching event in the external mode.

The error message is:

```
Event E N at time T in internal mode does not have matching event in
external mode in procedure P. (W26)
```

E is the type of event, N is the name of the pin to which the event applies, T is the time of the event, and P is the procedure name.

To correct this violation, you must do two things:

- Add the missing event to the external mode.
- Modify the timeplate(s) so that the event added to the external mode occurs at the same time as in the internal mode.

W27

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

In a named capture procedure, the start time for the scan load cycle in the internal mode must have a match in the external mode. This error occurs if there is not a match in the external mode.

The error message is:

```
Start time for scan load cycle in internal mode does not have a match in
external mode in procedure P. (W27)
```

P is the procedure name.

To correct this violation, adjust the timeplate(s) or procedure so that the internal scan load cycle has a corresponding external cycle starting at the same time.

W28

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In a named capture procedure, the measure_po time in the external mode must match the internal mode. This error occurs if the measure_po time in the external mode does not match the internal mode.

The error message is:

```
Measure_po time in external mode does not match internal mode in procedure
P. (W28)
```

P is the procedure name.

To troubleshoot this violation, check that there is a measure_po statement in both the internal mode and the external mode. If a measure_po exists in both modes, adjust the timeplate(s) to ensure the measure_po occurs at the same time in both modes.

W29

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

In a named capture procedure, the force_pi time in the external mode must match the internal mode. This error occurs if the force_pi time in the external mode does not match the internal mode.

The error message is:

```
Force_pi time in external mode does not match internal mode in procedure
P. (W29)
```

P is the procedure name.

To troubleshoot this violation, check that there is a force_pi statement in both the internal mode and the external mode. If a force_pi exists in both modes, adjust the timeplate(s) to ensure the force occurs at the same time in both modes.

W30

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

In a named capture procedure, an event on an internal pin is not allowed in the external mode. This warning alerts you that there is an internal pin event in the external mode of the procedure.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The warning message is:

```
Event E N on internal pin is not allowed in external mode of procedure P.
(W30)
```

E is the type of event, N is the name of the pin to which the event applies, and P is the procedure name.

The tool ignores the internal pin event in the external mode; however, you can avoid the violation by removing the internal pin event from the external mode.

W31

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

A named capture procedure that has an external mode must also have an internal mode. This error occurs when an external mode exists without an accompanying internal mode.

The error message is:

```
Procedure P has an external mode but is missing an internal mode. (W31)
```

P is the procedure name.

To correct this violation, do one of the following:

- Add an internal mode definition to the named capture procedure, or
- Remove the “mode external =” statement and its corresponding “end;” statement from the named capture procedure.

W32

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

A force_pi in a named capture procedure must not occur when a clock is in an on state.

W32 ensures the procedure file is valid.

This DRC checks the force_pi statement in named capture procedures in the procedure file when loading a new procedure file, which can occur during DRC, when the read_procfile

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

command is used, or when the write_patterns command is used with a new procedure file specified on the write_patterns command line.

Messaging

If the procedure file fails W32, the following message displays:

```
A force_pi statement occurs while a clock or control is active in
procedure P (W32).
```

Where P is the name of the procedure file.

Troubleshooting

To correct this violation, edit the procedure file so the force_pi statement does not occur during the time that a clock is in its on state.

W33

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

The named capture procedure in the procedure file cannot be loaded using the -Append_or_timing_update switch in the read_procfile command.

W33 ensures the procedure file is valid.

This DRC checks a new procedure file when it is loaded with the read_procfile or write_patterns command. A W33 violation occurs when the new procedure file contains a new scan procedure (load_unload, alternate shift, master_observe, and so on) or a new named capture procedure. W33 is only issued for new named capture procedures if the -Append_or_timing_update switch is specified with the read_procfile command.

Messaging

If the test_setup procedure file fails W33, the following message displays:

```
Cannot add new procedure P, procedure ignored (W33)
```

Where P is the procedure name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Troubleshooting

To correct this violation, do one of the following:

- If the violation is for a named capture procedure, either change the read_procfile command option from -Append_or_timing_update to -Replace or edit the procedure file to remove the named capture procedure.
- If the violation is for a scan procedure, edit the procedure file to remove the scan procedure.
- For both cases, you can return to setup mode and load the procedure file with the add_scan_groups command.

W34

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This error occurs when saving patterns if a force event in the external mode conflicts with a pin constraint in a named capture procedure.

The error message is:

```
// Error: Event E N at time T in external mode conflicts with pin
// constraint on same pin in procedure P. (W34)
```

Where E is the event type, N is the name of the event pin, T is the time at which the conflict occurs, and P is the name of the procedure that contains the conflicting pin constraint.

To resolve this issue, either the pin constraint or the named capture procedure must be changed. If this error is ignored, you cannot load external mode STIL or WGL patterns back into the tool because the pattern data (using the force event) does not match the pin constraint.

W35

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

This error occurs when a pulse-always clock needs to be added to a cycle, and that cycle uses a timeplate that has no pulse timing for the clock.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The error message is:

```
// Error: The following occured at line N in file F
// Pulse-always clock C is not pulsed in T. (W35)
```

Where N is the line number, F is the filename, C is the clock name, and T is the timeplate name.

The W35 rule is an error, and the handling cannot be changed.

W36

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This error occurs when “offstate” statements in the timeplates used by a named capture procedure are defined for a pin that has already been defined as a clock in the dofile. The unnecessary “offstate” statements cause extra pulses in a cycle.

The error message is:

```
// Error: Offstate cannot be used with pin N already defined as a
// clock (W36)
```

Where N is the name of the clock pin.

To resolve this error, remove the unnecessary “offstate” statements from the timeplates.

W37

Category: Timing

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Each load cycle of a named capture procedure must include a force_pi event. The ATPG tools require a force_pi event after each scan load. If there is no force_pi event in the load cycle of a named capture procedure, the generated test patterns may be incorrect and produce errors when read back into the ATPG tools.

The W37 violation does not override or mask a W20 violation. Therefore, if this error occurs in the first cycle and there is no force_pi event before the first clock pulse, then both a W37 and W20 violation are issued.

The W37 rule can be reduced to a warning or note only if every signal having a force event in a load_unload procedure has a pin constraint defined with a matching value.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The occurrence message is:

```
// Error: A force_pi statement does not exist in a scan load cycle
// in Procedure P. (W37-1)
```

Where P is the name of the procedure.

Troubleshooting

Edit the load cycle(s) of the named capture procedure to include a force_pi event.

W38

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

This rule checks whether pulse-always clocks that are pulsed in the external mode of a named capture procedure and are internally connected also pulse in the internal mode of the named capture procedure at the same time.

The error message is:

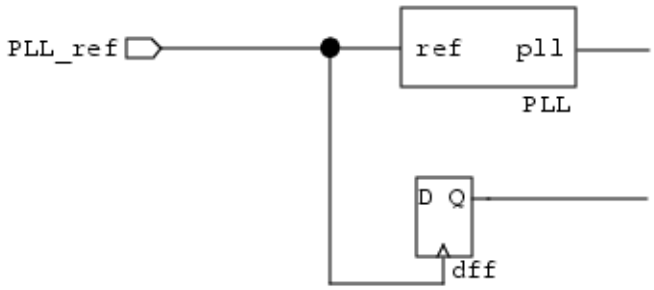
```
// Error: External event  E on pulse-always clock at time T needs to have
// matching internal event in procedure P or clock be disconnected from
// internal sequential elements or observe points.
```

Where E is a combination of the event type command and the signal in question. For example, if MyClock was the pulse-always clock, E would be “pulse MyClock”. T is the time at which the conflict occurs, and P is the name of the procedure that contains the unmatched external event.

A PLL clock connected to a sequential cell can also produce a W38 violation. Figure 10-63 shows the type of circuit that would cause a W38 error if there is no matching internal event. For this example, assume that the clock is defined with the following command:

> add_clocks 0 PLL_ref -pulse_always

Figure 10-63. W38 Reference Circuit



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

How to Correct W38 Violations

You can correct a W30 violation by either adding a matching internal event for every external mode event, or adding a cut point to disconnect the clock from sequential elements and observe points.

This rule defaults to error handling. If you set the rule to warning handling, the tool attempts to auto-correct the W38 problem by copying external mode event(s) to the appropriate time in the internal mode. If the auto-correct fails (for example, a W35 violation in the internal mode), then the W38 rule's handling is changed to an error by the tool.

W39

Contexts Supported: All contexts

Default Handling: Error

report_drc_rules: Supported

This rule ensures that all test clocks (TCKs) defined in ICL have consistent timing.

When you have an elaborated ICL, regardless of context, W39 is checked during system mode transition, after the read_procfile command. This ensures that the waveforms of all TCK ports, in each timeplate, adhere to these conditions:

- All TCK ports must have the same off state.
- The TCK ports cannot have more than one pulse.
- The waveforms of all TCK ports must either have one pulse or no pulse.

If there is no specific timing for the clock, the tool uses the pulse_clock timing. By default, the W39 DRC triggers an error, but may be set to trigger a warning.

W39 can report three error conditions. The error messages are:

```
//  Error: The waveform of TCK port C has multiple pulses.
//  TCK waveform cannot have more than one pulse. (W39-1)
```

Where C is the TCK port name.

```
//  Error: Inconsistent TCK timings between the 2 TCK ports: C1 and C2.
//  All TCK portss must have the same off state. (W39-1)
```

Where C1 and C2 are the TCK port names.

```
//  Error: Inconsistent TCK timings between the 2 TCK ports: C1 and C2.
//  The waveforms of all TCK ports must either have one pulse or no pulse
//  at all. (W39-1)
```

Where C1 and C2 are the TCK port names.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

W41

Contexts Supported: All contexts

Default Handling: Error

report_drc_rules: Supported

Flags the improper use of timeplates that pulse frequency divided clocks within capture procedures.

The W41 rule check defaults to an error, but its handling can be changed to warning, note, or ignore. The message has the following format:

```
//  The following occurred at line 53 in file
//  ../data/design_with_chains_modified_error.testproc:
//  Error: Event 'pulse user_added_clk_1' at time 100 conflicts with
//  frequency divided clock in procedure my_ncp_proc. (W41-1)
```

W42

Contexts Supported: All contexts

Default Handling: Error (can be changed to Warning, Note, or Ignore)

report_drc_rules: Supported

This rule ensures that any timing variables resolve to a valid time unit after computing timing expressions.

When pure numbers are used in a timing variable expression, they are adjusted to have units or not have units when computing the expression to ensure that the variable resolves to having a correct unit. However, when previously declared variables or timing variables are used in an expression, they cannot be adjusted. This can potentially lead to invalid units in a timing variable.

The error message has the following format:

```
//  The following occurred at line <num> in file <filename>:
//  Error: Expression for time variable '<varname>' doesn't resolve to
//   proper time unit. (W42-<n>)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following examples illustrate some variable definitions that can trigger a W42 error and some that do not:

```
set time scale 1 ps ;
timing_variables =
  t_var1 = 10; // Uses the time scale unit, so 10 ps
  t_var2 = (1000 / t_var1); // Results in W42. Because the 1000 occurs
      // first, it uses the time scale unit. Then it is divided by the
      // time scale unit from t_var1, resulting in a unitless value
  t_var3 = 20;
  t_var4 = (t_var1 * t_var3); // Results in W42, because the final unit
      // is ps²
  t_var5 = (t_var1 * 20); // Valid expression; "20" is adjusted to have
      // no unit to correctly interact with t_var1
end;
```

The following examples correct the errors from the previous examples:

```
set time scale 1 ps ;
variables =
  var1 = 10; // No units, just pure numbers
  var3 = 20;
end;
timing_variables =
  t_var1 = 10;
  t_var2 = (1000 / var1); // Valid expression; this uses the time scale
      // unit
  t_var4 = (t_var1 * var3); // Valid expression; t_var1 has a unit of ps
      // and var3 is unitless, so this resolves to ps
end;
```

W43

Category: Timing

Contexts Supported: dft -edt, dft -scan, dft -test_points, patterns -ijtag, patterns -scan, patterns -scan_retargeting

Default Handling: Warning (can be changed to Error, Note, or Ignore)

report_drc_rules: Supported

This rule occurs when a named capture procedure (NCP) or clock control (CC) is replacing a nonequivalent NCP or CC (that is, one that changes more than just the timing).

The warning messages are:

```
//  The following occurred at line L in file F:
//  Warning: New clock control C is not equivalent to existing clock
//   control. (W43-1)

//  The following occurred at line L in file F:
//  Warning: New procedure P is not equivalent to existing procedure.
//  (W43-1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

where L is the line number, F is the procedure file, C is the clock control name, and P is the procedure name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

X-bounding Rules (XB Rules)

The X-bounding DRC rule violations occur when an X-source cannot be bounded.

X-bounding violations occur when the analyze_xbounding command has been run, and the analysis failed to find suitable bounding locations for some X-sources. In the dft -test_points context, the analyze_test_points command may also perform the X-bounding analysis and produce X-bounding violations.

XB1 3645
XB2 3646

XB1

Category: X-bounding

Contexts Supported: dft -scan, dft -test_points

Default Handling: Error

report_drc_rules: Supported

Scannability rule XB1 detects and reports failures to bound static X-sources.

Failure to bound a static X-source results in an XB1 DRC violation. By default the default handling is set to “Error” which means that the analyze_xbounding command reports the DRC violations and exits with an error status. You can change this and set the default handling to “Warning”.

The tool generates one of the following DRC violations:

Path between X-source and observation point is blocked by no_control_point attribute

```
//  Warning: Could not insert X-bounding hardware on the path between
//      the static X-source at <source_gate_pin> and the potential observation point
//      at <end_gate_pin>. Please check the no_control_reason attribute for the
//      gate-pins on the path.. (XB1-1)
```

Path between X-source and clock network is blocked by no_control_point attribute

```
//  Warning: Could not insert X-bounding hardware on the path between
//      the static X-source at <start_gate_pin> and the gate-pin at <end_gate_pin> which
//      is on the clock network. Please check the no_control_reason attribute for the
//      gate-pins on this path. (XB1-2)
```

Path between X-source and clock gater is blocked by no_control_point attribute

```
//  Warning: The static X-source at <start_gate_pin> reaches <end_gate_pin>
//      which is the FE port of a clock gater. Could not insert hardware because both
//      the FE and TE pins are marked with the no_control_reason attribute. (XB1-3)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

XB2

Category: X-bounding

Contexts Supported: dft -scan, dft -test_points

Default Handling: Error

report_drc_rules: Supported

Scannability rule XB2 detects and reports failures to bound dynamic (multi-cycle-path or false-path related) X-sources.

Failure to bound a dynamic X-source results in an XB2 DRC violation. By default the default handling is set to “Error” which means that the analyze_xbounding command reports the DRC violations and exits with an error status. You can change this and set the default handling to “Warning”.

X-bounding of false/multi-cycle paths is done by inserting an inverting loopback path at any flop that would capture the value from the path. The tool does not search for alternate locations to inject the hardware if the port is not directly accessible.

The tool generates one of the following DRC violations:

Data input port of library cell with multi-cycle path marked with no_control_point attribute

```
//  Warning: Could not insert X-bounding hardware at <gate_pin> which is the data input port
//      of a library cell and is part of a false or multi-cycle path. Please check
//      the no_control_reason for this location. (XB2-1)
```

Unable to find data input (all library cell inputs are apparently on scan path)

```
//  Warning: Could not identify the data path input for the library cell containing
//      the memory element at /dff115/D (382.4). X-bounding logic cannot be inserted in
//      the scan path so this is a fatal error. This could be caused by missing
//      pin constraints and/or scan_enable connections. (XB2-1)
```

Set/reset input port of library cell with multi-cycle path marked with no_control_point attribute

```
//  Warning: Could not insert X-bounding hardware at <gate_pin> which is set or reset port
//      of a library cell and is part of a false or multi-cycle path. Please check
//      the no_control_reason attribute for this location. (XB2-1)
```

Could not fix false path at clock gater FE pin because TE pin is marked with no_control_point attribute

```
//  Warning: Could not insert X-bounding hardware at <gate_pin> which is test enable port
//      of a clock gating cell where the gate driving the functional enable port is
//      part of a false or multi-cycle path. Please check the no_control_reason attribute
//      for this location.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Other DRC Messages

DRC generates various messages to indicate the status of various processes and state elements.

Some of them are discussed here.

Oscillation Limitation 3647
RAM Summary Results and Test Capability 3647

Oscillation Limitation

The DRC simulator limits the maximum number of iterations performed to 500 while stabilizing state changes. If state elements continue to change after the maximum number of iterations is reached, the tool displays a message warning that the limit was reached. Additionally, the message reports each state element set to X to limit the iterations. State elements include latches, flip-flops, and feedback buffers.

The warning message is:

```
S forced to X in procedure P at time T due to iteration limit.
```

Where S is the state element, P is the procedure type, and T is the time.

Example

```
// latch1/Q (151) forced to X in procedure load_unload at
//     time 25 due to iteration limit.
```

RAM Summary Results and Test Capability

Upon completion of RAM rules checking, DRC displays summary result information as well as test capability information.

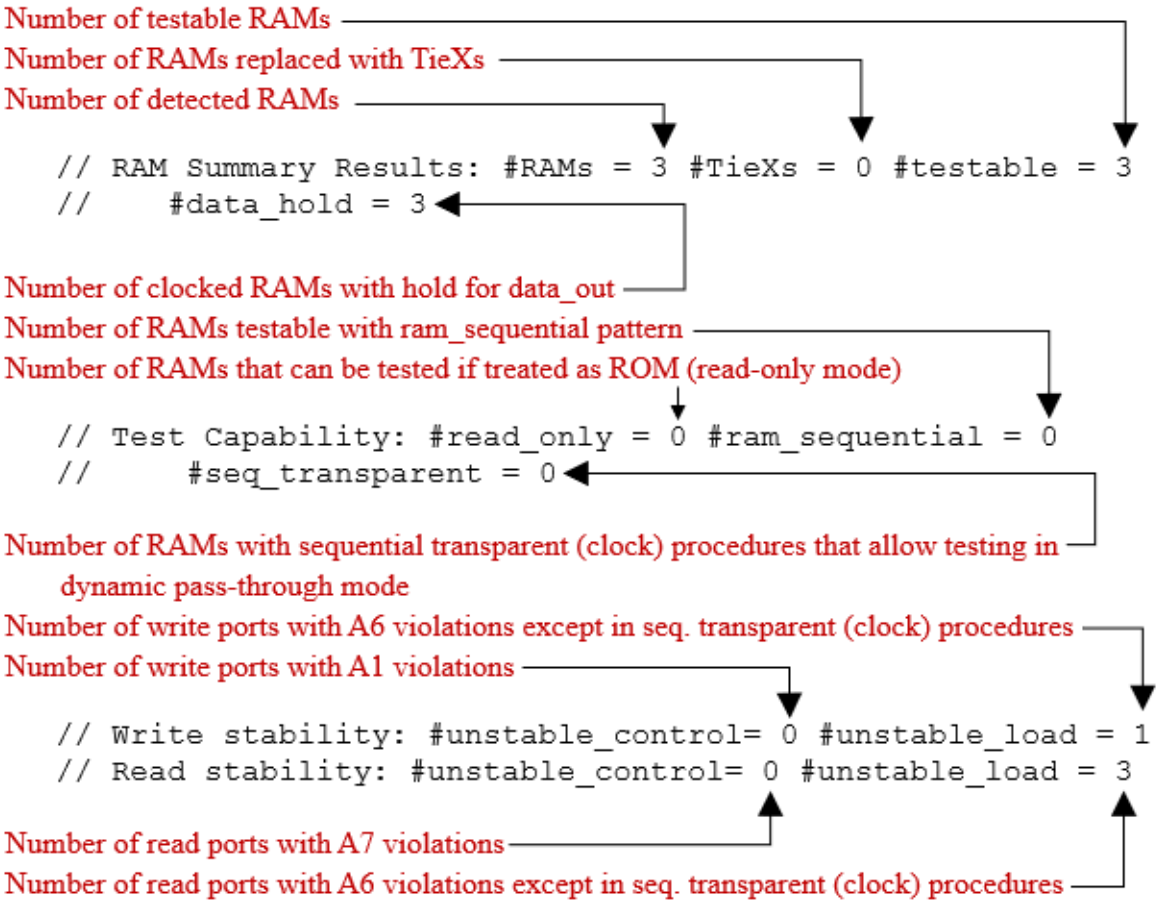
Example

```
// RAM Summary Results: #RAMs = 3 #TieXs = 0 #testable = 3
//     #data_hold = 3
// Test Capability: #read_only = 0 #ram_sequential = 0
//     #seq_transparent = 0
// Write stability: #unstable_control = 0 #unstable_load = 1
// Read stability:  #unstable_control = 0 #unstable_load = 3
```

Figure 10-64 describes the information displayed after running RAM rules checking.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-64. RAM Summary Results and Test Capability Messages



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 11
Data Models

Tessent Shell has an internal database that is organized into data models which are broad categories of design information (such as hierarchical design data). Each of the data models contains one or more types of design objects (such as pins and modules), and each design object has a list of attributes (such as an ID or bit-width of a bus).

For additional information about the design data models and how to work with them, refer to "Design Data Model, Attributes, and Introspection" in the *Tessent Shell User's Manual*.

You can access the pre-defined attributes on those design objects using the *_attribute* tool commands listed in Table 12-1 on page 3829. Those commands also provide a way for you to register new user-defined attributes for those design objects in the database.

The following sections describe the various data models, the types of objects they can contain, and the attributes associated to the objects:

Hierarchical Design Data Model 3651
Module 3654
Instance 3668
Port 3683
port_bundle 3704
Pin 3710
pin_bundle 3720
Net 3728
net_bundle 3736
Bundle Examples 3742
pseudo_port 3745

Flat Design Data Model 3749
Flat Design Object Types 3750

ICL Data Model 3764
ICL Object Types 3766
ICL File Examples With Complex Ports 3806

Scan Data Model 3813
Scan Element Object Type 3813
Scan Chain Family Object Type 3820
Scan Mode Object Type 3820

Test Point Data Model 3821

Configuration Data Model 3823
config_csv_wrapper 3823

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

config_data_wrapper 3824
config_property. 3825
config_repeatable_property 3825
config_wrapper 3826

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Hierarchical Design Data Model

The hierarchical design data model contains the following object types: module, instance, port, port_bundle, pin, pin_bundle, net, net_bundle, and pseudo_port.

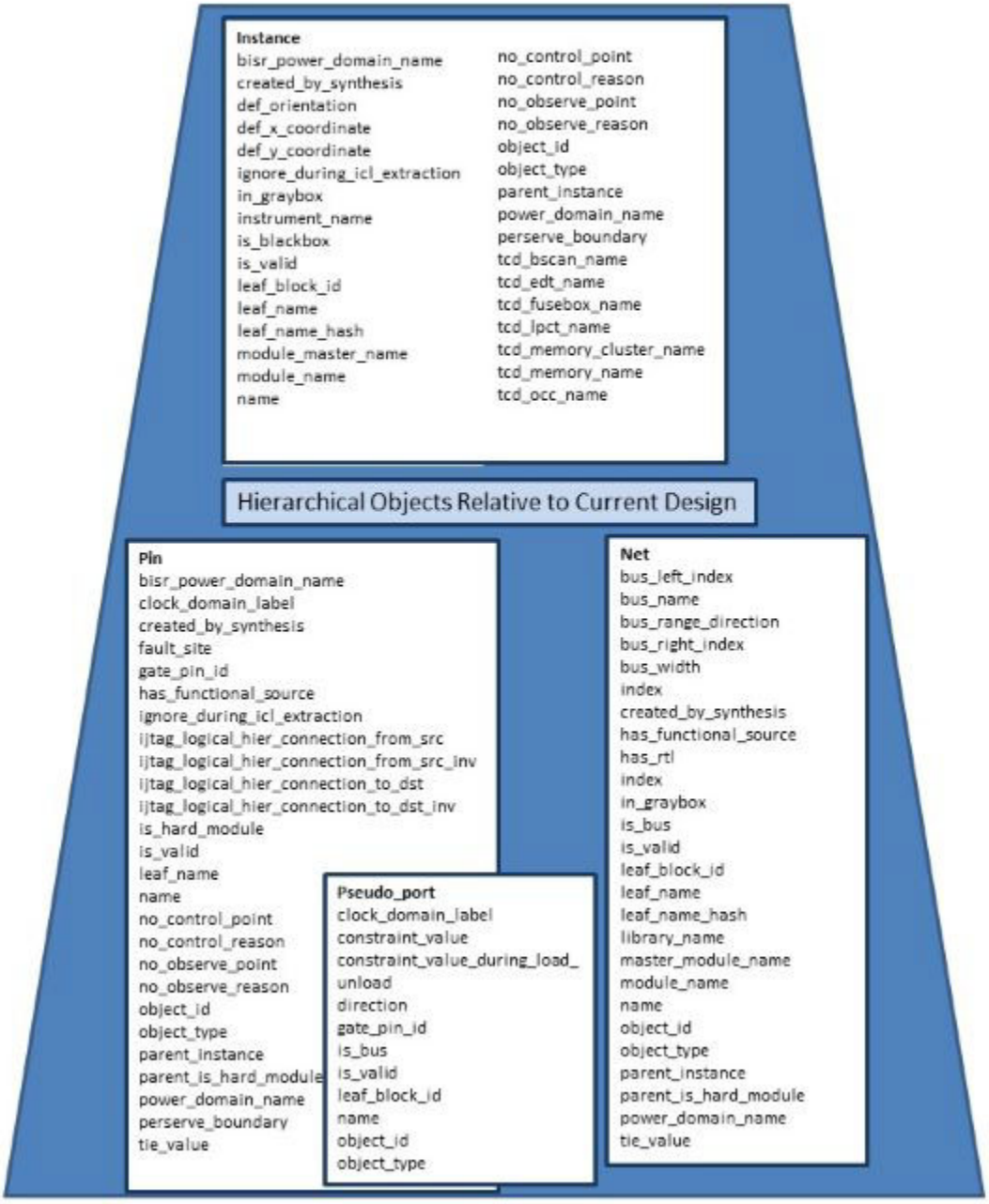
As suggested by Figure 11-1, instances, pins, and nets are hierarchical objects that exist relative to the current design. Ports and modules, shown in Figure 11-2, are non-hierarchical objects.

The figures also show the attributes that can be associated with the design objects.

The *current design* is the design specified by the set_current_design command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 11-1. Tessent Shell Hierarchical Design Objects



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 11-2. Tessent Shell Non-Hierarchical Design Objects

The hierarchical design object types are described on the following reference pages:

- Module
- Instance

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Port
- port_bundle
- Pin
- pin_bundle
- Net
- net_bundle
- pseudo_port

Reference information about attributes is provided in section "Attributes" on page 3829.

Module 3654
Instance 3668
Port 3683
port_bundle 3704
Pin 3710
pin_bundle 3720
Net 3728
net_bundle 3736
Bundle Examples 3742
pseudo_port 3745

Module

A module definition can come from a Verilog module, a VHDL entity, a Tessent library model, or it can be a built-in primitive. A module has a simple non-hierarchical name.

The module object type interacts with attributes as follows:

- Attributes are automatically visible as read-only attributes on instances if the same attribute is not also registered on the instance object type. Such attributes remain stored on the module definition and not on the instance itself
- Attributes with the applies_to_child_instances option set to On are visible to all child instances below the module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Built-In Attributes

The module object type has the following attributes:

active_child_scan_mode	When a child core exposes scan chain segments from several scan modes, this attribute makes the existing segments from a specific child scan mode active. During DRC, these active segments are the ones that get traced if tracing is enabled. Scan segments of the “active” child scan mode get the “state” attribute “usable”, while segments from other child scan modes are attributed with the “state” “unusable_child_mode”. Accordingly, during analysis, when creating scan chain families and/or adding scan modes, this attribute is also used to control their populations. If the tool cannot uniquely resolve the active scan mode for an instance by using the rules below, then the “state” attribute is set to “unresolved_child_mode”. The tool determines the effective “active_child_scan_mode" attribute for scan segments of a multi-mode instance using the following rule sequence: 1. The mode corresponding to the “active_child_scan_mode” if it is set for the instance and matches one of its child modes. 2. The mode of type “external” if “active_child_scan_mode” is not set for the instance and a unique child mode of type “external” exists. 3. The mode of type “unwrapped” if “active_child_scan_mode” is not set for the instance and a unique child mode of type “unwrapped” exists.
cell_type	String that reports characteristics of the cell. This attribute is initially null and is populated by the set_current_design command with information extracted from the cell library files. For example, any cell described as a pad in the library has cell_type set to “pad”. For details see *Model-Level Attribute Descriptions in the Tessent Cell Library Manual*.
def_x_coordinate_list	For the top module of the current design, this attribute is a list of all the x coordinates from a specified def file when you execute the read_def command.
def_y_coordinate_list	For the top module of the current design, this attribute is a list of all the corresponding y coordinates of the die from a specified def file when you execute the read_def command.
def_xl_coordinate	For the top module of the current design, the lower left x coordinate from a specified def file when you execute the read_def command. For other modules, the lower left x coordinate from a specified LEF file when you execute the read_lef command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

def_yl_coordinate	For the top module of the current design, the lower left y coordinate from a specified def file when you execute the read_def command. For other modules, the lower left y coordinate from a specified LEF file when you execute the read_lef command.
def_xh_coordinate	For the top module of the current design, the upper right x coordinate from a specified def file when you execute the read_def command. For other modules, the upper right x coordinate from a specified LEF file when you execute the read_lef command.
def_yh_coordinate	For the top module of the current design, the upper right y coordinate from a specified def file when you execute the read_def command. For other modules, the upper right y coordinate from a specified LEF file when you execute the read_lef command.
design_id	A string attribute that is set to reflect its design_id. It is set by read_design when reading a design. It also gets set after elaboration on the current design to the value returned by get_context -design_id when the value is not null.
design_level	A string attribute that is set to reflect its design_level. It is set by read_design when reading a design. It also gets set on the current design when the set_design_level command is run. The value is set to one of the following: • chip • physical_block • sub_block • instrument_block
exclude_from_concatenated_netlist	Boolean that specifies that the given module is a hard macro and should not be included in the concatenated netlist.
exclude_from_synthesis	Boolean that specifies that the given module is excluded from any synthesis. During ICL extraction and other tasks, the combinational path between the IJTAG port of the modules having a corresponding ICL module is traced and any module encountered along the way is synthesized. The pre-tracing step that does the synthesis performs a structural tracing of the input or output cone of a port. In the opposite case, following the flat model tracing only follows the controllable paths. Therefore in some cases the synthesis on the modules is not required and may take a long time, for example, for large memories that were not defined as memories. The exclude_from_synthesis attribute can be used to shorten the pre tracing time in that case.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

file_path_name	String that specifies the pathname to the file where a given module is defined. This attribute is null for created modules or modules without a definition.
format	Contains the value of the -format option as it was specified by the read_verilog or read_vhdl command when the module was read in. This value is only defined for modules with the "language" attribute set to "verilog" or "vhdl"; if the language attribute is set to "tessent_lib", an empty string is returned. See the "language" attribute for more information. For no_rtl and created modules, "2001" is the value for "verilog".
has_async_set_reset	Boolean that is true when a module has an asynchronous set port (port with active_high_set or active_low_set pin function) and/or an asynchronous reset port (port with active_high_reset or active_low_reset pin function).
has_def	Boolean that indicates whether or not the DEF file of the module was read in.
has_no_definition	Boolean that is true when a module does not have a definition, but a blackbox model was created during elaboration from its instantiation. You cannot have this type of module before creating the flat module unless you identify the module as a blackbox using the add_black_boxes command.
has_ram_or_rom_primitives	Boolean that is true when a module contains a _ram, _rom, or _cram primitive at any level within its hierarchy.
ijtag_is_icl_module	Boolean that identifies a design module as an ICL module. This attribute is only set in the patterns -ijtag context.
instrument_name	String that is set to the ICL module name it matches when its ijtag_is_icl_module attribute is true. The default is null.
is_blackbox	Boolean that returns true if the module was blackboxed in one of the following manners: • The add_black_boxes -module command was run on the module • The add_black_boxes -auto switch is used • The module was read using read_verilog -interface_only or read_vhdl -interface_only A false value is returned otherwise.
is_created	Boolean that specifies whether the module was created using the create_module command in insertion mode. In -rtl mode, this attribute is set as soon as the module is created and remains set until the insertion state is left. In -no_rtl mode, this attribute is set once when the module is created and never reset.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_elaborated	Boolean that specifies whether the module is part of an elaborated design. When set to true, the module is part of the elaborated design, that is, it is either the current design module or there is at least one instance of this module.
is_graybox	Boolean that is set to true for the top module loaded with read_design -view graybox. When reading a graybox module of a child physical block, the exclude_from_concatenated_netlist attribute is set on the module such that if you call the write_design command, the child physical graybox netlist is not concatenated within the parent physical netlist. When using the write_design -graybox command, elements that the tool finds within the child physical graybox may need to be preserved in the parent physical region's graybox. This typically happens when a clock is sourced from a PLL found inside the child physical region. The is_graybox attribute set on that module is used by the write_design -graybox command to ignore the exclude_from_concatenated_netlist attribute for that module.
is_hard_module	Boolean that specifies the module cannot be edited when set to true. This attribute is automatically set to true for all modules with type = "cell" or "primitive". This attribute is set for design modules in the dofile using the following command: set_attribute_value -name is_hard_module. This property has applies_to_child_instances set to On by default. The setting of this attribute also has on effect on the is_non_editable instance attribute.
is_incomplete	Boolean that specifies that the module cannot be edited and/or written out when set to true. This attribute is set to true when the module was read with the read_verilog or read_vhdl -interface_only commands, or when the module includes RTL constructs that are not supported by the set_context -no_rtl option. The is_hard_module attribute automatically returns true for any module with the is_incomplete attribute set to true. To write out a module with only its port and parameter definition, read it in normally and write it back out using the write_design -interface option.
is_internal_scan_only	Boolean that specifies that the module is internal scan only.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_internal_scan_only_reason	String that specifies the reason the instance is internal scan only as follows: • imported_from_tcd_scan — The object was inferred to be internal scan only because it has a Core/Scan TCD description that sets the internal_scan_only property equal to 1. • user_specified — The instance is internal scan only because the specified_internal_scan_only attribute as set to true on this instance or the instance inherited the true value from an instance in its ancestry. The values are reported only when is_internal_scan_only is true.
is_modified	Boolean that specifies that a module read in from a file was edited during the insertion mode. In -rtl mode, this attribute is set as soon as the module is modified and remains set until the insertion state is left. In -no_rtl mode, the attribute is set once when the module is edited and never reset. See the get_modules command description for usage examples.
is_physical_module	Boolean that specifies that the module is a physical block. It gets set during design elaboration if the module is found in an open TSDB and its level was defined as chip or physical_block when its TSDB was created.
is_power_always_on_cell	Boolean that is set to true when the module is specified as an always_on cell when power data is read using the read_upf or read_cpf commands. It is populated during the circuit flattening process after reading the UPF or CPF. The flattening can be done using create_flat_model or by transitioning from setup to analysis mode after issuing the check_drc_rules command. By default, this attribute is false.
is_power_isolation_cell	Boolean that is set to true when the module is specified as an isolation cell when power data is read using the read_upf or read_cpf commands. It is populated during the circuit flattening process after reading the UPF or CPF. The flattening can be done using create_flat_model or by transitioning from setup to analysis mode after issuing the check_drc_rules command. By default, this attribute is false.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_power_level_shifter_cell	Boolean that is set to true when the module is specified as a level shifter cell when power data is read using the read_upf or read_cpf commands. It is populated during the circuit flattening process after reading the UPF or CPF. The flattening can be done using create_flat_model or by transitioning from setup to analysis mode after issuing the check_drc_rules command. By default, this attribute is false.
is_power_retention_cell	Boolean that is set to true when the module is specified as a retention cell when power data is read using the read_upf or read_cpf commands. It is populated during the circuit flattening process after reading the UPF or CPF. The flattening can be done using create_flat_model or by transitioning from setup to analysis mode after issuing the check_drc_rules command. By default, this attribute is false.
is_sequential_cell	Boolean that specifies that the module has at least one _dff or _dlat primitive inside at any level of hierarchy. This attribute only applies to library cells specified by the read_cell_library command.
is_synthesized	Boolean that specifies that the given module was synthesized when going to analysis mode by the quick synthesis engine.
is_unsaved	Boolean that specifies that a module that was either modified or created within the insertion mode of the dft context was not yet written out. This attribute is set as soon as the module is modified or created and is reset as soon as the module is saved using the write_design command. See the get_modules command description for examples of its usage.
is_valid	Boolean that specifies whether the object still exists. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
language	String that specifies the supported library language. Set to “tessent_lib” for Tessent library models. Set to “verilog” for Verilog modules and Verilog primitives. Set to “vhdl” for VHDL entities.
library_name	String that specifies the logic name of the library containing the module definition. All commands that accept a module name look for the module in the default library as specified by the set_logical_design_libraries command. You can use the get_modules command to access modules from different libraries. Libraries are only supported in -rtl sub-contexts. In -no_rtl contexts, one library exists and it is named “work”.
line_number	Integer that specifies a line number in the file where a given module is defined. This number is 0 for created modules and for modules without a definition.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

master_name	String that specifies the real name of the module as it was defined independently of the parameterized view.
master_object_id	An attribute that returns the object_id of the master module associated with this parameterized module. If this module is not a parameterized view, it returns the object_id of this module.
name	String that specifies the name of the module. Modules are defined from the following sources: • read_verilog files • read_vhdl files • create_design and copy_design commands • read_cell_library files • primitives have built-in names For parameterized modules, a unique name is created for each parameterized view by appending to the master_name the string @PV# where # is an integer starting from 1 and incremented for each parameterized view. Use the get_module_parameter_list and get_module_parameter_value commands to introspect the parameter values.
no_control_point	Boolean or literal that prohibits the insertion of a control point on any pin, port, or instance in the specified module when the attribute value is true. Possible values are true, false, and unspecified. The default value is unspecified, which means that the location can have control points unless the attribute is true on one of its parent modules or instances.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

no_control_reason	String that specifies why no control point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified. The no control point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode. The possible values for the tool identified locations are: • cell_internal - all the gate pins inside the library cells. • clock_cone - including set and reset line. • dft_signal — Including all the gates in the forward and backward path of any dft_signals such as observe_test_point_en, control_test_point_en, mcp_bounding_en, edt_update, ltest_en. • pin_with_constant_value - including all the gates that have constant value during capture. This can be determined either by the add_input_constraints command or test procedure. • power_cell_pin - input pins of power cells (isolation cell, level shifter) • scan_path - including sub_chain, shift_register and any logic between scan in and scan out. • tessent_ip_instance — Including all the gates and pins of Tessent instrument instances such as OCC, LBIST, TK. • tristate_enable - including tristate logic and bidirectional gates. The possible values for the user-specified locations are: • user_specified - can be specified using the “add_notest_points” command, or the “set_attribute_value” command by tagging the appropriate no_control_point attribute. • cross_clock_domain_path - can be specified using “set_test_point_analysis_option -exclude_cross_domain_paths” command. • xbounded_pin - can be specified using “set_test_point_analysis_options -allow_in_xbounding_regions” command or “analyze_xbounding” command in dft -scan context. • multicycle_path - can be specified by reading SDC file . • false_path - can be specified by reading SDC file. • excluded_clock_domain - can be specified using “set_test_point_analysis_option -exclude_clock_domains “ command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

no_observe_point	Boolean or literal that prohibits the insertion of an observe point on any pin, port, or instance in the specified module when the attribute value is true. Possible values are true, false, and unspecified. The default value is unspecified, which means that the location can have control points unless the attribute is true on one of its parent modules or instances. **Note:** The tool does not consider the no_observe_point attribute while analyzing the clock network to identify the location to drive the clock port on an inserted memory element, such as the flop associated with an observe point.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

no_observe_reason	String that specifies why no observe point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified. The no observe point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode. The possible values for the tool identified locations are: • cell_internal — All the gate pins inside the library cells. • clock_cone — Including set and reset line. • dft_signal — Including all the gates in the forward and backward path of any dft_signals such as observe_test_point_en, control_test_point_en, mcp_bounding_en, edt_update, ltest_en. • pin_with_constant_value — Including all the gates that have constant value during capture. This can be determined either by the add_input_constraints command or test procedure. • power_cell_pin — Output pins of power cells (isolation cell, level shifter) • scan_path — Including sub_chain, shift_register and any logic between scan in and scan out. • tessent_ip_instance — Including all the gates and pins of Tessent instrument instances such as OCC, LBIST, TK. • tristate_enable — Including tristate logic and bidirectional gates. The possible values for the user-specified locations are: • user_specified — Can be specified using the “add_notest_points” command, or the “set_attribute_value” command by tagging the appropriate no_observe_point attribute. • cross_clock_domain_path — Can be specified using “set_test_point_analysis_option -exclude_cross_domain_paths” command. • xbounded_pin — Can be specified using “set_test_point_analysis_options -allow_in_xbounding_regions” command or “analyze_xbounding” command in dft -scan context. • multicycle_path — Can be specified by reading SDC file. • false_path — Can be specified by reading SDC file. • excluded_clock_domain — Can be specified using “set_test_point_analysis_option -exclude_clock_domains” command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

object_id	The object_id attribute is a unique id for the object that remains valid while the design is in memory. This attribute is useful to identify the object in a Tcl dictionary. The object_id is only valid within one tool invocation. Once you terminate the tool or delete the design, the object_id is no longer valid. The object_id has the form "module: *integer*"
object_type	Returns the object data type “module”.
power_domain_island	Returns the name of the power domain island associated with the current design. This attribute is null for all other modules. This attribute is populated when going from setup to analysis system mode if a UPF or CPF file has been read in. Its value is computed using the design hierarchy and the power_domain_name attribute. For a definition of Power Domain Island, refer to the *Tessent Glossary*.
power_domain_name	Returns the name of the power domain associated with the current design. This attribute is null for all other modules and is populated when you read in a UPF or CPF file. For a definition of Power Domain, refer to the *Tessent Glossary*.
preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, the tool creates an inferred buffer on all pins on all instances of the module in the flat model to preserve the separation of nets inside and outside the module. Any attribute stored on the hierarchical pin is visible in the flat model. If you set this attribute on a pin of the design, the flat model must be reconstructed. The attribute is not lost during flattening.
prohibited	String that specifies that the tool cannot learn and set the cell_type automatically. It effectively prevents the test insertion tools from inserting the model as a cell. For details see *Model-Level Attribute Descriptions in the Tessent Cell Library Manual*.
simulation_function	String that specifies the function as learned by simulation. Refer to “Model-Level Attribute Descriptions” in the *Tessent Cell Library Manual* for complete information.
specified_internal_scan_only	An attribute used to specify that the instances of the module are forced to remain internal during wrapper analysis. Dedicated isolation wrapper cells are inferred to isolate the input and output pins of the instances if any interact with the primary input and output ports.
sync_cell_length	For posedge and negedge synchronizer cells, this attribute is an integer that specifies the number of DFFs in the cell. For all other cell types, this attribute is an empty string.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

synthesize_before_analysis	Boolean that specifies that the given module is to be synthesized when going to analysis mode. During ICL extraction, the combinational path between the IJTAG port of the modules having a corresponding ICL module is traced and any module encountered along the way is synthesized. If you use a test_setup procedure during ICL extraction, you must set this attribute on all RTL modules which must be synthesized in order for the test setup simulation to propagate its value to the entire ICL network as the pretracer does not traverse sequential cells. The quick synthesis engine uses a cell library defined in the build tree at the given location "$TESSENT_HOME/lnx-x86/lib64/hdleng/lib". Both Tessent and Verilog cell library models are supplied, namely "dft_sim.atpg" and "dft_sim.v". The tessent libraries are automatically loaded when synthesis is used.
tcd_bscan_name	A string attribute that gets set on every module for which there is a matched TCD BoundaryScan description (see "Tessent Core Description" on page 4793). See "set_design_sources -format tcd_bscan" and set_module_matching_options commands for more information about this matching process.
tcd_fusebox_name	A string attribute that gets set on every module for which there is a matched TCD FuseBoxInterface description (see "Tessent Core Description" on page 4793). See "set_design_sources -format tcd_memory" and set_module_matching_options commands for more information about this matching process.
tcd_memory_cluster_name	A string attribute that gets set on every module for which there is a matched TCD MemoryCluster description (see "Tessent Core Description" on page 4793). See "set_design_sources -format tcd_memory_cluster" and set_module_matching_options commands for more information about this matching process.
tcd_memory_name	A string attribute that gets set on every module for which there is a matched TCD Memory description (see "Tessent Core Description" on page 4793). This attribute is also set on memory modules that are *inside* another memory. See "set_design_sources -format tcd_memory" and set_module_matching_options commands for more information about this matching process.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

tcd_memory_cells_info	A string attribute describing a Tcl dictionary that gets set on every module for which there is a matched TCD Memory description and is identified as a wrapper that contains exactly one memory cell instance. The Tcl dictionary format for the attribute is as follows:
	`module_name {`
	`  Instances {instance_path}`
	`}`
	Where module_name is the wrapper module name and instance_path is the memory cell instance path relative to the wrapper module.

type	Set to “cell” for the following modules: • Models read in with read_cell_library. • Verilog modules surrounded by the `celldefine/ `endcelldefine directives and read in with read_verilog. Set to “primitive” for the following modules: • Verilog primitives used in the design. Set to “design” for all other modules.
vhdl_architecture	String that reports the architecture of vhdl modules. It is null for modules of other languages. All commands that accept a module name return the vhdl entity associated with the default architecture which is the last compiled version. Use get_modules to access modules with the non default architecture.

Inheritance

Built-in inherited attributes:

- None

User-defined attributes are inherited. See “Attribute Inheritance Behavior” on page 3840.

Module Specification

A module object (module_object) is specified by a module name which is a string. For example: “modA”.

Related Topics

get_attribute_list

get_attribute_value_list

get_common_parent_instance

get_modules

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_name_list

get_ports

report_attributes

Instance

An instance is a single instantiation of a module. No instance object exists until you run the set_current_design command.

The instance object type interacts with attributes as follows:

- Module attributes are automatically visible as read-only attributes on corresponding instances if the instance object type does not have the same attribute registered.
- Attributes defined with the applies_to_child_instances option set to On automatically become visible to all child instances of the instance object.

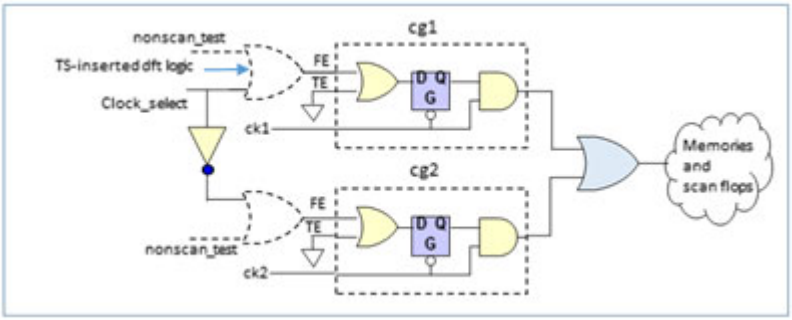
Built-In Attributes

active_child_scan_mode	When a child core exposes scan chain segments from several scan modes, this attribute makes the existing segments from a specific child scan mode active. During DRC, these active segments are the ones that get traced if tracing is enabled. Scan segments of the “active” child scan mode get the “state” attribute “usable”, while segments from other child scan modes are attributed with the “state” “unusable_child_mode”. Accordingly, during analysis, when creating scan chain families and/or adding scan modes, this attribute is also used to control their populations. If the tool cannot uniquely resolve the active scan mode for an instance by using the rules below, then the “state” attribute is set to “unresolved_child_mode”. The tool determines the effective “active_child_scan_mode" attribute for scan segments of a multi-mode instance using the following rule sequence: 1. The mode corresponding to the “active_child_scan_mode” if it is set for the instance and matches one of its child modes. 2. The mode of type “external” if “active_child_scan_mode” is not set for the instance and a unique child mode of type “external” exists. 3. The mode of type “unwrapped” if “active_child_scan_mode” is not set for the instance and a unique child mode of type “unwrapped” exists.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

bisr_power_domain_name	An attribute that can be set on memory instance to associate a BISR power domain name to a memory. You typically do not need to use this attribute when you read in a UPF or CPF file as the power_domain_name attribute is used to infer the power domain of the memories. You use the bisr_power_domain_name attribute if you want to have more BISR power than those described in the UPF/CPF file. You use this attribute on Pins to map specific BISR chains on subblock to power domain at the current level as documented in the bisr_segment_order_file section.
created_by_synthesis	Boolean that indicates that the instance was created by quick synthesis. When an instance has this attribute set to true, it does not exist in the original RTL and cannot be used for design edits. See the set_quick_synthesis_options command.
def_orientation	Contains the DEF orientation of the flip-flop, memory, or sub-physical region extracted by the read_def command. The attribute can have the following values: N, S, E, W, FN, FS, FW, FN.
def_x_coordinate	The x coordinate of an instance extracted from a specified def file when you execute the read_def command.
def_y_coordinate	The y coordinate of an instance extracted from a specified def file when you execute the read_def command.
design_hierarchy_level	Integer value that indicates the depth in the design where the instance exists. For example, instances in the current design module have a value of 1. For i1/i2/i3, instance i3 has a value of 3. You can use this attribute with the sort_collection command to sort the instance according to the topology. This can be useful for design manipulation with auto uniquification.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dont_touch_enables_of_clock_gater	Boolean attribute that disables the automatic insertion of DFT logic in the fan-in of both the “func_enable” and “test_enable” pins of a functional clock gating cell during process_dft_spec (PDS). By default, the DFT_C1 and DFT_C6 pre-DFT rules assume the presence of this logic and force these pins active during the rule check. You must disable this insertion in a design with different clock gater cells fanning out to a common point through a multi-input gate, such as the clock multiplexer circuit example below.  By default, if the Clock_select signal is forced high by the test setup procedure, DFT_C1 would infer that both FE pins are forced high during memory BIST, enabling both ck1 and ck2 clocks to pass through and causing a conflict at the OR gate output. The same is true for DFT_C6 and the TE pins during logictest shift mode. The following command removes these violations by inhibiting the addition of DFT logic before the cg2 instance, leaving only ck1 to reach the OR gate in both DRCs and test: `set_attribute_value cgc1 -name dont_touch_enables_of_clock_gater -value on` This attribute features the -applies_to_child_instances option, thus propagates downward to any child instance having a func_en or test_en pin. It is visible as an instance attribute in Tessent Visualizer, replaces the obsolete exclude_from_dft_gbl_fe_forces attribute, and extends its scope to the “test_enable” clock gater input.
ignore_during_boundary_scan_extraction	When set to true, the tool ignores the instance for the purposes of the BSCAN3 design rule check.
ignore_during_icl_extraction	When set to true, even if the instance has an associated ICL module during ICL extraction, the instance is ignored and not reported in the extracted ICL. This attribute is useful to allow extracting the ICL for the portion of the network that is correct while other instances still have issues to resolve. The generated ICL module has the tessent_ignored_design_instances attribute defined referencing all design instances that were ignored.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

in_graybox	A read-only attribute that is valid after the analyze_graybox command has been run. You can query this attribute to check if a specific instance is included in the graybox. For more information about this attribute, refer to the analyze_graybox command description.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

in_graybox_reason	A read_only attribute that is only valid after you use the "analyze_graybox -scan | -ijtag" command. You can query this attribute on any instance for which the in_graybox attribute returns true. The in_graybox_reason return value indicates the reason why the tool included the instance in the graybox. Some reasons listed below only apply to the scan graybox analysis and not for the ijtag graybox analysis. The attribute can have the following values: • assert_shift_path — If a wrapper cell was implemented using a cell with pins that are marked with an assert_shift value in the Tessent cell library, and this wrapper cell must be included in the graybox, the tool must preserve the logic in the fan-in of the assert_shift pin of the library cell to avoid failing asserts when using the graybox.This return value indicates the instance was in the fan-in of the assert_shift pin. This value only appears when you use "analyze_graybox -scan" and not when you use "analyze_graybox -ijtag". • complete_preserve — This return value identifies the library cells that were either directly specified for complete_preserve using the preserve_in_graybox attribute or by the "analyze_graybox -preserve_instance" switch. • complete_preserve_clock_path — The memory elements inside the complete_preserve instances must be functional, so the tool must mark all the logic that provides control of the clock and the set and reset pins. This return value indicates that the tool added the logic during the backtrace of the clock, set, and reset pins of a flop inside a complete_preserve instance. • fanin_of_complete_preserve — This return value identifies logic that is between complete_preserve instances identified while tracing back from the fan-in of a complete_preserve instance, and marking logic that is also in the fanout of another preserve instance. This search is currently limited to combinational logic paths. • Primary_outputs — This return value identifies any logic identified by tracing back from primary output pins and looking for either scan_cells (wrapper cells) or complete_preserve instances. • primary_outputs_clock_path — There may be non-scan flops encountered when tracing back from the primary outputs. The clock, set, and reset pins of these flops must also remain controllable. This return value identifies logic added to preserve controllability of the clock paths for these flops

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

in_graybox_reason (continued)	• scan_path_logic — This return value identifies the library cells that are necessary to make the scan chains traceable, which means all the library cells to preserve the path from the input of the scan chain to the output. This also includes logic driving the scan_enable because that is necessary to enable the scan path. This value only appears when you use “analyze_graybox -scan” and not when you use “analyze_graybox -ijtag”. • stem_preservation — To properly compare fault lists between the graybox netlist and the complete netlist, any net that originally consists of multiple fanouts must also have multiple fanouts in the graybox. The tool accomplishes this by adding an extra library cell into the graybox to avoid fault collapsing during ATPG. This return value indicates that the tool added the instance to preserve multiple-fanout for one of the nets that drives it. • stem_preservation_clock_path — During the marking of extra logic that is needed for stem_preservation, the tool always tries to use combinational logic gates. However, sometimes there is no other option, and the tool must include another non-scan cell. This return value indicates that the clock, set, and reset pins for these flops must remain controllable and identifies logic that was added to maintain controllability of the clock, set, or reset pin of one of these flops. • wrapper_cells — This return value identifies any logic identified by tracing back from the scan cell data_in of the scan chains. This value only appears when you use “analyze_graybox -scan” and not when you use “analyze_graybox -ijtag”. • wrapper_cell_scan_enable_clock_path — During the marking of the scan_path_logic, the tool also preserves the logic that feeds the scan_enable. This back-trace may encounter and preserve non-scan cells. This return value indicates that the clock, set, and reset pins for these flops remain controllable. This value only appears when you use “analyze_graybox -scan” and not when you use “analyze_graybox -ijtag”. • wrapper_feedback_clock_path — There may be non-scan flops encountered when searching back from the scan-path cells. This return value indicates that the clock, set, and reset pins of these flops must also remain controllable. This value only appears when you use “analyze_graybox -scan” and not when you use “analyze_graybox -ijtag”.
instrument_name	A string attribute that is set on instances of modules for which there is a matched ICL instruments description. If an ICL instruments instance is instantiated below another ICL instruments instance, the attribute is only set on the higher instance. See set_design_sources -format icl and set_module_matching_options commands for more information about this matching process.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_blackbox	Boolean that specifies that the instance was black boxed with add_black_boxes -instance or that the module of the instance was black boxed as described for the is_blackbox attribute of the module object type. The default is false.
is_internal_scan_only	Boolean that specifies that the instance is internal scan only.
is_internal_scan_only_reason	String that specifies the reason the instance is internal scan only as follows: • imported_from_tcd_scan — The object was inferred to be internal scan only because it has a Core/Scan TCD description and the property internal_scan_only is equal to 1. • user_specified — The instance is internal scan only because the specified_internal_scan_only attribute as set to true on this instance or the instance inherited the true value from an instance in its ancestry. • user_specified_from_net — The instance is internal scan only because the specified_internal_scan_only attribute was set on a net which happens to be an RTL register. The instances happen to be the flop-flop instance that the tool created during quick synthesis for that RTL register. The values are reported only when is_internal_scan_only is true.
is_non_editable	Boolean that specifies that the instance is non-editable.
is_non_editable_reason	A string that specifies why the instance is non-editable as follows: • below_hard_module — The instance is below an instance which has is_hard_module equal to 1 • hard_module — The instance has is_hard_module equal to 1 • created_by_quick_synthesis — The instance was created by quick synthesis • matched_icl_module — The module of the instance has a matched ICL module • below_matched_icl_module — The instance is below a module that has a matched ICL module
is_non_scannable	Boolean that specifies that the instance is non-scannable.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_non_scannable_reason	A string that specifies why the instance is non-scannable as follows: • s_rule_violation — The library cell has an unfixable s-rule violation. • non_scan_path_cell_in_hard_module — This attribute value is used for library cells that are inside a hard module that has only traced scan segments, and are not part of any traced segment. • clock_gater_cell — By default, memory elements inside a clock gater are not considered as scannable cells. However, you can still define a scan segment if the clock gater was implemented as a scannable cell. • add_nonscan_instance_control — The instance was marked as non-scannable as a result of using the “add_nonscan_instances -control” command in analysis mode. • ignored_slice_due_to_segment_definition — You have defined a scan segment for one or more slices (scan paths) in the multi-slice library cell, but not for all of them. In this case, the tool uses the segment definitions and ignores the slices (scan paths) that do not have a corresponding segment definition. • driven_scan_enable — The scan_enable is connected to something that was not defined as a scan enable. • shadow_cell — This memory element was identified to be a shadow cell. It is loaded with a known value during scan shifting so there is no reason to convert it into a scan cell. Also, it only captures the value from the scan cell, so observability is also not an issue. • transparent_latch — This cell is a latch (or transparent latch) which makes it non-scannable. • black_box — The memory element is a black box or is inside a black box. • inferred_during_system_mode_transition — This is used to mark cells that remain non-scannable due to previously unspecified conditions. It occurs very infrequently. • imported_from_tcd_scan — The object was inferred to be non-scannable because it has a Core/Scan TCD description and the property drc_from_boundary_only is equal to 1. • imported_from_icl — The instance was inferred to be non-scannable because it has a matched ICL description and the attribute keep_active_during_scan_test is not specified as false. • imported_from_bscan — The instance was inferred to be non-scannable because it is part of the boundary scan chain. • missing_scan_model — The instance was inferred to be non-scannable because it does not have a scan flop equivalent mapping. This only happens in the dft -scan context. • user_specified — The object was specified to be non-scannable using the add_nonscan_instances command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_non_scannable_reason (continued)	• imported_from_mbisr — The instance was inferred to be non-scannable because it is part of the memory bist controller. • multi_slice_cell_needs_mapping — The tool does not support mapping of multi-slice non-scan cells to scan cells. • segment_inside_segment_or_chain — This scan segment is part of a larger segment or chain so it is not directly usable. Gate pin-specific return values: • non_scan_path_element — This reason only applies to gate-pins and is used to identify the memory elements (DFFs) that are not part of the scan path (and are thus not directly controllable and observable). • disabled_segment — This reason only applies to gate-pins and is used to identify memory elements within a traced segment that fails DRC (for one of the previously mentioned reasons).
is_power_always_on_cell	Boolean that is set to true when the instance is specified as an always_on cell when power data is read using the read_upf or read_cpf commands. It is populated during the circuit flattening process after reading the UPF or CPF. The flattening can be done using create_flat_model or by transitioning from setup to analysis mode after issuing the check_drc_rules command. By default, this attribute is false.
is_power_isolation_cell	Boolean that is set to true when the instance is specified as an isolation cell when power data is read using the read_upf or read_cpf commands. It is populated during the circuit flattening process after reading the UPF or CPF. The flattening can be done using create_flat_model or by transitioning from setup to analysis mode after issuing the check_drc_rules command. By default, this attribute is false.
is_power_level_shifter_cell	Boolean that is set to true when the instance is specified as a level shifter cell when power data is read using the read_upf or read_cpf commands. It is populated during the circuit flattening process after reading the UPF or CPF. The flattening can be done using create_flat_model or by transitioning from setup to analysis mode after issuing the check_drc_rules command. By default, this attribute is false.
is_power_retention_cell	Boolean that is set to true when the instance is specified as a retention cell when power data is read using the read_upf or read_cpf commands. It is populated during the circuit flattening process after reading the UPF or CPF. The flattening can be done using create_flat_model or by transitioning from setup to analysis mode after issuing the check_drc_rules command. By default, this attribute is false.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_valid	Boolean that specifies whether the object still exists. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_block_id	The leaf_name of instances located inside generate blocks have the generate block ids joined by periods prepended to their instantiation name. This attribute returns the block_id part of the leaf_name. It is null for instances that are not inside generate blocks.
leaf_name	String that specifies the name of the instance (value of the name attribute) without the hierarchical path. After leaving insertion mode in the tool, this attribute shows the value of the unrolled_leaf_name attribute.
leaf_name_hash	This attribute is normally identical to the leaf_name attribute except when the leaf instance is located inside a generate loop construct inside an RTL file. The leaf_name includes the block id with the loop index such as in loopA[4].u2. The leaf_name_hash attribute is the same except that the loop index is replaced by a hash sign as shown here: loopA[#].u2. You use this attribute if you are editing the content of generate loops and you have not enabled auto uniquification. For the delete_connections and move_connections commands, you must use a lookup table with *parent_module.leaf_name_hash.pin_leaf_name* entries to know if you have already processed that pin. See the examples in the move_connections and get_dft_cell command descriptions for examples of such usage. Note that the value of this attribute always refer to the pre_synthesis_name just like the pre_synthesis_leaf_name attribute.
master_module_name	String that specifies the value of the master_name on the module object associated with the instance.
module_name	String that specifies the value of the name of the module object associated with the instance.
name	String that specifies the hierarchical name of the instance relative to the current design. The name of every instance is unique. After leaving insertion mode in the tool, this attribute shows the value of the unrolled_name attribute.
no_control_point	Boolean or literal that prohibits the insertion of a control point on any pin or instance in the specified instance when the attribute value is true. Possible values are true, false, and unspecified. The default value is unspecified, which means that the location can have control points unless the attribute is true on one of its parent instances.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

no_control_reason	String that specifies why no control point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified. The no control point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode. For possible values for the tool identified locations and the user-specified locations, refer to no_control_reason in the Module attribute section.
no_observe_point	Boolean or literal that prohibits the insertion of an observe point on any pin or instance in the specified instance when the attribute value is true. Possible values are true, false, and unspecified. The default value is unspecified, which means that the location can have observe points unless the attribute is true on one of its parent instances. **Note:** The tool does not consider the no_observe_point attribute while analyzing the clock network to identify the location to drive the clock port on an inserted memory element, such as the flop associated with an observe point.
no_observe_reason	String that specifies why no observe point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified. The no observe point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode. For possible values for the tool identified locations and the user-specified locations, refer to no_observe_reason in the Module attribute section.
object_id	The object_id attribute is a unique id for the object that remains valid while the design is in memory. This attribute is useful to identify the object in a Tcl dictionary. The object_id is only valid within one tool invocation. Once you terminate the tool or delete the design, the object_id is no longer valid. The object_id has the form "instance: *integer*"
object_type	Returns the object data type “instance”.
parent_instance	String that specifies the instance name in which the instance exists. This attribute automatically handles the escaped identifier and returns the instance path of the instance without the /<leaf_name> part of the path.
post_synthesis_name	The computed equivalent post synthesis name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

power_domain_island	Returns the name of the power domain island associated to the instance. This attribute is populated when going from setup to analysis system mode if a UPF or CPF file has been read in. Its value is computed using the design hierarchy and the power_domain_name attribute. For a definition of Power Domain Island, refer to the *Tessent Glossary*.
power_domain_name	Returns the name of the power domain associated with the instance. This attribute is populated when you read in a UPF or CPF file. For a definition of Power Domain, refer to the *Tessent Glossary*.
pre_synthesis_leaf_name	String that specifies the original RTL leaf name of the object even when the quick synthesis view of the design is currently in memory. When the quick synthesis view of the design is not in memory, then the value of this attribute equals the value of the leaf_name attribute. If the object is created by the synthesis (created_by_synthesis attribute is true which means the object did not exist before the synthesis), then the string is empty. The pre_synthesis_leaf_name is different from the leaf_name only when the instance is inside a generate block that the tool synthesized during quick synthesis.
pre_synthesis_leaf_block_id	String that shows which generated block instantiation is the current focus of the tool. This attribute is used to detect and compute instances that are from generated blocks.
pre_synthesis_leaf_name_hash	String that shows where the parameter of the generated block is located. This attribute is used to detect and compute instances that are from generated blocks.
pre_synthesis_name	String that specifies the original RTL name of the object even when the quick synthesis view of the design is currently in memory. When the quick synthesis view of the design is not in memory, then the value of this attribute equals the value of the name attribute. If the object is created by the synthesis (created_by_synthesis attribute is true which means the object did not exist before the synthesis), then the string is empty. The pre_synthesis_name is different from the name only when the instance has one or more generate blocks in its ancestry that the tool synthesized during quick synthesis
preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, the tool creates inferred buffers on all pins on the instance to preserve the separation of nets inside and outside the instance. Any attribute stored on the hierarchical pin is visible in the flat model. If you set this attribute on a pin of the design, the flat model must be reconstructed. The attribute is not lost during flattening.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

preserve_in_graybox	Boolean attribute that you can set to true prior to issuing the analyze_graybox command to force the specified instance to be included in the graybox. This behavior is the same as using the -preserve_instances switch with the analyze_graybox command.
specified_internal_scan_only	An attribute used to specify that the instance is to be forced to remain internal during wrapper analysis. Dedicated isolation wrapper cells are inferred to isolate the input and output pins of the instances if any interact with the primary input and output ports. When the attribute is not specified on an instance, it inherits its value from its associated module when the attribute is specified on the module. If it is not specified on the module, it inherits its value from its parent instance.
specified_wrapper_type	An attribute that forces the wrapper type of a flop or clock gater instance. Available values are: off, input, output, and generic. The default is off. When set or left to default to off, the wrapper_type value of each flop and clock gater instance is computed by the analyze_wrapper_cells command. Special treatment is performed on instances of OCCs. See the inferred_from_clock_source value in the shared_wrapper_cell_reason attribute description. When you set the specified_wrapper_type attribute to input, output, or generic, the specification is kept during the analysis. If you used the input value, the analyze_wrapper_cells command automatically finds all flip-flops fanning out from the instance you forced to input type and infers them as output wrapper type. The result of the analyze_wrapper_cells command, which combines the specified_wrapper_type values with the inferred values, are reflected in the wrapper_type and shared_wrapper_cell_reason attributes described below. If you set the specified_wrapper_type attribute after wrapper analysis has been performed, the tool issues an error. Switching back to setup mode resets wrapper analysis and enables you to set the specified_wrapper_type attribute again.
tcd_bscan_name	A string attribute that is set on instances of modules for which there is a matched TCD BoundaryScan description (see "Tessent Core Description" on page 4793). If a tcd BoundaryScan instance is instantiated below another tcd BoundaryScan instance, the attribute is only set on the higher instance. See set_design_sources -format tcd_bscan and set_module_matching_options commands for more information about this matching process.
tcd_edt_name	A string attribute that is set to the name of a core instance when the EDT instrument core instance is associated with a design instance. This ensures boundary preservation during flattening.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

tcd_fusebox_name	A string attribute that is set on instances of modules for which there is a matched TCD FuseBoxInterface description (see "Tessent Core Description" on page 4793). If a tcd FuseBoxInterface instance is instantiated below another tcd FuseBoxInterface instance, the attribute is only set on the higher instance. See set_design_sources -format tcd_fusebox and set_module_matching_options commands for more information about this matching process.
tcd_lpct_name	A string attribute that is set to the name of a core instance when the LPCT core instance is associated with a design instance. This ensures boundary preservation during flattening.
tcd_memory_cluster_name	A string attribute that is set on instances of modules for which there is a matched TCD MemoryCluster description (see "Tessent Core Description" on page 4793). If a TCD Memory or MemoryCluster instance is instantiated below another TCD MemoryCluster instance, the attribute is only set on the higher instance. See set_design_sources -format tcd_memory_cluster and set_module_matching_options commands for more information about this matching process.
tcd_memory_name	A string attribute that is set on instances of modules for which there is a matched TCD Memory description (see "Tessent Core Description" on page 4793). If a TCD Memory instance is instantiated below another TCD Memory instance, the attribute is only set on the higher instance. See set_design_sources -format tcd_memory and set_module_matching_options commands for more information about this matching process.

tcd_memory_cells_info	A string attribute describing a Tcl dictionary that gets set on every module for which there is a matched TCD Memory description and is identified as a wrapper that contains exactly one memory cell instance. The Tcl dictionary format for the attribute is as follows:
	`module_name {`
	`  Instances {instance_path}`
	`}`
	Where module_name is the wrapper module name and instance_path is the memory cell instance path relative to the wrapper module.

tcd_occ_name	A string attribute that is set to the name of a core instance when the OCC core instance is associated with a design instance. This ensures boundary preservation during flattening.
unrolled_leaf_name	An attribute that indicates whether the tool performed loop unrolling. In insertion mode and if the tool unrolled loops, then this attribute shows the name the object has when written with the write_design command. When there is no loop or the loop is not unrolled, this attribute is the same value as the leaf_name attribute.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

unrolled_name	An attribute that indicates whether the tool performed loop unrolling. In insertion mode and if the tool unrolled loops, then this attribute shows the name the object has when written with the write_design command. When there is no loop or the loop is not unrolled, this attribute is the same value as the name attribute.
wrapper_type	An attribute used to reflect the resulting wrapper type of flop and clock gater instances based on the specified_wrapper_type attribute value you set, special processing of the OCC modules, and the result of the analysis after the analyze_wrapper_cells command is run. See the specified_wrapper_type attribute description above for more information. The instances of type input have their scan enable held on during the input test modes. The cells of type output have their scan enable held on during the output test modes. The cells of type generic do not have their scan enable held on during any mode type. The allowed values of this attribute are input, output, generic, or an empty string.

Inheritance

Built-in attribute inheritance from associated module objects:

- cell_type
- has_no_definition
- ijtag_is_icl_module
- is_created
- is_hard_module
- is_incomplete
- is_modified
- is_synthesized
- is_unsaved
- language
- library_name
- preserve_boundary
- type
- vhdl_architecture

User-defined attributes are inherited. See “Attribute Inheritance Behavior” on page 3840.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Instance Specification

An instance object (instance_object) is specified by a complete hierarchical instance pathname. For example: “/u1/u2/u3”.

Related Topics

get_attribute_list

get_attribute_value_list

get_common_parent_instance

get_name_list

report_attributes

Port

A port is an object on a module and can have an input, output or inout direction. Ports have a simple non-hierarchical name.

One port object exists for each bit of a root bundle port so every bit of a root bundle port can be attributed independently.

The port object type interacts with attributes as follows:

- Attributes defined on ports are automatically visible as read-only attributes on the associated pin objects unless the attribute is also registered for the pin object type. This is the case for the name attribute which is defined for both port and pin object types. These attributes are stored on the associated port of the pins and not on the pin object itself.

Built-In Attributes

See “Bundle Examples” on page 3742 for an example used to describe these attributes.

active_clock_edge	A string attribute that reflects the value of the gate_pin attribute having the same name when the gate_pin maps to a net object. See the description of the Gate_pin attribute description for the meaning of its value.
active_polarity	A string attribute that reflects the value of the gate_pin attribute having the same name when the gate_pin maps to a net object. See the description of the Gate_pin attribute description for the meaning of its value.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

assert_capture	A string attribute that reflects the assert_capture value parsed by the read_cell_library command for the introspected port. The values returned can be {} (if none), “0”, “1”, “Z”, or “stable”. See the assert_capture attribute description for the meaning of its values.
assert_pattern	A string attribute that reflects the assert_pattern value parsed by the read_cell_library command for the introspected port. The values returned can be {} (if none), “0”, “1”, “Z”, or “stable”. See the assert_pattern attribute description for the meaning of its values.
assert_shift	A string attribute that reflects the assert_shift value parsed by the read_cell_library command for the introspected port. The values returned can be {} (if none), “0”, “1”, “Z”, or “stable”. See the assert_shift attribute description for the meaning of its values.
associated_scan_interface	String attribute used to associate the following BISR data input ICL ports with a ScanInterface wrapper: • bisr_reset • bisr_select • bisr_mem_disable • parallel_in • serial_repair_enable Because DataInPorts cannot be specified inside the ScanInterface wrapper, the associated_scan_interface and bisr_function attributes enable the memory_bisr plugin to identify all BISR chain ports associated to each power domain. The BISR scan_in, scan_out, clock, and shift_en ICL ports can be extracted from the ScanInterface wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

base_class	String that represents the class category of the base type and completes the base_type definition. Possible values are as follows:4_state_signed | 4_state_unsigned | 2_state_signed | 2_state_unsigned | enum | struct_packed | struct_unpacked | union_packed | union_unpacked | interface | interface_modport | other. See "base_class and base_type Attributes" in the *Tessent Shell User's Manual* for examples of base_class and base_type.
base_type	String that represents the base type of the subtype of the bundle. The base_type is a key or user name that helps in understanding the base type of the bundle. In the case of an array, the base type is the type of the array element. For example: The base_type of "data_in[2][3].b" is "logic". The base_type of "data_in[2][3]" is "struct_unpacked". The base_type of "data_in[2]" is "struct_unpacked".
bisr_function	String that reflects the function of the design port for Tessent Memory Repair. Legal values for this attribute are: Reset, MemoryDisable, Select.
bus_left_index	Integer that specifies the left index of the complete bus. Null for scalar ports. In the case of a leaf complex port, this attribute always returns a value of 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

bus_name	String that specifies the name of the bus without the index. <name> = bus_name [index] when is_bus is true; otherwise, bus_name is null. In the case of complex ports, this attribute returns an empty string. If the bus is one-dimensional, the attribute is a valid string. You need to use the root_bundle_name attribute for leaf objects of complex ports. The root_bundle_name attribute is equivalent to the bus_name attribute, but it works for all datatypes, from simple to complex. See the root_bundle_name attribute description in this table.
bus_range_direction	String set to “up” or “down” that reflects the bus range direction. This attribute follows the to and downto BIT_VECTOR types in VHDL. In the case of complex ports, this attribute always returns an empty string.
bus_right_index	Integer that specifies the right index of the complete bus. Null for scalar ports. In the case of a leaf complex port, this attribute always returns a value of 0.
bus_width	Integer that specifies the number of bits independent of whether left_index is smaller, larger than, or equal to right_index. In case of a leaf complex port, this attribute always returns a value of 0.
clock_domain_inversion	A string attribute that specifies whether there is an inversion between the clock domain base and the node. When 1, then the inversion exists.
clock_domain_label	A string attribute that is populated when running the DFT_C1, DFT_C2, and DFT_C3 DRC rules. It is attached to every port on which a clock was defined. The string contains the label associated to the clock defined on the port. It is also set by DFT_C6 on every node between the clock and the sequential elements.
clock_reference_destinations	String defined on a reference pin, pointing to its generated clock pin.
clock_reference_inv_destinations	String defined on an inverted reference pin, pointing to its generated clock pin.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

clock_reference_inv_source	String defined on a generated clock pin, pointing its inverted reference pin.
clock_reference_source	String defined on a generated clock pin, pointing to its reference pin.
constraint_value	String that specifies that the port's primary input was set to a constant with the add_input_constraints command. When set to C0, CT0, C1, CT1, CX, or Cz, this attribute displays 0, 0, 1, 1, x, or z, respectively. The default is null.
constraint_value_during_load_unload	String attribute available for top-level ports and pseudo-ports that specifies the value forced at the beginning of load_unload procedure. This attribute is supported in patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis and dft -scan contexts. Possible values are: 0, 1, X and Z.
control_scan_flop	Reserved for future use.
created_by_synthesis	Boolean that indicates that the port was created by synthesis. When a port has this attribute set to true, it does not exist in the original RTL and cannot be used for design edits. See the set_quick_synthesis_options command.
def_x_coordinate	The x coordinate of a top-level port extracted from a specified def file when you execute the read_def command.
def_y_coordinate	The y coordinate of a top-level port extracted from a specified def file when you execute the read_def command.
DFT_Cx_synth_stop_point	Boolean indicating a stop point during fan-in tracing from BIST'd memory cell clock pins happening during selective (partial) quick synthesis. Selective synthesis speeds up tessent shell execution by synthesizing only those design modules containing RTL constructs in the way of the clock paths to BIST'ed memory clock pins. Notice that unconditional full modules synthesis, requiring no such tracing, is instead used when the set_dft_specification_requirements command is specified with the -logic_test option.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dft_clock_enable_func_en	Boolean defined on the functional enable pin of a clock gater cell, as specified by the add_dft_clock_enables command.
dft_clock_enable_test_en	Boolean defined on the test enable pin of a clock gater cell, as specified by the add_dft_clock_enables command.
dft_clock_mux_destinations	String defined on the test input pin of a dft_clock_mux, pointing to its output pin. See the add_dft_clock_enables command.
dft_clock_mux_source	String defined on the output of a dft_clock_mux, pointing to its test input pin. See the add_dft_clock_mux command.
dft_control_async_set_reset	Boolean defined on the source of an asynchronous set or reset signal. It can either be inferred by rule DFT_C9 or explicitly set by the "add_dft_control_points -type async_set_reset" command. The process_dft_specification command gates this node with dft signal "async_set_reset_dynamic_disable".
direction	String that specifies the direction of the port. Possible values are as follows: • buffer (VHDL) • inout (Verilog and VHDL) • input (Verilog and VHDL) • linkage (VHDL) • output (Verilog and VHDL) In the case of a leaf SV Interface port, this attribute returns the appropriate direction depending on whether a modport is used or not. If a modport is not used for the port, the direction is inout.
dynamic_dft_signal	A string attribute that is set on the node where a dynamic DFT signal is defined. The value of the attribute is the name of the dynamic DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a tooltip is shown letting you know that it is the location of the dynamic DFT signal.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

function	String that either reports the function of the port on cell modules or reports the presence of the power or ground attributes on ports of the current_design. For a list of possible values, see "Function Attribute Values" on page 3700. For cell modules, this attribute is initially null and is set by the set_current_design command from information extracted from the cell library files. For information about how cell_type information can be loaded, refer to the read_cell_library command. Note that the LogicVision scang.lib, pad.lib and cell.lib format are natively supported by the read_cell_library command in case your cell library file only includes the ATPG cell modeling information. See the table in the Pin Attributes section of the *Tessent Cell Library Manual* for a complete list of available function values. For a top-level module, this attribute is initially null and you can set one of the following values on an input/inout port with the set_attribute_value command: • **power** — Constrains power input/inout ports during circuit setup and DRC. • **ground** — Constrains ground input/inout ports during circuit setup and DRC. For more information about using these values, refer to "Power and Ground Ports Exclusion from the Pattern Set" in the *Tessent Scan and ATPG User's Manual*.
gate_pin_id	Returns the id of a gate_pin when the pin has an associated gate_pin inside the flat model. Exists for cell pins associated to fault sites and for hierarchical pins that have an attribute that is set to a non-default value and whose -preserve_boundary_in_flat_model option is set to true.
has_functional_source	Returns true unless the port is of direction output and the port net is undriven or tied. In case of a leaf complex port, the tracing is also done. The tool computes this attribute correctly.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ignore_during_icl_extraction	Ports of the current design with this attribute set to true are ignored during ICL extraction. This is useful, for example, if an ICL instance has connections to primary ports as well as to other ICL instances, but the ICL shall contain only the connections between the instances. This attribute does not have any effect on ports of other modules besides the current design.
ignore_for_graybox	Set to true when an output port is to be excluded from graybox analysis. By default, this attribute is set to false for all output ports (considered by the graybox analysis backtracing). Setting this attribute on an input port has no effect on the analyze_graybox command. You need to exclude the scan_out ports of the core chains (or channel outputs if it's a design containing EDT hardware) from graybox analysis using this attribute because the tool has no way of knowing which outputs are core chain scan output ports. For more information about this attribute, refer to the analyze_graybox command description.
ijtag_c0	Boolean attribute that is set on the design ports of a module that has a corresponding ICL module, if the port name is listed in the forced_low_output_port_list attribute on the ICL module.
ijtag_c1	Boolean attribute that is set on the design ports of a module that has a corresponding ICL module, if the port name is listed in the forced_high_output_port_list attribute on the ICL module.
ijtag_connection_rule_option	An attribute that reflects the connection_rule_option attribute of the ICL port of the ICL module associated to the design module during ICL extraction.
ijtag_forced_high_dft_signal_list	An attribute that reflects the forced_high_dft_signal_list attribute of the ICL port of the ICL module associated to the design module during ICL extraction.
ijtag_forced_low_dft_signal_list	An attribute that reflects the forced_low_dft_signal_list attribute of the ICL port of the ICL module associated to the design module during ICL extraction.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ijtag_forced_high_output_port_list	An attribute that reflects the forced_high_output_port_list attribute of the ICL port of the ICL module associated to the design module during ICL extraction.
ijtag_forced_low_output_port_list	An attribute that reflects the forced_low_output_port_list attribute of the ICL port of the ICL module associated to the design module during ICL extraction.
ijtag_function	String that reflects the port function of the ports as described in the ICL module. Its value, which is populated in the ICL extraction mode of the patterns -ijtag context, is the port function with an underscore separating the words. For example, a ScanInPort in ICL has icl_function equal to scan_in.
ijtag_function_modifier	An attribute that reflects the function_modifier attribute of the ICL port of the ICL module associated to the design module during ICL extraction.
ijtag_logical_hier_connection_from_src	An attribute that is set on output ports and contains the port on the module that is the logical source of the output port. It is extracted from the associated ICL module attribute ijtag_logical_connection = “{port1 port2} {port3 port4} ...”
ijtag_logical_hier_connection_from_src_inv	An attribute that is set on output ports and contains the port on the module that is the inverted logical source of the output port. It is extracted from the associated ICL module attribute ijtag_logical_connection_inv = “{port1 port2} {port3 port4} ...”
ijtag_logical_hier_connection_to_dst	This attribute is set on input ports and contains the ports on the module that are the logical destinations of the input port. It is extracted from the associated ICL module attribute ijtag_logical_connection = “{port1 port2} {port3 port4} ...”
ijtag_logical_hier_connection_to_dst_inv	This attribute is set on input ports and contains the ports on the module that are the inverted logical destinations of the input port. It is extracted from the associated ICL module attribute ijtag_logical_connection_inv = “{port1 port2} {port3 port4} ...”

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

index	Integer that specifies the index of the port that is already part of the name when the port is part of a bus. In case of a leaf complex port, this attribute always returns a value of 0.
is_bus	Boolean that specifies whether the port is part of a bus. This attribute is set to true when the port is part of a bus. In case of a leaf complex port, this attribute always returns a false value.
is_complex	Boolean that specifies if the port object is from a complex type: • Pre-Synthesis — Specifies if the object is a leaf of or not of a complex port. • Post-Synthesis — Specifies if the object is inferred or not from a RTL complex port.
is_created	Boolean that specifies whether the port was created using the create_port command in insertion mode. The port can be created also by the create_pin command or implicit from some other design editing command. For example, the create_connections and move_connections commands can create intermediate ports, if done over hierarchy levels. In -rtl mode, this attribute value is set back to false once the insertion state is left.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_excluded_from_isolation_constraints	Boolean that specifies if the port should be excluded from CX constraints (input ports) or masking (output ports) when set to true.During core-level pattern generation, the tool automatically adds X constraints on most input ports and adds output masks on all output ports so that generated patterns can be mapped to the chip level for retargeting. Pulse-always clocks, pulse-in-capture clocks, constrained ports and internal primary inputs are excluded from these X constraints.You can exclude additional primary inputs from X constraints by setting this attribute to true; this is necessary if the ports are controlled by a Named Capture Procedure (NCP) that models chip-level constraints such as a clock controller located outside of the core.If a port is not a pulse-always or pulse-in-capture clock, is not constrained, and is excluded from isolation constraints (that is, the attribute is set to true on the port), the create_patterns command checks that the port is controlled (for example, by NCPs or clock control definitions).If the port is not controlled, the tool issues an R7 DRC violation.The default is false.
is_non_editable	Boolean that specifies that the port is non-editable.
is_non_editable_reason	A string that specifies why the port is non-editable as follows: • below_hard_module — The port is inside an instance that has is_hard_module equal to 1 • created_by_quick_synthesis — The port was created by quick synthesis • below_matched_icl_module — The port is below a module that has a matched ICL module • is_part_of_complex_object — The port is a leaf of a complex object
is_valid	Boolean that specifies whether the object still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
library_name	String that specifies the library the port belongs to.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

loadboard_pad_decay	Specifies the leakage time of a pin. The attribute value must be a valid time value and time unit.
loadboard_pull_resistor	Specifies whether the port is a pullup or pulldown. Possible values are zero for pulldown, one for pullup, and an <EMPTY_STRING> for a normal port.
master_module_name	String that specifies the value of the master_name on the module object the port belongs to.
module_name	String that specifies the name of the module object the port belongs to.
name	String that specifies the name of the port object. One port object exists for each bit of a bused port and the name includes the index of the port. For example: A[3] This attribute represents the active name. The active name depends on the selected view, and if the name is synthesized or its original name. This attribute is either the RTL name or synthesized name.
no_control_point	Boolean or literal that prohibits the insertion of a control point on any top-level pin or a pin in any module when the attribute value is true. Possible values are true, false, and unspecified. The default value is unspecified, which means the location can have control points unless the attribute is true on one of its parent modules or instances.
no_control_reason	String that specifies why no control point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified. The no control point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode. For possible values for the tool identified locations and the user-specified locations, refer to no_control_reason in the Module attribute section.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

no_observe_point	Boolean or literal that prevents the insertion of an observe point on a top-level pin or a module pin in any module with the attribute value true. Possible values are true, false, and unspecified. The default value is unspecified, which means the location can have control points unless the attribute is true in one of its parent modules or instances. **Note:** The tool does not consider the no_observe_point attribute while analyzing the clock network to identify the location to drive the clock port on an inserted memory element, such as the flop associated with an observe point.
no_observe_reason	String that specifies why no observe point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified. The no observe point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode. For possible values for the tool identified locations and the user-specified locations, refer to no_observe_reason in the Module attribute section.
object_id	The object_id attribute is a unique id for the object that remains valid while the design is in memory. This attribute is useful to identify the object in a Tcl dictionary. The object_id is only valid within one tool invocation. Once you terminate the tool or delete the elaboration tree, the object_id is no longer valid. The object_id has the form "port: *integer*"
object_type	Returns the object data type “port”.
observe_scan_flop	Reserved for future use.
parent_bundle_name	Optional string that is the bundle parent of this port. For example: For “data_in[2][3].b[1]”, the parent_bundle_name is “data_in[2][3].b”

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

post_synthesis_name	Read-only string that represents the post quick synthesis equivalent name. For example: For “data_in[2][3].b[1]”, the post_synthesis_name is “data_in[40]”
power_domain_name	Returns the name of the power domain of the current design port. This attribute is populated when you read in a UPF or CPF file. For a definition of Power Domain, refer to the *Tessent Glossary*. **Note:** If the power_domain_name attribute is not set on the current design port, its power domain name value is retrieved from the current design power domain name.
pre_synthesis_name	Read-only string that represents the original RTL full name of the port before quick synthesis. For example: For post_synthesis_name “data_in[40]”, the pre_synthesis_name is “data_in[2][3].b[1]”
preferred_core_extraction_endpoint	Boolean that specifies the preferred endpoint for core extraction. In cases of ambiguity, the tool selects the port with this attribute as the extraction endpoint.
preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, the tool creates inferred buffers on all pins of the port in the flat model to preserve the separation of nets inside and outside the module. Any attribute stored on the hierarchical pin is visible in the flat model. If you set this attribute on a pin of the design, the flat model must be reconstructed. The attribute is not lost during flattening.
pull_drive	String value extracted from the cell library data. See the read_cell_library command for information on how cell_type information can be loaded. Note that the LogicVision scang.lib, pad.lib and cell.lib format are natively supported by the read_cell_library command in case your cell library file only includes the ATPG cell modeling information. Possible values are: pull0 | pull1.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

root_bundle_name	Read-only string that returns the root name of this leaf object. For example, for a port with the name “data_in[2][3].b[1]”, the root_bundle_name is “data_in”. For a scalar, for example, with the name “clk”, the root_bundle_name is “clk”.
select_part_value_list	Read-only list of values representing the selected part of the bundle. For example, for “data_in[2][3].b”, the select_part_value_list is “{2 3 b}”. In the case of a scalar with no bitselect, this attribute is assigned an empty list.
special_drive	String value extracted from the cell library data. See the read_cell_library command for information on how cell_type information can be loaded. Note that the LogicVision scang.lib, pad.lib and cell.lib format are natively supported by the read_cell_library command in case your cell library file only includes the ATPG cell modeling information. Possible values are: open_drain | open_source.
static_dft_control_point	Indicates the node was declared as a “add_dft_control_point -type static_dft_control_point” command and ProcessDftSpecification gates it with the specified dft signal.
static_dft_signal	A string attribute that is set on the node where a static DFT signal is defined. The value of the attribute is the name of the static DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a tooltip is shown letting you know that it is the location of the static DFT signal.
tcd_bscan_function	A string attribute that reflects on the port the function the given port has within the matched TCD BoundaryScan description (see “Tessent Core Description” on page 4793). Only module which has the tcd_bscan_name attribute defined has ports with the tcd_bscan_function defined. The property names found in the Interface wrapper gets transferred to the port they map to.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

tcd_memory_function	A string attribute that reflects on the port the function the given port has within the matched TCD Memory description (see "Tessent Core Description" on page 4793). Only module which has the tcd_memory_name attribute defined that has ports with the tcd_memory_function defined. The function name found in the Core/Memory/Port wrapper gets transferred to the port they map to.
tie_value	String that indicates the value to which the port is tied. Possible values are: 0, 1, X, Z, or "" (empty string). The default is the empty string, which means the port is not tied to a value.

wrapper_analysis_status	A read-only string attribute that is only visible in dft -scan context. It has different sets of values before and after analyze_wrapper_cells is executed (provided the analysis results are not invalidated).
	The only values of wrapper_analysis_status available in SETUP mode are "excluded" and "dedicated_wrapper_cell". The values in parentheses correspond to the setting of the set_dedicated_wrapper_cell_options command for that port.
	Possible values are:
	• Pre-analysis:
	  • dedicated_wrapper_cell (on)
	  • unwrapped_or_shared_wrapper_cells (off)
	  • dedicated_or_shared_wrapper_cells (auto)
	  • excluded - The port is either excluded with set_wrapper_analysis_options or automatically ignored
	• Post-analysis:
	  • dedicated_wrapper_cells
	  • shared_wrapper_cells
	  • unwrapped

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

wrapper_analysis_status_reason	A read-only string attribute that is only visible in dft -scan context. • user_specified — set if the port was specified with the "set_dedicated_wrapper_cell_options on" command or "set_wrapper_analysis_options -exclude {}" • bidi_port • black_box_reached • blocked_port • clock_gater_latch_reached — reached a clock gater that is either excluded with "set_clock_gating off" command, or does not drive any scannable cells. • clock_port • combinational_feedthrough • dft_signal • flop_threshold_exceeded — input/output fanout/fan-in flop threshold exceeded • flop_threshold_exceeded_with_feedback — Flop threshold exceeded during feedback tracing • icl_port • input_constrained • interface_only_reached — reached an instance of interface-only module • internal_scan_reached — reached a scan element with is_internal_scan_only attribute, or reached scan segment and "set_wrapper_analysis_options -allow_internal_scan_segments" is off • libcell_levels_threshold_exceeded — input/output fanout/fan-in library cell threshold exceeded • libcell_levels_threshold_exceeded_with_feedback — Library cell threshold exceeded during feedback tracing • non_scan_depth_threshold_exceeded — the number of non-scan sequential elements exceeded during direct or feedback tracing. • scan_chain_reached — reached a pre-existing scan chain • scan_port

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

	• scan_resource_instrument — the specified port did not receive any wrapper cells because its fanout leads to an in-system test controller, and such instruments do not require dedicated wrapper cells. • set_reset_port • specified_wrapper_cell_reached — reached a scan element with specified_wrapper_type attribute set by the user • unconnected_port

Inheritance

Built-in attribute inheritance from associated objects:

- None

User-defined attributes are inherited. See "Attribute Inheritance Behavior" on page 3840.

Port Specification

A port object (port_object) is an input, output, or inout interface of a module, which is specified by a string. Examples of port names can be "In1", "ClkA", and "Out3".

Function Attribute Values

The following list shows the possible values for the function attribute. See the table in the Pin Attributes section of the *Tessent Cell Library Manual* for a complete list of available function values.

- active_high_clock
- active_high_reset
- active_high_set
- active_low_clock
- active_low_reset
- active_low_set
- asynch_disable
- asynch_disable_inv
- asynch_enable
- asynch_enable_inv
- clock_in

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- clock_out
- data_in
- data_out
- data_out_inv
- func_enable
- func_enable_inv
- ground
- mux_in0
- mux_in1
- mux_in2
- mux_in3
- mux_in4
- mux_in5
- mux_in6
- mux_in7
- mux_out
- mux_select
- mux_select0
- mux_select1
- mux_select2
- negedge_clock
- posedge_clock
- power
- power_isolation_external
- power_isolation_internal
- scan_enable
- scan_enable_inv
- scan_in
- scan_out

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- scan_out_inv
- scan_out_no_polarity
- tck
- tdi
- tdo
- test_enable
- test_enable_inv
- tie0
- tie1
- tms
- trst

Pad Function Attribute Values

The following list shows the possible values for the pad_function attribute. See the table in the Pin Attributes section of the *Tessent Cell Library Manual* for a complete list of available function values.

- to_pad
- enable_low
- enable_high
- pad_io
- pad_io_inv
- force_disable
- to_sje_mux_low
- to_sje_mux_high
- to_sji_mux
- to_sjo_mux
- select_jtag_enable
- select_jtag_in
- select_jtag_out
- init_data_dot6

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- init_data_inv_dot6
- init_clock_dot6
- ac_mode_dot6
- from_pad
- from_sje_mux
- from_sji_mux
- from_sjo_mux
- test_data_dot6
- test_data_inv_dot6
- from_io
- from_io_inv
- to_io
- to_inv_io
- keep_tracing
- sample_pad
- input_enable_low
- input_enable_high
- init_posedge_clock_dot6
- init_negedge_clock_dot6
- init_enable_high_dot6
- init_enable_low_dot6
- parametric

Related Topics

get_attribute_list

get_attribute_value_list

get_fanins

get_fanouts

get_common_parent_instance

get_name_list

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

report_attributes

port_bundle

A port_bundle is an object on a module and can have an input, output, or inout direction. In the case of System Verilog Interface, the different bundles of the same original RTL port can have a different direction depending on whether the bundle has a modport.

Port bundles have a simple non-hierarchical name. The name represents a bundle part of the original port. Certain attributes defined on port_bundle are automatically visible by the sub bundles and ports under the given port_bundle. The port object is always a bit, but the port_bundle object may have a complex datatype.

For more information, see "Bundle Object Introspection" in the *Tessent Shell User's Manual*.

The following table describes the attributes associated to port_bundle object type.

See "Bundle Examples" on page 3742 for an example used to describe these attributes.

array_left_index_list	A list of elaborated left ranges of the subtype of the bundle.
array_range_direction_list	A list of directions (up or down) corresponding to the list of ranges represented by array_left_index_list and array_right_index_list.
array_right_index_list	A list of elaborated right ranges of the subtype of the bundle.
array_type_list	A list of range types (packed or unpacked) corresponding to the list of ranges represented by array_left_index_list and array_right_index_list.
array_width_list	A list of widths corresponding to the list of ranges represented by array_left_index_list and array_right_index_list.
associated_scan_interface	String attribute used to associate the following BISR data input ICL ports with a ScanInterface wrapper: • bisr_reset • bisr_select • bisr_mem_disable • parallel_in • serial_repair_enable Because DataInPorts cannot be specified inside the ScanInterface wrapper, the associated_scan_interface and bisr_function attributes enable the memory_bisr plugin to identify all BISR chain ports associated to each power domain. The BISR scan_in, scan_out, clock, and shift_en ICL ports can be extracted from the ScanInterface wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

base_class	String that represents the class category of the base type and completes the base_type definition. Possible values are as follows:4_state_signed | 4_state_unsigned | 2_state_signed | 2_state_unsigned | enum | struct_packed | struct_unpacked | union_packed | union_unpacked | interface | interface_modport | other. See "base_class and base_type Attributes" in the *Tessent Shell User's Manual* for examples of base_class and base_type.
base_type	String that represents the base type of the subtype of the bundle. The base_type is a key or user name that helps in understanding the base type of the bundle. In the case of an array, the base type is the type of the array element. For example: The base_type of "data_in[2][3].b" is "logic". The base_type of "data_in[2][3]" is "struct_unpacked". The base_type of "data_in[2]" is "struct_unpacked".
bisr_function	String that reflects the function of the design port for Tessent Memory Repair. Legal values for this attribute are: Reset, MemoryDisable, Select.
bisr_power_domain_name	An attribute that can be set on a memory instance to associate a BISR power domain name to a memory. You typically do not need to use this attribute when you read in a UPF or CPF file as the power_domain_name attribute is used to infer the power domain of the memories. You use the bisr_power_domain_name attribute if you want to have more BISR power than those described in the UPF/CPF file. Refer to bisr_segment_order_file for specifies.
bus_left_index	Integer that specifies the left index of the complete bus. Null for scalar ports. In the case of a bundle object complex port, this attribute always returns a value of 0.
bus_name	String that specifies the name of the bus without the index. <name> = bus_name [index] when is_bus is true; otherwise, bus_name is null. In the case of complex ports, this attribute returns an empty string. If the bus is one-dimensional, the attribute is a valid string. You need to use the root_bundle_name attribute for bundle objects of complex ports. The root_bundle_name attribute is equivalent to the bus_name attribute, but it works for all datatypes, from simple to complex. See the root_bundle_name attribute description in this table.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

bus_range_direction	String set to “up” or “down” that reflects the bus range direction. This attribute follows the to and downto BIT_VECTOR types in VHDL. In the case of complex ports, this attribute always returns an empty string.
bus_right_index	Integer that specifies the right index of the complete bus. Null for scalar ports. In the case of a bundle object complex port, this attribute always returns a value of 0.
bus_width	Integer that specifies the number of bits independent of whether left_index is smaller, larger than, or equal to right_index. In case of a bundle object complex port, this attribute always returns a value of 0.
clock_reference_source	String defined on a generated clock pin, pointing to its reference pin.
clock_reference_destinations	String defined on a reference pin, pointing to its generated clock pin.
clock_reference_inv_destinations	String defined on an inverted reference pin, pointing to its generated clock pin.
clock_reference_inv_source	String defined on a generated clock pin, pointing its inverted reference pin.
control_scan_flop	Reserved for future use.
created_by_synthesis	Boolean that indicates that the port_bundle was created by synthesis. When a port_bundle has this attribute set to true, it does not exist in the original RTL and cannot be used for design edits. See the set_quick_synthesis_options command.
DFT_Cx_synth_stop_point	Boolean indicating a stop point during fanin tracing from BISTd memory cell clock pins happening during selective (partial) quick synthesis. Selective synthesis speeds up tessent shell execution by synthesizing only those design modules containing RTL constructs in the way of the clock paths to BIST’d memory clock pins. Notice that unconditional full modules synthesis, requiring no such tracing, is instead used when the set_dft_specification_requirements command is specified with the -logic_test option.
dft_clock_enable_func_en	Boolean defined on the functional enable pin of a clock gater cell, as specified by the add_dft_clock_enables command.
dft_clock_enable_test_en	Boolean defined on the test enable pin of a clock gater cell, as specified by the add_dft_clock_enables command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dft_clock_mux_destinations	String defined on the test input pin of a dft_clock_mux, pointing to its output pin. See the add_dft_clock_mux command.
dft_clock_mux_source	String defined on the output of a dft_clock_mux, pointing to its test input pin. See the add_dft_clock_mux command.
dft_control_async_set_reset	Boolean defined on the source of an asynchronous set or reset signal. It can either be inferred by rule DFT_C9 or explicitly set by the "add_dft_control_points -type async_set_reset" command. The process_dft_specification command gates this node with dft signal "async_set_reset_dynamic_disable".
direction	String that specifies the direction of the port. Possible values are as follows: • buffer (VHDL) • inout (Verilog and VHDL) • input (Verilog and VHDL) • linkage (VHDL) • output (Verilog and VHDL) In the case of a bundle object within an SV Interface port, this attribute returns the appropriate direction depending on whether a modport is used or not. If a modport is not used for the bundle, then the direction is inout. If the bundle object is a root SV Interface port, the direction is not defined and this attribute gets an empty string.
dynamic_dft_signal	A string attribute that is set on the node where a dynamic DFT signal is defined. The value of the attribute is the name of the dynamic DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a tooltip is shown letting you know that it is the location of the dynamic DFT signal.
has_functional_source	Returns true unless the port is of direction output and the port_bundle net is undriven or tied. In case of a bundle object complex port, the tracing is also done. The tool computes this attribute correctly.
ijtag_c0	Boolean attribute that is set on the design ports of a module that has a corresponding ICL module, if the port_bundle name is listed in the forced_low_output_port_list attribute on the ICL module.
ijtag_c1	Boolean attribute that is set on the design ports of a module that has a corresponding ICL module, if the port_bundle name is listed in the forced_high_output_port_list attribute on the ICL module.
is_array	Boolean that specifies whether the subtype of this bundle is an array.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_bus	Boolean that specifies whether the port_bundle is part of a bus. This attribute is set to true when the port_bundle is part of a bus. In case of a bundle object complex port, this attribute always returns a false value.
is_complex	Boolean that specifies if the port_bundle object is from a complex type: • Pre-Synthesis — Specifies if the object is a bundle object or not of a complex port. • Post-Synthesis — Specifies if the object is inferred or not from a RTL complex port.
is_non_editable	Boolean that specifies that the port_bundle is non-editable.
is_non_editable_reason	A string that specifies why the port_bundle is non-editable as follows: • below_hard_module — The port_bundle is inside an instance that has is_hard_module equal to 1 • created_by_quick_synthesis — The port_bundle was created by quick synthesis • below_matched_icl_module — The port_bundle is below a module that has a matched ICL module • is_bundle_object — The port has the net_bundle object type
library_name	String that specifies the library the port_bundle belongs to.
master_module_name	String that specifies the value of the master_name on the module object the port_bundle belongs to.
module_name	String that specifies the name of the module object the port_bundle belongs to.
name	String that specifies the name of the port_bundle object. One port_bundle object exists for each bit of a bused port_bundle and the name includes the index of the port_bundle. For example: A[3] This attribute represents the active name. The active name depends on the selected view, and if the name is synthesized or its original name. This attribute is either the RTL name or synthesized name.
object_id	The object_id attribute is a unique id for the object that remains valid while the design is in memory. This attribute is useful to identify the object in a Tcl dictionary. The object_id is only valid within one tool invocation. Once you terminate the tool or delete the elaboration tree, the object_id is no longer valid. The object_id has the form “port_bundle: integer”
object_type	Returns the object type “port_bundle”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

observe_scan_flop	Reserved for future use.
parent_bundle_name	Optional string that represents the bundle parent of this bundle. For example, for “data_in[2][3]”, the parent_bundle_name is “data_in[2]”
post_synthesis_name	Read-only string list of post_synthesis_names. From left to right of the list, each name corresponds to the MSB to LSB of the bundle. For example, for “data_in[2][3]”, the post_synthesis_name is {“data_in[41]” “u_1/data_in[40]” “data_in[39]”}. A Tcl list is shown here, because in some cases (in particular the SystemVerilog Interface), the RTL port has several separate ports after synthesis.
pre_synthesis_name	The original RTL name before synthesis.
preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, the tool creates an inferred buffer on all pins of the port_bundle in the flat model to preserve the separation of nets inside and outside the module. Any attribute stored on the hierarchical pin is visible in the flat model. If you set this attribute on a pin of the design, the flat model must be reconstructed. The attribute is not lost during flattening.
root_bundle_name	Read-only string that returns the root name of this bundle object. For example, for a port with the name “data_in[2][3].b[1]”, the root_bundle_name is “data_in”. For a scalar, for example, with the name “clk”, the root_bundle_name is “clk”.
select_part_value_list	Read-only list of values representing the selected part of the bundle. For example, for “data_in[2][3].b”, the select_part_value_list is “{2 3 b}”. In the case of a scalar with no bitselect, this attribute is assigned an empty list.
static_dft_control_point	Indicates the node was declared with a “add_dft_control_points -type static_dft_control_point” command. The process_dft_specification command gates it with the specified dft signal.
static_dft_signal	A string attribute that is set on the node where a static DFT signal is defined. The value of the attribute is the name of the static DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a tooltip is shown letting you know that it is the location of the static DFT signal.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

tcd_memory_function	A string attribute that reflects on the port the function the given port has within the matched TCD Memory description (see "Tessent Core Description"). Only a module which has the tcd_memory_name attribute defined has port_bundles with the tcd_memory_function defined. The function name found in the Core/Memory/Port wrapper gets transferred to the port_bundle they map to.

Pin

A pin is an object on an instance that can be connected to other pins or ports with net objects. No pin object exists until the set_current_design command is executed. One pin object exists for each bit of a root bundle pin so every bit of a root bundle pin can be attributed independently.

The pin object type interacts with attributes as follows:

- Attributes defined on ports are automatically visible as read-only attributes on the associated pin objects unless the attribute is also registered for the pin object type. This is the case for the name attribute which is defined for both port and pin object types.
- Port attributes are stored on the associated port of the pin and not on the pin object itself.
- Attributes defined on a net are automatically visible by a pin having the exact same name unless the attribute is also defined on the pin object type. The reverse is also true. Attributes defined on a pin are automatically visible by a net having the exact same name unless the attribute is also defined on the net object type. This behavior enables you to register an attribute for net object types and have the attribute fully visible from the root-level ports and the hierarchical pins.

Built-In Attributes

See "Bundle Examples" on page 3742 for an example used to describe these attributes.

active_clock_edge	A string attribute that reflects the value of the gate_pin attribute having the same name when the gate_pin maps to a net object. See the description of the Gate_pin attribute description for the meaning of its value.
active_polarity	A string attribute that reflects the value of the gate_pin attribute having the same name when the gate_pin maps to a net object. See the description of the Gate_pin attribute description for the meaning of its value.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

base_class	String that represents the class category of the base type and completes the base_type definition. Possible values are as follows:4_state_signed | 4_state_unsigned | 2_state_signed | 2_state_unsigned | enum | struct_packed | struct_unpacked | union_packed | union_unpacked | interface | interface_modport | other. See "base_class and base_type Attributes" in the *Tessent Shell User's Manual* for examples of base_class and base_type.
base_type	String that represents the base type of the subtype of the bundle. The base_type is a key or user name that helps in understanding the base type of the bundle. In the case of an array, the base type is the type of the array element. For example: The base_type of "data_in[2][3].b" is "logic". The base_type of "data_in[2][3]" is "struct_unpacked". The base_type of "data_in[2]" is "struct_unpacked".
base_name	String that specifies the name of the pin without the leaf name.
bisr_power_domain_name	An attribute that can be set on BIST_SI pins of sub modules to associate a BISR chain to a power domain defined in the current level. You typically do not need to use this attribute if you have used consistent power domain name in your bottom-up flow but you can use this attribute on pins to map specific BISR chains on sub modules to power domain at the current level if they are different as documented in the bisr_segment_order_file section.
bus_left_index	Integer that specifies the left index of the complete bus. Null for scalar ports. In the case of a leaf complex pin, this attribute always returns a value of 0.
bus_name	String that specifies the name of the bus without the index. <name> = bus_name [index] when is_bus is true; otherwise, bus_name is null. In the case of complex pins, this attribute returns an empty string. If the bus is one-dimensional, the attribute is a valid string. You need to use the root_bundle_name attribute for leaf objects of complex pins. The root_bundle_name attribute is equivalent to the bus_name attribute, but it works for all datatypes, from simple to complex. See the root_bundle_name attribute description in this table.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

bus_right_index	Integer that specifies the right index of the complete bus. Null for scalar ports. In the case of a leaf complex pin, this attribute always returns a value of 0.
bus_width	Integer that specifies the number of bits independent of whether left_index is smaller, larger than, or equal to right_index. In case of a leaf complex port, this attribute always returns a value of 0.
clock_domain_inversion	A string attribute that specifies whether there is an inversion between the clock domain base and the node. When 1, then the inversion exists.
clock_domain_label	A string attribute that is populated when running the DFT_C1, DFT_C2, and DFT_C3 DRC rules. It is attached to every pin found between the clock source and the memory and other clock in its fanout. The string contains the label associated to the clock defined at the source. It is also set by DFT_C6 on every node between the clock and the sequential elements.
clock_reference_source	String defined on a generated clock pin, pointing to its reference pin.
clock_reference_destinations	String defined on a reference pin, pointing to its generated clock pin.
clock_reference_inv_destinations	String defined on an inverted reference pin, pointing to its generated clock pin.
clock_reference_inv_source	String defined on a generated clock pin, pointing its inverted reference pin.
control_scan_flop	Reserved for future use.
created_by_synthesis	Boolean that indicates that the pin was created by synthesis. When a pin has this attribute set to true, it does not exist in the original RTL and cannot be used for design edits. See the set_quick_synthesis_options command.
design_hierarchy_level	Integer value that indicates the depth in the design where the pin exists. For example, pins on instances in the current design module have a value of 1. Pin i1/i2/i3/p6 has a value of 3. You can use this attribute with the sort_collection command to sort the pin according to the topology. This can be useful for design manipulation with auto uniquification.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DFT_Cx_synth_stop_point	Boolean indicating a stop point during fanin tracing from BISTd memory cell clock pins happening during selective (partial) quick synthesis. Selective synthesis speeds up tessent shell execution by synthesizing only those design modules containing RTL constructs in the way of the clock paths to BIST'd memory clock pins. Notice that unconditional full modules synthesis, requiring no such tracing, is instead used when the set_dft_specification_requirements command is specified with the -logic_test option.
dft_clock_enable_func_en	Boolean defined on the functional enable pin of a clock gater cell, as specified by the add_dft_clock_enables command.
dft_clock_enable_test_en	Boolean defined on the test enable pin of a clock gater cell, as specified by the add_dft_clock_enables command.
dft_clock_mux_destinations	String defined on the test input pin of a dft_clock_mux, pointing to its output pin. See the add_dft_clock_mux command.
dft_clock_mux_source	String defined on the output of a dft_clock_mux, pointing to its test input pin. See the add_dft_clock_mux command.
dft_control_async_set_reset	Boolean defined on the source of an asynchronous set or reset signal. It can either be inferred by rule DFT_C9 or explicitly set by the "add_dft_control_points -type async_set_reset" command. The process_dft_specification command gates this node with dft signal "async_set_reset_dynamic_disable".
direction	String that specifies the direction of the pin. Possible values are as follows: • buffer (VHDL) • inout (Verilog and VHDL) • input (Verilog and VHDL) • linkage (VHDL) • output (Verilog and VHDL) In the case of a leaf SV Interface port, this attribute returns the appropriate direction depending on whether a modport is used or not. If a modport is not used, then the direction is returned as inout.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dynamic_dft_signal	A string attribute that is set on the node where a dynamic DFT signal is defined. The value of the attribute is the name of the dynamic DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a tooltip is shown letting you know that it is the location of the dynamic DFT signal.
fault_site	String that is populated in the analysis mode of the patterns -scan context and reports if the pin is a fault site for 0, 1, or both fault polarity types. The attribute is null on pins that are not a fault site for both polarities.
gate_pin_id	Returns a gate_pin id when the pin has an associated gate_pin inside the flat model. Set to true for cell pins associated to fault sites and for hierarchical pins that have an attribute that is set to a non-default value and whose -preserve_boundary_in_flat_model option is set to true.
has_functional_source	Returns true unless the pin is of direction input and the pin is not connected, is connected to an undriven or tied net or is connected to a constant. In case of a leaf complex port, the tracing is also done. The tool computes this attribute correctly.
ignore_during_icl_extraction	Pins of the current design with this attribute set to true are ignored during ICL extraction. This is useful, for example, if you wish to restrict the scope of the ICL extraction. If the ignored pin is an InputPort of an ICL instance, the port is set to unknown ('bx) in the instantiation.
ijtag_logical_hier_connection_from_src	Reflects the result of the add_ijtag_logical_connection commands. Each destination points back to its logical source.
ijtag_logical_hier_connection_from_src_inv	Reflects the result of the add_ijtag_logical_connection commands. Each destination points back to its inverted logical source.
ijtag_logical_hier_connection_to_dst	Reflects the result of the add_ijtag_logical_connection commands. Each source points forward to its logical destinations.
ijtag_logical_hier_connection_to_dst_inv	Reflects the result of the add_ijtag_logical_connection commands. Each source points forward to its inverted logical destinations.
index	Integer that specifies the index of the pin that is already part of the name when the pin is part of a bus. In case of a leaf complex pin, this attribute always returns a value of 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_bus	Boolean that specifies whether the pin is part of a bus. This attribute is set to true when the pin is part of a bus. In case of a leaf complex pin, this attribute always returns a false value.
is_complex	Boolean that specifies if the pin object is from a complex type: • Pre-Synthesis — Specifies if the object is a leaf or not of a complex object. • Post-Synthesis — Specifies if the object is inferred or not from a RTL complex object.
is_hard_module	Boolean that reports the state of the is_hard_module attribute on the instance on which the pin exists.
is_non_editable	Boolean that specifies that the pin is non-editable.
is_non_editable_reason	A string that specifies why the pin is non-editable as follows: • below_hard_module — The pin is on an instance that is inside an instance that has is_hard_module equal to 1 • created_by_quick_synthesis — The pin is on an instance that was created by quick synthesis • below_matched_icl_module — The pin is on an instance that is below a module that has a matched ICL module • is_part_of_complex_object — The pin is a leaf of a complex object
is_valid	Boolean that specifies whether the object still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the name of the pin (value of the name attribute) without the hierarchical path. After leaving insertion mode in the tool, this attribute shows the value of the unrolled_leaf_name attribute.
name	String that specifies the hierarchical name of the pin object relative to the current design. One pin object exists for each bit of a bused port; the name includes the index of the port appended to its associated instance name. For example: u1/u2/A[3]. After leaving insertion mode in the tool, this attribute shows the value of the unrolled_name attribute. This attribute represents the active name. The active name depends on the selected view, and if the name is synthesized or its original name. This attribute is either the RTL name or synthesized name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

no_control_point	Boolean or literal that prohibits the insertion of a control point on a pin in a specific instance when the attribute value is true. Possible values are true, false, and unspecified. The default value is unspecified, which means that the location can have control points unless the attribute is true on one of its parent modules or instances.
no_control_reason	String that specifies why no control point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified. The no control point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode. For possible values for the tool identified locations and the user-specified locations, refer to no_control_reason in the Module attribute section.
no_observe_point	Boolean or literal that prohibits the insertion of an observe point on a pin in a specific instance when the attribute value is true. Possible values are true, false, and unspecified. The default value is unspecified, which means the location can have control points unless the attribute is true on one of its parent modules or instances. **Note:** The tool does not consider the no_observe_point attribute while analyzing the clock network to identify the location to drive the clock port on an inserted memory element, such as the flop associated with an observe point.
no_observe_reason	String that specifies why no observe point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified. The no observe point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode. For possible values for the tool identified locations and the user-specified locations, refer to no_observe_reason in the Module attribute section.
object_id	The object_id attribute is a unique id for the object that remains valid while the design is in memory. This attribute is useful to identify the object in a Tcl dictionary. The object_id is only valid within one tool invocation. Once you terminate the tool or delete the elaboration tree, the object_id is no longer valid. The object_id has the form "pin: integer"

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

object_type	Returns the object data type “pin”.
observe_scan_flop	Reserved for future use.
parent_bundle_name	Optional string that specifies the bundle parent of this pin. For example: For “u_1/data_in[2][3].b[1]”, the parent_bundle_name is “u_1/data_in[2][3].b”
parent_instance	String that specifies the instance name of the instance on which the pin is instantiated. This attribute automatically handles the escaped identifier and returns the instance path of the instance without the <leaf_instance_name>/ <leaf_pin_name> part of the path. For example: parent_instance of pin (u2/u7/a) is u2 parent_instance of pin (u2/b) is null
parent_is_hard_module	Boolean that specifies whether the is_hard_module attribute on the pin’s parent instance is set to true. This attribute is set to true if the is_hard_module attribute on the pin’s parent instance is set to true. The default is false.
post_synthesis_name	Read-only string that represents the post quick synthesis equivalent name. For example: For “u_1/data_in[2][3].b[1]”, the post_synthesis_name is “u_1/data_in[40]”
power_domain_name	Returns the name of the power domain of the pin. This attribute is populated when you read in a UPF or CPF file. For a definition of Power Domain, refer to the *Tessent Glossary*. **Note:** If the attribute is not set on a hierarchical pin, its power domain name value is retrieved from its corresponding instance power domain name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

pre_synthesis_name	String that specifies the original RTL name of the object even when the quick synthesis view of the design is currently in memory. When the quick synthesis view of the design is not in memory, then the value of this attribute equals the value of the name attribute. If the object is created by the synthesis (created_by_synthesis attribute is true which means the object did not exist before the synthesis), then the string is empty. The pre_synthesis_name is different from the name only when the pin has one or more generate blocks in its ancestry that the tool synthesized during quick synthesis. For bundle objects, the string represents the original RTL full name pre quick synthesis. For example: For post_synthesis_name “u_1/data_in[40]”, the pre_synthesis_name is “u_1/data_in[2][3].b[1]”
preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, the tool creates an inferred buffer on the hierarchical pin in the flat model to preserve the separation of nets inside and outside the instance. Any attribute stored on the hierarchical pin is visible in the flat model. If you enable this attribute on a pin of the design, the flat model must be reconstructed. The attribute is not lost during flattening.
pre_tracing_stop_point	Boolean that indicates the pin is a stop point during hierarchical fan-in or fanout tracing. When set to true, it ends hierarchical pre-tracing for the branch at the stop point. Use this to avoid quick synthesis at the stop point.
root_bundle_name	Read-only string that returns the root name of this leaf object. For example, for a pin with the name “u_1/data_in[2][3].b[1]”, the root_bundle_name is “u_1/data_in”. In case of a scalar, for example, with the name “u_1/clk”, the root_bundle_name is “u1/clk”.
select_part_value_list	Read-only list of values representing the selected part of the bundle. For example, for “data_in[2][3].b”, the select_part_value_list is “{2 3 b}”. In the case of a scalar with no bitselect, this attribute is assigned an empty list.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

static_dft_control_point	Indicates the node was declared as a "add_dft_control_points -type static_dft_control_point" command. The process_dft_specification command gates it with the specified dft signal.
static_dft_signal	A string attribute that is set on the node where a static DFT signal is defined. The value of the attribute is the name of the static DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a tooltip is shown letting you know that it is the location of the static DFT signal.
tie_value	String that indicates the value to which the pin is tied. Possible values are: 0, 1, X, Z, or "" (empty string). The default is the empty string which means the pin is not tied to a value.
unrolled_leaf_name	An attribute that indicates whether the tool performed loop unrolling. In insertion mode and if the tool unrolled loops, then this attribute shows the name the object has when written with the write_design command. When there is no loop or the loop is not unrolled, this attribute is the same value as the leaf_name attribute.
unrolled_name	An attribute that indicates whether the tool performed loop unrolling. In insertion mode and if the tool unrolled loops, this attribute shows the name the object has when written with the write_design command. When there is no loop or the loop is not unrolled, this attribute is the same value as the name attribute.
unused_tcd_bscan_external_port	When true, the tool does not include the pins described in the Core()/BoundaryScan/ExternalPort() wrapper of the BoundaryScan core description of a lower-level block. The tool converts any attached data cells to internal cells, and converts enable and AC select cells orphaned by this action to internal cells. If specified, the tool preserves the safe value; otherwise, it uses the inverse value, and the original cell has a control_enable_value. The default is false.

Inheritance

Built-in attribute inheritance from associated port objects:

- bus_left_index
- bus_name
- bus_range_direction

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- bus_right_index
- bus_width
- direction
- function
- ijtag_function
- index
- is_bus
- library_name
- master_module_name
- module_name
- preserve_boundary
- special_drive
- pull_drive

User-defined attributes are inherited. See "Attribute Inheritance Behavior" on page 3840.

Pin Specification

A pin object (pin_object) is specified by a complete hierarchical pin pathname including the pin. For example: /u1/u2/u3/Y refers to pin Y of the hierarchical instance /u1/u2/u3.

Related Topics

get_attribute_list

get_attribute_value_list

get_fanins

get_fanouts

get_name_list

get_pins

report_attributes

pin_bundle

A pin_bundle is an object on an instance that can be connected to other pin bundles or port bundles.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Built-In Attributes

The pin_bundle represents a bundle part of the original pin. Certain attributes defined on a pin_bundle are automatically visible by the sub bundles and pins under the given pin_bundle. The pin object is always a bit, but the pin_bundle object may have a complex datatype.

For more information, see "Bundle Object Introspection" in the *Tessent Shell User's Manual*.

The following table describes the attributes associated to pin_bundle object type.

See "Bundle Examples" on page 3742 for an example used to describe these attributes.

array_left_index_list	A list of elaborated left ranges of the subtype of the bundle.
array_range_direction_list	A list of directions (up or down) corresponding to the list of ranges represented by array_left_index_list and array_right_index_list.
array_right_index_list	A list of elaborated right ranges of the subtype of the bundle.
array_type_list	A list of range types (packed or unpacked) corresponding to the list of ranges represented by array_left_index_list and array_right_index_list.
array_width_list	A list of widths corresponding to the list of ranges represented by array_left_index_list and array_right_index_list.
base_class	A string that represents the class category of the base type and completes the base_type definition. Possible values are as follows:4_state_signed | 4_state_unsigned | 2_state_signed | 2_state_unsigned | enum | struct_packed | struct_unpacked | union_backed | union_unpacked | interface | interface_modport | othertype. See "base_class and base_type Attributes" in the *Tessent Shell User's Manual* for examples of base_class and base_type.
base_name	String that specifies the name of the pin without the leaf name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

base_type	String. This is a key or user name that helps you to understand the base type of this bundle. Possible values are as follows: logic | register | wire | bit | supply0 | supply1 | tri | triand | trior | tri0 | tri1 |wand | wor | byte | shortint | integer | longint | <user_name> | other For example, for “u_1/data_in[2][3].b”, the base_type is “logic”. The base_type of “u_1/data_in[2][3]” is “struct_unpacked”. The base_type of “u_1/data_in[2]” is “struct_unpacked”. See “base_class and base_type Attributes” in the *Tessent Shell User's Manual* for examples of base_class and base_type.
bisr_power_domain_name	An attribute that can be set on BIST_SI pins of sub modules to associate a BISR chain to a power domain defined in the current level. You typically do not need to use this attribute if you have used consistent power domain name in your bottom-up flow but you can use this attribute on pins to map specific BISR chains on sub modules to power domain at the current level if they are different as documented in the bisr_segment_order_file section.
bus_left_index	Integer that specifies the left index of the complete bus. Null for scalar ports. In the case of a bundle object complex port, this attribute always returns a value of 0.
bus_name	String that specifies the name of the bus without the index.<name> = bus_name [index] when is_bus is true; otherwise, bus_name is null. In the case of complex ports, this attribute returns an empty string. If the bus is one-dimensional, the attribute is a valid string. You need to use the root_bundle_name attribute for bundle objects of complex ports. The root_bundle_name attribute is equivalent to the bus_name attribute, but it works for all datatypes, from simple to complex. See the root_bundle_name attribute description in this table.
bus_right_index	Integer that specifies the right index of the complete bus. Null for scalar ports. In the case of a bundle object complex port, this attribute always returns a value of 0.
bus_width	Integer that specifies the number of bits independent of whether left_index is smaller, larger than, or equal to right_index. In case of a bundle object complex port, this attribute always returns a value of 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

clock_reference_source	String defined on a generated clock pin, pointing to its reference pin.
clock_reference_destinations	String defined on a reference pin, pointing to its generated clock pin.
clock_reference_inv_destinations	String defined on an inverted reference pin, pointing to its generated clock pin.
clock_reference_inv_source	String defined on a generated clock pin, pointing its inverted reference pin.
control_scan_flop	Reserved for future use.
created_by_synthesis	Boolean that indicates that the pin was created by synthesis. When a pin has this attribute set to true, it does not exist in the original RTL and cannot be used for design edits. See the set_quick_synthesis_options command.
design_hierarchy_level	Integer value that indicates the depth in the design where the pin_bundle exists. For example, pin_bundles on instances in the current design module have a value of 1. Pin_bundle i1/i2/i3/p[1][2].b[1] has a value of 3. You can use this attribute with the sort_collection command to sort the pin according to the topology. This can be useful for design manipulation with auto uniquification.
DFT_Cx_synth_stop_point	Boolean indicating a stop point during fanin tracing from BIST'd memory cell clock pins happening during selective (partial) quick synthesis. Selective synthesis speeds up tessent shell execution by synthesizing only those design modules containing RTL constructs in the way of the clock paths to BIST'd memory clock pins. Notice that unconditional full modules synthesis, requiring no such tracing, is instead used when the set_dft_specification_requirements command is specified with the -logic_test option.
dft_clock_enable_func_en	Boolean defined on the functional enable pin of a clock gater cell, as specified by the add_dft_clock_enables command.
dft_clock_enable_test_en	Boolean defined on the test enable pin of a clock gater cell, as specified by the add_dft_clock_enables command.
dft_clock_mux_destinations	String defined on the test input pin of a dft_clock_mux, pointing to its output pin. See the add_dft_clock_mux command.
dft_clock_mux_source	String defined on the output of a dft_clock_mux, pointing to its test input pin. See the add_dft_clock_mux command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dft_control_async_set_reset	Boolean defined on the source of an asynchronous set or reset signal. It can either be inferred by rule DFT_C9 or explicitly set by the "add_dft_control_points -type async_set_reset" command. The process_dft_specification command gates this node with dft signal "async_set_reset_dynamic_disable".
direction	String that specifies the direction of the pin_bundle. Possible values are as follows: • buffer (VHDL) • inout (Verilog and VHDL) • input (Verilog and VHDL) • linkage (VHDL) • output (Verilog and VHDL) In the case of a bundle object within an SV Interface port, this attribute returns the appropriate direction depending on whether a modport is used or not.If the bundle object is a root SV Interface pin, the direction is not defined and this attribute gets an empty string. If a modport is not used for the bundle, then the direction is inout.
dynamic_dft_signal	A string attribute that is set on the node where a dynamic DFT signal is defined. The value of the attribute is the name of the dynamic DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a tooltip is shown letting you know that it is the location of the dynamic DFT signal.
has_functional_source	Returns true unless the pin_bundle is of direction input and the pin is not connected, is connected to an undriven or tied net or is connected to a constant. In case of a bundle object complex port, the tracing is also done. The tool computes this attribute correctly.
has_rtl	Boolean that indicates whether the pin_bundle is connected to an RTL construct, such as an always block.
is_array	Boolean that specifies whether the subtype of this bundle is an array.
is_bus	Boolean that specifies whether the pin_bundle is a part of a bus. This attribute is set to true when the pin_bundle is part of a bus. In case of a bundle object complex port, this attribute always returns a false value.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_complex	Boolean that specifies if the pin_bundle object is from a complex type. • Pre-Synthesis — Specifies if the object is a bundle object or not of a complex object. • Post-Synthesis — Specifies if the object is inferred or not from a RTL complex object.
is_non_editable	Boolean that specifies that the pin_bundle is non-editable.
is_non_editable_reason	A string that specifies why the pin_bundle is non-editable as follows: • below_hard_module — The pin_bundle is inside an instance that has is_hard_module equal to 1. • created_by_quick_synthesis — The pin_bundle was created by quick synthesis. • below_matched_icl_module — The pin_bundle is below a module that has a matched ICL module. • is_bundle_object — The pin has the net_bundle object type
leaf_name	String that specifies the name of the pin (value of the name attribute) without the hierarchical path. After leaving insertion mode in the tool, this attribute shows the value of the unrolled_leaf_name attribute.
master_module_name	String that specifies the value of the master_name in which the pin_bundle resides.
module_name	String that specifies the name of the module object in which the pin_bundle resides.
name	String that specifies the hierarchical name of the pin_bundle object relative to the current design. One pin_bundle object exists for each bit of a bused port; the name includes the index of the port appended to its associated instance name. For example: u1/u2/A[3]. After leaving insertion mode in the tool, this attribute shows the value of the unrolled_name attribute. This attribute represents the active name. The active name depends on the selected view, and if the name is synthesized or its original name. This attribute is either the RTL name or synthesized name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

object_id	The object_id attribute is a unique id for the object that remains valid while the design is in memory. This attribute is useful to identify the object in a Tcl dictionary. The object_id is only valid within one tool invocation. Once you terminate the tool or delete the elaboration tree, the object_id is no longer valid. The object_id has the form “pin_bundle: integer”
object_type	Returns the object type “pin_bundle”.
observe_scan_flop	Reserved for future use.
parent_bundle_name	Optional string that represents the bundle parent of this bundle. For example, for “u_1/data_in[2][3]”, the parent_bundle_name is “u_1/data_in[2]”.
parent_instance	String that specifies the instance name of the instance on which the pin_bundle is instantiated. This attribute automatically handles the escaped identifier and returns the instance path of the instance without the <leaf_instance_name>/<leaf_pin_name> part of the path.
parent_is_hard_module	Boolean that specifies whether the is_hard_module attribute on the pin_bundle’s parent instance is set to true. This attribute is set to true if the is_hard_module attribute on the pin_bundle’s parent instance is set to true. The default is false.
post_synthesis_name	Read-only string list of post_synthesis_names. From left to right of the list, each name corresponds to the MSB to LSB of the bundle. For example, for “u_1/data_in[2][3]”, the post_synthesis_name is {“u_1/data_in[41]” “u_1/data_in[40]” “u_1/data_in[39]”}. A Tcl list is shown here, because in some cases (in particular the SystemVerilog Interface), the RTL port has several separate ports after synthesis.
pre_synthesis_name	String that specifies the original RTL name of the object even when the quick synthesis view of the design is currently in memory. When the quick synthesis view of the design is not in memory, then the value of this attribute equals the value of the name attribute. If the object is created by the synthesis (created_by_synthesis attribute is true which means the object did not exist before the synthesis), then the string is empty. The pre_synthesis_name is only different from the name when the pin_bundle has one or more generate blocks in its ancestry that the tool flattened during quick synthesis.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, the tool creates an inferred buffer in the flat model to preserve the separation of nets inside and outside the instance. Any attribute stored on the hierarchical pin is visible in the flat model. If you enable this attribute on a pin_bundle of the design, the flat model must reconstructed. The attribute is not lost during flattening.
root_bundle_name	Read-only string list that represents the root name of this bundle object. For example, for a port with the name “u_1/data_in[2][3].b”, the root_bundle_name is “u_1/data_in”. In case of scalar, for example, with the name “u_1/clk”, the root_bundle_name is “u_1/clk”.
select_part_value_list	String list of values representing the selected part of the bundle. For example: For “u_1/data_in[2][3].b”, the select_part_value_list is “{2 3 b}” In case of a scalar with no bitselect, this attribute returns an empty list.
static_dft_control_point	Indicates the node was declared with a “add_dft_control_points -type static_dft_control_point” command. The process_dft_specification command gates it with the specified dft signal.
static_dft_signal	A string attribute that is set on the node where a static DFT signal is defined. The value of the attribute is the name of the static DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a tooltip is shown letting you know that it is the location of the static DFT signal.
unrolled_leaf_name	An attribute that indicates whether the tool performed loop unrolling. In insertion mode and if the tool unrolled loops, then this attribute shows the name the object has when written with the write_design command. When there is no loop or the loop is not unrolled, this attribute is the same value as the leaf_name attribute.
unrolled_name	An attribute that indicates whether the tool performed loop unrolling. In insertion mode and if the tool unrolled loops, then this attribute shows the name the object has when written with the write_design command. When there is no loop or the loop is not unrolled, this attribute is the same value as the name attribute.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Net

For every port and pin object, a net object with the same name exists. No net objects exist until the set_current_design command is executed. One net object exists for each bit of a root bundle net so every bit of a root bundle net can be attributed independently.

When objects are referenced by name in the editing and attribute commands, if a specified name exists as both a pin and a net or a port and a net, the pin or port is selected. If you want to force the selection of the net object that has the same name as a pin or port, use the get_nets introspection command. For examples of this usage, see the examples shown on the move_connections command reference page.

Attributes defined on a net are automatically visible by a pin or port having the exact same name unless the attribute is also defined on the pin object type. The reverse is also true. Attributes defined on a pin or port are automatically visible by a net having the exact same name unless the attribute is also defined on the net object type. This is true for attributes on pins inherited from their associated port. This behavior enables you to register an attribute for the net object type and have the attribute fully visible from the hierarchical pins and from the ports of the current design.

Built-In Attributes

See "Bundle Examples" on page 3742 for an example used to describe these attributes.

active_clock_edge	A string attribute that reflects the value of the gate_pin attribute having the same name when the gate_pin maps to a net object. See the description of the Gate_pin attribute description for the meaning of its value.
active_polarity	A string attribute that reflects the value of the gate_pin attribute having the same name when the gate_pin maps to a net object. See the description of the Gate_pin attribute description for the meaning of its value.
base_class	String that represents the class category of the base type and completes the base_type definition. Possible values are as follows:4_state_signed | 4_state_unsigned | 2_state_signed | 2_state_unsigned | enum | struct_packed | struct_unpacked | union_packed | union_unpacked | interface | interface_modport | other. See "base_class and base_type Attributes" in the *Tessent Shell User's Manual* for examples of base_class and base_type.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

base_type	String that represents the base type of the subtype of the bundle. The base_type is a key or user name that helps in understanding the base type of the bundle. In the case of an array, the base type is the type of the array element. For example: The base_type of “data_in[2][3].b” is “logic”. The base_type of “data_in[2][3]” is “struct_unpacked”. The base_type of “data_in[2]” is “struct_unpacked”.
bus_left_index	Integer that specifies the left index of the complete bus. Null for scalar ports. In the case of a leaf complex net, this attribute always returns a value of 0.
bus_name	String that specifies the name of the bus without the index. <name> = bus_name [index] when is_bus is true; otherwise, bus_name is null. In the case of complex nets, this attribute returns an empty string. If the bus is one-dimensional, the attribute is a valid string. You need to use the root_bundle_name attribute for leaf objects of complex nets. The root_bundle_name attribute is equivalent to the bus_name attribute, but works for all datatypes, from simple to complex. See the root_bundle_name attribute description in this table.
bus_range_direction	String set to “up” or “down” that reflects the bus range direction. This attribute follows the to and downto BIT_VECTOR types in VHDL. In the case of complex nets, this attribute always returns an empty string.
bus_right_index	Integer that specifies the right index of the complete bus. Null for scalar ports. In the case of a leaf complex net, this attribute always returns a value of 0.
bus_width	Integer that specifies the number of bits independent of whether left_index is smaller, larger than, or equal to right_index. In case of a leaf complex net, this attribute always returns a value of 0.
clock_domain_inversion	A string attribute that specifies whether there is an inversion between the clock domain base and the node. When 1, then the inversion exists.
clock_domain_label	A string attribute that is populated when running the DFT_C1, DFT_C2, and DFT_C3 DRC rules. It is attached to every net found between the clock source and the memory and other clock in its fanout. The string contains the label associated to the clock defined at the source. It is also set by DFT_C6 on every node between the clock and the sequential elements.
clock_reference_source	String defined on a generated clock pin, pointing to its reference pin.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

clock_reference_destinations	String defined on a reference pin, pointing to its generated clock pin.
clock_reference_inv_destinations	String defined on an inverted reference pin, pointing to its generated clock pin.
clock_reference_inv_source	String defined on a generated clock pin, pointing its inverted reference pin.
control_scan_flop	Reserved for future use.
created_by_synthesis	Boolean that indicates that the net was created by synthesis. When a net has this attribute set to true, it does not exist in the original RTL and cannot be used for design edits. See the set_quick_synthesis_options command.
design_hierarchy_level	Integer value that indicates the depth in the design where the net exists. For example, nets in the current design module have a value of 1. Net i1/i2/n5 has a value of 3. You can use this attribute with the sort_collection command to sort the net according to the topology. This can be useful for design manipulation with auto uniquification.
dft_clock_enable_func_en	Boolean defined on the functional enable pin of a clock gater cell, as specified by the add_dft_clock_enables command.
dft_clock_enable_test_en	Boolean defined on the test enable pin of a clock gater cell, as specified by the add_dft_clock_enables command.
dft_clock_mux_destinations	String defined on the test input pin of a dft_clock_mux, pointing to its output pin. See the add_dft_clock_mux command.
dft_clock_mux_source	String defined on the output of a dft_clock_mux, pointing to its test input pin. See the add_dft_clock_mux command.
dft_control_async_set_reset	Boolean defined on the source of an asynchronous set or reset signal. It can either be inferred by rule DFT_C9 or explicitly set by the "add_dft_control_points -type async_set_reset" command. The process_dft_specification command gates this node with dft signal "async_set_reset_dynamic_disable".
direction	String that specifies the direction of the net. Possible values are as follows: • buffer (VHDL) • inout (Verilog and VHDL) • input (Verilog and VHDL) • linkage (VHDL) • output (Verilog and VHDL) In the case of a leaf SV Interface PortNet, this attribute returns the appropriate direction depending on whether a modport is used or not. When a modport is not used for the net, the direction is inout.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

dynamic_dft_signal	A string attribute that is set on the node where a dynamic DFT signal is defined. The value of the attribute is the name of the dynamic DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a tooltip is shown letting you know that it is the location of the dynamic DFT signal.
has_functional_source	Returns true unless the net is undriven or tied. In case of a leaf complex net, the tracing is also done. The tool computes this attribute correctly.
has_rtl	Boolean that indicates whether the net is connected to an RTL construct, such as an always block. Also indicates when the driver of the net is an RTL construct. The attribute is true under the following circumstances: • The net is used in an RTL expression. • The net is directly connected to a complex pin (not via an assign). The attribute is always false in the -no_rtl flow.
has_rtl_fanin	Boolean that indicates whether the net is connected to a driver embedded inside an RTL construct, such as an always block.
has_rtl_fanout	Boolean that indicates whether the net is connected to one or more sinks embedded inside an RTL construct, such as an always block.
index	Integer that specifies the index of the net that is already part of the name when the net is part of a bus. In case of a leaf complex net, this attribute always returns a value of 0.
in_graybox	A read-only attribute that is valid after the analyze_graybox command has been run. You can query this attribute to check if a specific net is included in the graybox. For more information about this attribute, refer to the analyze_graybox command description.
is_bus	Boolean that specifies whether the net is a part of a bus. This attribute is set to true when the net is part of a bus. In case of a leaf complex net, this attribute always returns a false value.
is_complex	Boolean that specifies if the net object is from a complex type: • Pre-Synthesis — Specifies if the object is a leaf of or not of a complex object. • Post-Synthesis — Specifies if the object is inferred or not from a RTL complex object.
is_non_editable	Boolean that specifies that the net is non-editable.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_non_editable_reason	A string that specifies why the net is non-editable as follows: • below_hard_module — The net is inside an instance that has is_hard_module equal to 1 • created_by_quick_synthesis — The net was created by quick synthesis • below_matched_icl_module — The net is below a module that has a matched ICL module • is_part_of_complex_object — The net is a leaf of a complex object
is_valid	Boolean that specifies whether the object still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_block_id	The leaf_name of nets located inside generate blocks have the generate block ids joined by periods prepended to their local name. This attribute returns the block_id part of the leaf_name. It is null for nets that are not inside generate blocks.
leaf_name	String that specifies the name of the net (value of the name attribute) without the hierarchical path. After leaving insertion mode in the tool, this attribute shows the value of the unrolled_leaf_name attribute.
leaf_name_hash	This attribute is normally identical to the leaf_name attribute except when the leaf instance is located inside a generate loop construct inside an RTL file. The leaf_name includes the block id with the loop index such as in loopA[4].u2. The leaf_name_hash attribute is the same except that the loop index is replaced by a hash sign as shown here: loopA[#].u2. You use this attribute if you are editing the content of generate loops and you have not enabled auto uniquification. For the delete_connections and move_connections commands, you must use a lookup table with *parent_module.leaf_name_hash.pin_leaf_name* entries to know if you have already processed that pin. See the examples in the move_connections and get_dft_cell command descriptions for examples of such usage. Note that the value of this attribute always refer to the pre_synthesis_name just like the pre_synthesis_leaf_name attribute.
library_name	String that specifies the library of the module in which the leaf net resides.
master_module_name	String that specifies the value of the master_name in which the leaf net resides.
module_name	String that specifies the name of the module object in which the leaf net resides.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

name	String that specifies the hierarchical name of the net object relative to current design. One net object exists for each bit of a bused net; the name includes the index of the net appended to its associated instance name. For example: u1/u2/A[3]. After leaving insertion mode in the tool, this attribute shows the value of the unrolled_name attribute. This attribute represents the active name. The active name depends on the selected view, and if the name is synthesized or its original name. This attribute is either the RTL name or synthesized name.
object_id	The object_id attribute is a unique id for the object that remains valid while the design is in memory. This attribute is useful to identify the object in a Tcl dictionary. The object_id is only valid within one tool invocation. Once you terminate the tool or delete the elaboration tree, the object_id is no longer valid. The object_id has the form "net: integer"
object_type	Returns the object data type “net”.
observe_scan_flop	Reserved for future use.
parent_bundle_name	Optional string that specifies the bundle parent of this net. For example: For “data_in[2][3].b[1]”, the parent_bundle_name is “data_in[2][3].b”
parent_instance	String that specifies the instance name in which the net exists. This attribute automatically handles the escaped identifier and returns the instance path of the instance without the /<leaf_name> part of the path.
parent_is_hard_module	Boolean that specifies whether the module containing the net has the is_hard_module attribute set to true. This attribute is set to true if the module containing the net has the is_hard_module attribute set to true. The default is false.
post_synthesis_name	Read-only string that represents the post quick synthesis equivalent name. For example: For “data_in[2][3].b[1]”, the post_synthesis_name is “data_in[40]”
power_domain_name	Returns the name of the power domain of the net. **Note:** If the power_domain_name attribute is not set on the net’s corresponding port, its power domain name value is retrieved from the current design power domain name. If the attribute is not set on the net’s corresponding hierarchical pin, its power domain name value is retrieved from its corresponding instance power domain name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

pre_synthesis_leaf_name	String that specifies the original RTL leaf name of the object even when the quick synthesis view of the design is currently in memory. When the quick synthesis view of the design is not in memory, then the value of this attribute equals the value of the leaf_name attribute. If the object is created by the synthesis (created_by_synthesis attribute is true which means the object did not exist before the synthesis), then the string is empty. The pre_synthesis_leaf_name is different from the leaf_name only when the net is inside a generate block that the tool synthesized during quick synthesis.
pre_synthesis_name	String that specifies the original RTL name of the object even when the quick synthesis view of the design is currently in memory. When the quick synthesis view of the design is not in memory, then the value of this attribute equals the value of the name attribute. If the object is created by the synthesis (created_by_synthesis attribute is true which means the object did not exist before the synthesis), then the string is empty. The pre_synthesis_name is different from the name only when the net has one or more generate blocks in its ancestry that the tool synthesized during quick synthesis. Read-only string that represents the original RTL full name before quick synthesis. For example: For post_synthesis_name “data_in[40]”, the pre_synthesis_name is “data_in[2][3].b[1]”
preserve_in_graybox	Boolean attribute that you can set to true prior to issuing the analyze_graybox command to force the specified net to be included in the graybox.
preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, the tool creates an inferred buffer on the net in the flat model to preserve the net name. If you set this attribute on a net of the design, the flat model must be reconstructed. The attribute is not lost during flattening.
root_bundle_name	Read-only string that represents the net name. For example, for a net with the name “data_in[2][3].b[1]”, the root_bundle_name is “data_in”. For a scalar, for example, with the name “clk”, the root_bundle_name is “clk”.
select_part_value_list	Read-only list of values representing the selected part of the bundle. For example, for “data_in[2][3].b”, the select_part_value_list is “{2 3 b}”. In the case of a scalar with no bitselect, this attribute is assigned an empty list.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

specified_internal_scan_only	An attribute used to specify that the flip-flop instance inferred from the given net during quick synthesis is to remain internal during wrapper analysis. Dedicated isolation wrapper cells are inferred to isolate the input and output pins of the instances if any interact with the primary input and output ports.
static_dft_control_point	Indicates the node was declared with a "add_dft_control_points -type static_dft_control_point" command. The process_dft_specification command gates it with the specified dft signal.
static_dft_signal	A string attribute that is set on the node where a static DFT signal is defined. The value of the attribute is the name of the static DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a tooltip is shown letting you know that it is the location of the static DFT signal.
tie_value	String that indicates the value to which the net is tied. Possible values are: 0, 1, X, Z, or "" (empty string). The default is the empty string which means the net is not tied to a value.
unrolled_leaf_name	An attribute that indicates whether the tool performed loop unrolling. In insertion mode and if the tool unrolled loops, then this attribute shows the name the object has when written with the write_design command. When there is no loop or the loop is not unrolled, this attribute is the same value as the leaf_name attribute.
unrolled_name	An attribute that indicates whether the tool performed loop unrolling. In insertion mode and if the tool unrolled loops, then this attribute shows the name the object has when written with the write_design command. When there is no loop or the loop is not unrolled, this attribute is the same value as the name attribute.

Inheritance

Built-in attribute inheritance from associated pin objects:

- Attributes defined on pins or ports having the same name as the net, if the attribute is not also registered on the net object type.

User-defined attributes are inherited. See "Attribute Inheritance Behavior" on page 3840.

Net Specification

A net object (net_object), sometimes called just a net, is a wire that connects the pins of instances in the design. A net is specified by a hierarchical net pathname. In the following example, u5/u2/Q is a pin that drives the net u5/u2_Q. This example is based on the example design presented in the "Hierarchical Design Data Model" section of the *Tessent Shell User's Manual*.

```
SETUP> get_fanouts u5/u2/Q
{u5/u1/A0 u5/u3/A0}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
SETUP> get_fanouts u5/u2/Q -stop_on net
{u5/u2_Q}
```

Related Topics

get_attribute_list

get_attribute_value_list

get_fanins

get_fanouts

get_name_list

get_nets

report_attributes

net_bundle

A net bundle represents a bundle part of the original net. It is a set of bits, and can be the whole original net or a sub signal of the given net.

Built-In Attributes

The net_bundle object type attributes are automatically visible by the sub bundles and nets under the given net_bundle. The net object is always a bit, but the net_bundle objects can have a complex datatype.

For more information, see "Bundle Object Introspection" in the *Tessent Shell User's Manual*.

The following table describes the attributes associated to net_bundle object type.

See "Bundle Examples" on page 3742 for an example used to describe these attributes.

array_left_index_list	A list of elaborated left ranges of the subtype of the bundle.
array_range_direction_list	A list of directions (up or down) corresponding to the list of ranges represented by array_left_index_list and array_right_index_list.
array_right_index_list	A list of elaborated right ranges of the subtype of the bundle.
array_type_list	A list of range types (packed or unpacked) corresponding to the list of ranges represented by array_left_index_list and array_right_index_list.
array_width_list	A list of widths corresponding to the list of ranges represented by array_left_index_list and array_right_index_list.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

base_class	String that represents the class category of the base type and completes the base_type definition. Possible values are as follows:4_state_signed | 4_state_unsigned | 2_state_signed | 2_state_unsigned | enum | struct_packed | struct_unpacked | union_packed | union_unpacked | interface | interface_modport | other See "base_class and base_type Attributes" in the *Tessent Shell User's Manual* for examples of base_class and base_type.
base_type	String that represents the base type of the subtype of the bundle. The base_type is a key or user name that helps in understanding the base type of the bundle. In the case of an array, the base type is the type of the array element. For example: The base_type of "data_in[2][3].b" is "logic". The base_type of "data_in[2][3]" is "struct_unpacked". The base_type of "data_in[2]" is "struct_unpacked".
bus_left_index	Integer that specifies the left index of the complete bus. Null for scalar ports. In the case of a bundle object complex net, this attribute always returns a value of 0.
bus_name	String that specifies the name of the bus without the index. <name> = bus_name [index] when is_bus is true; otherwise, bus_name is null. In the case of complex nets, this attribute returns an empty string. If the bus is one-dimensional, the attribute is a valid string. You need to use the root_bundle_name attribute for bundle objects of complex nets. The root_bundle_name attribute is equivalent to the bus_name attribute, but it works for all datatypes, from simple to complex. See the root_bundle_name attribute description in this table.
bus_range_direction	String set to "up" or "down" that reflects the bus range direction. This attribute follows the to and downto BIT_VECTOR types in VHDL. In the case of complex nets, this attribute always returns an empty string.
bus_right_index	Integer that specifies the right index of the complete bus. Null for scalar ports. In the case of a bundle object complex net, this attribute always returns a value of 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

bus_width	Integer that specifies the number of bits independent of whether left_index is smaller, larger than, or equal to right_index. In case of a bundle object complex net, this attribute always returns a value of 0.
clock_reference_source	String defined on a generated clock pin, pointing to its reference pin.
clock_reference_destinations	String defined on a reference pin, pointing to its generated clock pin.
clock_reference_inv_destinations	String defined on an inverted reference pin, pointing to its generated clock pin.
clock_reference_inv_source	String defined on a generated clock pin, pointing to its inverted reference pin.
control_scan_flop	Reserved for future use.
created_by_synthesis	Boolean that indicates that the net_bundle was created by synthesis. When a net bundle has this attribute set to true, it does not exist in the original RTL and cannot be used for design edits. See the set_quick_synthesis_options command.
design_hierarchy_level	Integer value that indicates the depth in the design where the net_bundle exists. For example, net_bundles in the current design module have a value of 1. Net_bundle i1/i2/n[1][2].b[1] has a value of 3. You can use this attribute with the sort_collection command to sort the pin according to the topology. This can be useful for design manipulation with auto uniquification.
dft_clock_enable_func_en	Boolean defined on the functional enable pin of a clock gater cell, as specified by the add_dft_clock_enables command.
dft_clock_enable_test_en	Boolean defined on the test enable pin of a clock gater cell, as specified by the add_dft_clock_enables command.
dft_clock_mux_destinations	String defined on the test input pin of a dft_clock_mux, pointing to its output pin. See the add_dft_clock_mux command.
dft_clock_mux_source	String defined on the output of a dft_clock_mux, pointing to its test input pin. See the add_dft_clock_mux command.
dft_control_async_set_reset	Boolean defined on the source of an asynchronous set or reset signal. It can either be inferred by rule DFT_C9 or explicitly set by the “add_dft_control_points -type async_set_reset” command. The process_dft_specification command gates this node with dft signal “async_set_reset_dynamic_disable”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

direction	String that specifies the direction of the net_bundle. Possible values are as follows: • buffer (VHDL) • inout (Verilog and VHDL) • input (Verilog and VHDL • linkage (VHDL) • output (Verilog and VHDL) In the case of a bundle object within an SV Interface PortNet, this attribute returns the appropriate direction depending on whether a modport is used or not. If a modport is not used for the bundle, then the direction is inout. If the bundle object is a root SV Interface "PortNet", the direction is not defined and this attribute gets an empty string.
dynamic_dft_signal	A string attribute that is set on the node where a dynamic DFT signal is defined. The value of the attribute is the name of the dynamic DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a tooltip is shown letting you know that it is the location of the dynamic DFT signal
has_functional_source	Returns true unless the net_bundle is undriven or tied. In case of a bundle object complex net, the tracing is also done. The tool computes this attribute correctly.
has_rtl	Boolean that indicates whether the net_bundle is connected to an RTL construct, such as an always block. Also indicates when the driver of the net_bundle is an RTL construct. The attribute is true under the following circumstance: • The net is used in an RTL expression. The attribute is always false in the -no_rtl flow.
is_array	Boolean that specifies whether the subtype of this bundle is an array.
is_bus	Boolean that specifies whether the net_bundle is a part of a bus. This attribute is set to true when the net_bundle is part of a bus. In case of a bundle object complex net, this attribute always returns a false value.
is_complex	Boolean that specifies if the net_bundle object is from a complex type. • Pre-Synthesis — Specifies if the object is a bundle object or not of a complex object. • Post-Synthesis — Specifies if the object is inferred or not from a RTL complex object.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_non_editable	Boolean that specifies that the net_bundle is non-editable.
is_non_editable_reason	A string that specifies why the net_bundle is non-editable as follows: • below_hard_module — The net is inside an instance that has is_hard_module equal to 1. • created_by_quick_synthesis — The net was created by quick synthesis. • below_matched_icl_module — The net is below a module that has a matched ICL module. • is_bundle_object — The net has the net_bundle object type
leaf_block_id	The leaf_name of net_bundles located inside generate blocks have the generate block ids joined by periods prepended to their local name. This attribute returns the block_id part of the leaf_name. It is null for net_bundles that are not inside generate blocks.
leaf_name	String that specifies the name of the net_bundle (value of the name attribute) without the hierarchical path. After leaving insertion mode in the tool, this attribute shows the value of the unrolled_leaf_name attribute.
leaf_name_hash	This attribute is normally identical to the leaf_name attribute except when the leaf instance is located inside a generate loop construct inside an RTL file. The leaf_name includes the block id with the loop index such as in loopA[4].u2. The leaf_name_hash attribute is the same except that the loop index is replaced by a hash sign as shown here: loopA[#].u2. You use this attribute if you are editing the content of generate loops and you have not enabled auto uniquification. For the delete_connections and move_connections commands, you must use a lookup table with parent_module.leaf_name_hash.pin_leaf_name entries to know if you have already processed that pin. See the examples in the move_connections and get_dft_cell command descriptions for examples of such usage. Note that the value of this attribute always refers to the pre_synthesis_name just like the pre_synthesis_leaf_name attribute.
library_name	String that specifies the library of the module in which the net_bundle resides.
master_module_name	String that specifies the value of the master_name in which the net_bundle resides.
module_name	String that specifies the name of the module object in which the leaf net_bundle resides.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

name	This attribute represents the active name. The active name depends on the selected view, and if the name is synthesized or its original name. This attribute is either the RTL name or synthesized name.
object_id	The object_id attribute is a unique id for the object that remains valid while the design is in memory. This attribute is useful to identify the object in a Tcl dictionary. The object_id is only valid within one tool invocation. Once you terminate the tool or delete the elaboration tree, the object_id is no longer valid. The object_id has the form “net: integer”
object_type	Returns the object type “net_bundle”.
observe_scan_flop	Reserved for future use.
parent_bundle_name	Optional string that specifies the bundle parent of the bundle. For example, for “data_in[2][3]”, the parent_bundle_name is “data_in[2]”
parent_instance	String that specifies the instance name in which the net_bundle exists. This attribute automatically handles the escaped identifier and returns the instance path of the instance, but without the /<leaf_name> part of the path.
parent_is_hard_module	Boolean that specifies whether the module containing the net_bundle has the is_hard_module attribute set to true. This attribute is set to true if the module containing the net_bundle has the is_hard_module attribute set to true. The default is false.
post_synthesis_name	Read-only string list of post_synthesis names. From left to right in the list, each name corresponds to the most and least significant bits (MSB and LSB) of the bundle. For example, for “data_in[2][3], the post_synthesis_name is {“data_in[41]” “data_in[40]” “data_in[39]”}. A Tcl list is shown here, because in some cases (in particular the SystemVerilog Interface), the RTL port has several separate ports after synthesis.
pre_synthesis_name	RTL name. The original name before synthesis
preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, the tool creates an inferred buffer on the net_bundle in the flat model to preserve the net name. If you set this attribute on a net of the design, the flat model must be reconstructed. The attribute is not lost during flattening.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

root_bundle_name	Read-only string that returns the root name of the bundle object. For example, for a PortNet with the name “data_in[2][3].b”, the root_bundle_name is “data_in”. In the case of scalar, for example, with the name “clk”, the root_bundle_name is “clk”.
select_part_value_list	Read-only list of values representing the selected part of the bundle. For example, for “data_in[2][3].b”, the select_part_value_list is “{2 3 b}”.
static_dft_control_point	Indicates the node was declared with a “add_dft_control_points -type static_dft_control_point” command. The process_dft_specification command gates it with the specified dft signal.
static_dft_signal	A string attribute that is set on the node where a static DFT signal is defined. The value of the attribute is the name of the static DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a callout box is shown letting you know that it is the location of the static DFT signal.
unrolled_leaf_name	An attribute that indicates whether the tool performed loop unrolling. In insertion mode and if the tool unrolled loops, then this attribute shows the name the object has when written with the write_design command. When there is no loop or the loop is not unrolled, this attribute is the same value as the leaf_name attribute.
unrolled_name	An attribute that indicates whether the tool performed loop unrolling. In insertion mode and if the tool unrolled loops, then this attribute shows the name the object has when written with the write_design command. When there is no loop or the loop is not unrolled, this attribute is the same value as the name attribute.

Bundle Examples

The following examples refer to an input port declared as a SystemVerilog array of structure:

```
typedef struct{
   logic a;
   logic [1:0] b;
} data_t;
input data_t [2:0][4:0] data_in;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-1. Example 1: port_bundle “data_in[1][4].b”

is_array	1
array_left_index_list	1
array_right_index_list	0
array_range_direction_list	down
array_width_list	2
array_type_list	packed
root_bundle_name	data_in
parent_bundle_name	data_in[1][4]
base_type	logic
base_class	4-state_unsigned
object_type	port_bundle
select_part_value_list	1 4 b

Table 11-2. Example 2: port “data_in[1][4].b[0]”

is_array	0
array_left_index_list	
array_right_index_list	
array_range_direction_list	
array_width_list	
array_type_list	
root_bundle_name	data_in
parent_bundle_name	data_in[1][4].b
base_type	logic
base_class	4-state_unsigned
object_type	port
select_part_value_list	1 4 b 0

Table 11-3. Example 3: port_bundle “data_in[1][4]”

is_array	0
array_left_index_list	
array_right_index_list	

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-3. Example 3: port_bundle “data_in[1][4]” (cont.)

array_range_direction_list	
array_width_list	
array_type_list	
root_bundle_name	data_in
parent_bundle_name	data_in[1]
base_type	data_t
base_class	struct_unpacked
object_type	port_bundle
select_part_value_list	1 4

Table 11-4. Example 4: port_bundle “data_in[1]”

is_array	1
array_left_index_list	4
array_right_index_list	0
array_range_direction_list	down
array_width_list	5
array_type_list	packed
root_bundle_name	data_in
parent_bundle_name	data_in
base_type	data_t
base_class	struct_unpacked
object_type	port_bundle
select_part_value_list	1

Table 11-5. Example 5: port_bundle “data_in”

is_array	1
array_left_index_list	2 4
array_right_index_list	0 0
array_range_direction_list	down down
array_width_list	3 5
array_type_list	packed packed
root_bundle_name	data_in

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-5. Example 5: port_bundle "data_in" (cont.)

parent_bundle_name	
base_type	data_t
base_class	struct_unpacked
object_type	port_bundle
select_part_value_list	

pseudo_port

A pseudo_port represents a user-added primary input or primary output.

A pseudo_port object enables the introspection of user-added primary inputs and outputs such as those created using the commands add_primary_inputs, add_primary_outputs, and add_clocks.

A pseudo_port is not a real port in that it is not present in the original design, it cannot be used for design editing, and it is not reflected in designs written out using the write_design command. However, the pseudo_port is reflected in the flat model and it is used for analysis operations that use the flat model such as ATPG and simulation.

A pseudo_port shares some of the same attributes as the port object. Pseudo ports have a simple non-hierarchical name which may be the same as the pin's hierarchical name (for non-grouped pseudo_ports).

Pseudo_ports can be introspected with the get_ports command. The "real" pins that underlie a pseudo_port can be introspected with the "get_pins -of_pseudo_ports *pseudo_port_objects*" command.

The pseudo_port object type does not support attribute inheritance to the real underlying objects (pins/nets) associated to it. That is, the real design object behind the pseudo_port does not see the attributes set on this type of port.

Built-In Attributes

active_clock_edge	A string attribute that reflects the value of the gate_pin attribute having the same name when the gate_pin maps to a net object. See the description of the Gate_pin attribute description for the meaning of its value.
active_polarity	A string attribute that reflects the value of the gate_pin attribute having the same name when the gate_pin maps to a net object. See the description of the Gate_pin attribute description for the meaning of its value.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

array_left_index_list	A list of elaborated left ranges of the subtype of the bundle.
array_range_direction_list	A list of directions (up or down) corresponding to the list of ranges represented by array_left_index_list and array_right_index_list.
array_right_index_list	A list of elaborated right ranges of the subtype of the bundle.
array_type_list	A list of range types (packed or unpacked) corresponding to the list of ranges represented by array_left_index_list and array_right_index_list.
array_width_list	A list of widths corresponding to the list of ranges represented by array_left_index_list and array_right_index_list.
base_class	String that represents the class category of the base type and completes the base_type definition. Possible values are as follows:4_state_signed | 4_state_unsigned | 2_state_signed | 2_state_unsigned | enum | struct_packed | struct_unpacked | union_packed | union_unpacked | interface | interface_modport | other. See "base_class and base_type Attributes" in the *Tessent Shell User's Manual* for examples of base_class and base_type.
base_type	String that represents the base type of the subtype of the bundle. The base_type is a key or user name that helps in understanding the base type of the bundle. In the case of an array, the base type is the type of the array element. For example: The base_type of "data_in[2][3].b" is "logic". The base_type of "data_in[2][3]" is "struct_unpacked". The base_type of "data_in[2]" is "struct_unpacked".
clock_domain_inversion	A string attribute that specifies whether there is an inversion between the clock domain base and the node. When 1, then the inversion exists.
clock_domain_label	A string attribute that is populated when running the DFT_C1, DFT_C2, and DFT_C3 DRC rules. It is attached to every pseudo_port on which a clock was defined. The string contains the label associated to the clock defined on the pseudo_port. It is also set by DFT_C6 on every node between the clock and the sequential elements.
clock_reference_destinations	String defined on a reference pin, pointing to its generated clock pin.
clock_reference_inv_destinations	String defined on an inverted reference pin, pointing to its generated clock pin.
clock_reference_inv_source	String defined on a generated clock pin, pointing its inverted reference pin.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

clock_reference_source	String defined on a generated clock pin, pointing to its reference pin.
control_scan_flop	Reserved for future use.
constraint_value	String that specifies that the port's primary input was set to a constant with the add_input_constraints command. When set to C0, CT0, C1, CT1, CX, or Cz, this attribute displays 0, 0, 1, 1, x, or z, respectively. The default is null.
constraint_value_during_load_unload	String attribute available for top-level ports and pseudo-ports that specifies the value forced at the beginning of load_unload procedure. This attribute is supported in patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis and dft -scan contexts. Possible values are: 0, 1, X and Z.
direction	String that specifies the direction of the pseudo_port. Possible values are as follows: • input • output
exclude_from_mode_internal_patterns	Boolean that excludes a pseudo_port from the pattern file or testbench. When you set this to true, the tool *excludes* the pseudo_port from the pattern file or testbench when you issue the following command: `write_patterns pattern_filename -mode_internal` This attribute only affects the STI based pattern formats, including the Verilog testbench, STIL, WGL, and the other tester formats. **Note:** STIL and WGL patterns written while using this attribute cannot be read back into the Tessent tools. The default value is false.
gate_pin_id	Returns the id of the gate_pin associated with the pseudo_port inside the flat model.
include_in_mode_external_patterns	Boolean that includes a pseudo_port in the pattern file or testbench. When you set this to true, the tool *includes* the pseudo_port in the pattern file or testbench when you issue the following command: `write_patterns pattern_filename -mode_external` This attribute overrides the ALL_EXCLUDE_UNUSED keyword. This attribute only affects the STI based pattern formats, including the Verilog testbench, STIL, WGL, and the other tester formats. **Note:** STIL and WGL patterns written while using this attribute cannot be read back into the Tessent tools. The default value is false.
is_array	Boolean that specifies whether the subtype of this bundle is an array.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is_bus	Boolean that specifies whether the pseudo_port is part of a bus. This attribute is always set to false. The pseudo_port object type does not support busses.
is_valid	Boolean that specifies whether the object still exists.
name	String that specifies the name of the pseudo_port object.
object_id	The object_id attribute is a unique id for the object that remains valid while the design is in memory. This attribute is useful to identify the object in a tcl dictionary. The object_id is only valid within one tool invocation. Once you terminate the tool or delete the design, the object_id is no longer valid. The object_id has the form "pseudo_port: integer"
object_type	Returns the object data type “pseudo_port”.
select_part_value_list	Read-only list of values representing the selected part of the bundle. For example, for “data_in[2][3].b”, the select_part_value_list is “{2 3 b}”. In the case of a scalar with no bitselect, this attribute is assigned an empty list.

Inheritance

Built-in attribute inheritance from associated objects:

- None

User-defined attributes are inherited. See “Attribute Inheritance Behavior” on page 3840.

Related Topics

add_clocks

add_primary_inputs

add_primary_outputs

get_attribute_list

get_attribute_value_list

get_fanins

get_fanouts

get_name_list

get_ports

report_attributes

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Flat Design Data Model

The flat design data model is an internal, flattened representation of the hierarchical design that is produced when exiting setup mode.

You can also explicitly create the flat model using the create_flat_model command.

The flat design data model consists of gates that are connected together. A gate is an instance of a primitive module, and a gate_pin object represents a pin on a gate instance. Gates and gate_pins have no unique instance name, instead having a unique ID that is available to differentiate one from another. The format of the ID is two integers separated by a period character. The first integer represents the gate and the second integer represents the gate_pin. However, this ID does not remain in place from one netlist version to the next or from one Tessent Shell invocation to the next. For that reason, you should avoid hardcoding a gate ID into a script.

Flat Design Object Types **3750**
Gate_pin 3750

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Flat Design Object Types

The gate_pin is the only object type in the flat model.

Gate_pin 3750

Gate_pin

A gate_pin represents a pin on a gate instance in a flattened design.

Built-In Attributes

The gate_pin object type has the following attributes:

active_clock_edge	A string attribute having the value “trailing” or “leading” which is set on the gate_pins of the DFF primitives. The attribute is set on the CK input gate_pin and its associated D input gate_pin. The attribute is also visible on the Q gate_pin. For multi clock DFF primitive, the value seen on the Q gate_pin is the list of all values present on all CK gate_pins. The value of the attribute is “leading” when the clock source has an off state of 0 and there is no inversion between the source and the DFF CK pin or when the clock source has an off state of 1 and there is an inversion between the source and the DFF CK pin. In the other two situations, the value is “trailing”.
active_polarity	A string attribute having the value “on_state” or “off_state” which is set on the gate_pins of the DLAT primitives. The attribute is set on the CK input gate_pin and its associated D input gate_pin. The attribute is also visible on the Q gate_pin. For multi clock DLAT primitive, the value seen on the Q gate_pin is the list of all values present on all CK gate_pins. The value of the attribute is “on_state” when the latch is open when the clock source is set to it its on state. The value of the attribute is “off_state” when the latch is open when the clock source is set to it its off state.
base_name	String that specifies the name of the instance in which the primitive module of the gate_pin is instantiated.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

clock_domain_inversion	A string attribute that specifies whether there is an inversion between the clock domain base and the node. When 1, then the inversion exists.
clock_domain_label	A string attribute that is populated when running the DFT_C1, DFT_C2, and DFT_C3 DRC rules. It is attached to every gate_pin found between the clock source and the memory and other clocks in its fanout. The string contains the label associated to the clock defined at the source. It is also set by DFT_C6 on every node between the clock and the sequential elements.
default_fault_classification_0	String that provides an initial fault classification for the gate_pin on which it is set. This attribute corresponds to stuck-at-0 or slow-to-rise. You use the set_attribute_value command to set the desired default fault classification. During the flow, when the add_faults and read_faults commands are called, or when the tool internally adds faults for scan-based test pattern generation, the fault sites marked with this attribute are initialized with the specified default fault classification, instead of UC. This attribute only applies to the stuck fault model and transition fault model. It also applies to the UDFM fault model with no condition (for example, port faults) so that UDFM continues to work as a superset of stuck/transition. If this is set on a pin that is no-faulted or a non-fault site, it is ignored. This attribute is useful to identify gate_pins which are AU during scan-based test generation, but tested by some other test mode. You identify those faults using DI.<label>, where label identifies the test mode in which those faults are covered. For example, Tessent memory BIST uses the DI.mbisr value to identify the fault in the BISR register which are not testable by scan-based test generation, but covered in the memory BIST test. See Example 4 in the set_attributes_value command for a typical use model

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

default_fault_classification_1	String that provides an initial fault classification for the gate_pin on which it is set.This attribute corresponds to stuck-at-1 or slow-to-fall. You use the set_attribute_value command to set the desired default fault classification. During the flow, when the add_faults and read_faults commands are called, or when the tool internally adds faults for scan-based test generation, the fault sites marked with this attribute are initialized with the specified default fault classification, instead of UC. This attribute only applies to the stuck fault model and transition fault model. It also applies to the UDFM fault model with no condition (for example, port faults) so that UDFM continues to work as a superset of stuck/transition. If this is set on a pin that is no-faulted or a non-fault site, it is ignored. This attribute is useful to identify gate_pins which are AU during scan-based test generation, but tested by some other test mode. You identify those faults using DI.<label>, where label identifies the test mode in which those faults are covered. For example, Tessent memory BIST uses the DI.mbisr value to identify the fault in the BISR register which are not testable by scan-based test generation, but covered in the memory BIST test. See Example 4 in the set_attributes_value command for a typical use model
DFT_Cx_dft_clock_mux_input	Boolean indicating the test input pin of a dft clock mux which was previously added with the add_dft_clock_mux command. That command instructs Tessent Shell to assume the presence of a clock multiplexer during pre-DFT DRC and to insert the multiplexer during DFT insertion as part of the process_dft_specification command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DFT_Cx_dft_clock_mux_output	Boolean indicating the test output pin of a dft clock mux which was previously added with the add_dft_clock_mux command. That command instructs Tessent Shell to assume the presence of a clock multiplexer during pre-DFT DRC and to insert the multiplexer during DFT insertion as part of the process_dft_specification command.
DFT_Cx_occ_inject_node	Boolean tagging the node where a DFT object affecting a clock (for example, a TCK injection multiplexer) should be inserted along the clock path during process_dft_specification, as previously specified by command "add_clocks -dft_inject_node *node*".
DFT_Cx_tag_condition	String indicating what type of DRC check to run on the attributed clock node. Possible values are clock_to_drc (DFT_C1), branch_clock_to_drc (DFT_C2) and clock_ref_to_drc (DFT_C3).
DFT_Cx_valid_clock_source	Boolean indicating the gate pin represents a port or a design pin previously specified as a clock with the add_clocks command.
dft_clock_enable_func_en	Boolean defined on the functional enable pin of a clock gater cell, as specified by the add_dft_clock_enables command.
dft_clock_enable_test_en	Boolean defined on the test enable pin of a clock gater cell, as specified by the add_dft_clock_enables command.
dft_clock_mux_destinations	String defined on the test input pin of a dft_clock_mux, pointing to its output pin. See the add_dft_clock_mux command.
dft_clock_mux_source	String defined on the output of a dft_clock_mux, pointing to its test input pin. See add_dft_clock_mux command.
dft_control_async_set_reset	Boolean defined on the source of an asynchronous set or reset signal. It can either be inferred by rule DFT_C9 or explicitly set by the "add_dft_control_points -type async_set_reset" command. The process_dft_specification command gates this node with dft signal "async_set_reset_dynamic_disable".

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

direction	String that specifies the direction of the gate_pin object. Possible values are as follows: • input • output
dynamic_dft_signal	A string attribute that is set on the node where a dynamic DFT signal is defined. The value of the attribute is the name of the dynamic DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a tooltip is shown letting you know that it is the location of the dynamic DFT signal.
fault_site	String that is populated in the analysis mode of the patterns -scan context and reports if the pin is a fault site for 0, 1, or both fault polarity types. The attribute is null on pins that are not a fault site for both polarities.
gate_id	Integer that specifies the unique ID of the gate to which the gate_pin belongs. This integer is identical to the first integer in the gate_pin's id attribute. This value is not persistent and changes every time you create a flat model.
gate_input_count	Integer that specifies the number of inputs on the gate associated with gate_pin. Use this attribute to filter a collection of gate_pins and quickly separate single input gates from multi-input gates. For example, to find all multi-input gates along the clock tree: get_gate_pins -filter "clock_domain_label && direction == output && primitve name != DFF && primitive_name != DLAT && gate_input_count > 1"
hier_pin_name	String that specifies a hierarchical pin name when the gate_pin is associated to a hierarchical pin. This attribute is null when the gate pin is not associated to a hierarchical pin. Pins on hierarchical instances of type cells and pins on design modules that have an attribute that is set to a non-default value and whose -preserve_boundary_in_flat_model option is set to true are the only hierarchical pins with a corresponding gate_pin.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

id	Specifies a unique ID for each gate_pin that consists of two integers separated by a period: *gate_id.pin_index*
ijtag_logical_hier_connection_from_src	Reflects the result of the add_ijtag_logical_connection commands. Each destination points back to the gate_pin id of its logical source.
ijtag_logical_hier_connection_from_src_inv	Reflects the result of the add_ijtag_logical_connection commands. Each destination points back to the gate_pin id of its inverted logical source.
ijtag_logical_hier_connection_to_dst	Reflects the result of the add_ijtag_logical_connection commands. Each source points forward to the gate_pin id of its logical destinations.
ijtag_logical_hier_connection_to_dst_inv	Reflects the result of the add_ijtag_logical_connection commands. Each source points forward to the gate_pin id of its inverted logical destinations.
is_internal_scan_only	Boolean that specifies whether the gate_pin object is internal scan only.
is_non_scannable	Boolean that specifies that the gate_pin object is non-scannable. Default is false.
is_valid	Boolean that specifies whether the gate_pin object still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the hierarchical pin name without the hierarchical path when the gate_pin is associated to a hierarchical pin. This is null by default.
name	String that specifies name of the gate_pin object. The value is the same as the hier_pin_name attribute when not empty; otherwise, the value is the same as the id attribute of the gate_pin object. Pins on hierarchical instances of type cells and pins on design modules that have an attribute that is set to a non-default value and whose -preserve_boundary_in_flat_model option is set to true are the only hierarchical pins with a corresponding gate_pin.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

no_control_point	Boolean or literal that prohibits the insertion of a control point on a gate_pin in a specific instance when the attribute value is true. Possible values are true, false, and unspecified. The default value is unspecified, which means that the location can have control points unless the attribute is true on one of its parent modules or instances.
no_control_reason	String that specifies why no control point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified. The no control point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode. For possible values for the tool identified locations and the user-specified locations, refer to no_control_reason in the Module attribute section.
no_observe_point	Boolean or literal that prohibits the insertion of an observe point on a gate_pin in a specific instance when the attribute value is true. Possible values are true, false, and unspecified. The default value is unspecified, which means that the location can have observe points unless the attribute is true on one of its parent modules or instances. **Note:** The tool does not consider the no_observe_point attribute while analyzing the clock network to identify the location to drive the clock port on an inserted memory element, such as the flop associated with an observe point.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

no_observe_reason	String that specifies why no observe point can be inserted at the specified location. The possible values fall in two categories, tool identified and user specified. The no observe point locations identified by the tools during analysis have the highest priority and you cannot overwrite or change them. These locations are deleted when you transition back to setup mode. For possible values for the tool identified locations and the user-specified locations, refer to no_observe_reason in the Module attribute section.
object_type	Returns the object data type “gate_pin”.
observe_scan_flop	Reserved for future use.
parent_instance	If the gate_pin is associated to a hierarchical pin, this string specifies the name of the instance on which the hierarchical pin is instantiated. If the gate_pin isn't associated to any hierarchical pin, this attribute specifies the name of the instance on which the primitive module is instantiated.
parent_is_hard_module	Boolean that specifies whether the is_hard_module attribute is set to true on the parent module of the gate_pin. This attribute is false if the parent module of the gate_pin does not have the is_hard_module attribute set to true. This attribute is true for primitives instantiated directly into a design module which does not have is_hard_module set to true. This attribute can also be true for a gate_pin that has the hier_pin_name pointing to a pin on a module instantiated into a non-hard design module.
pin_index	Integer that specifies the *id* of the pin. This integer is identical to the second integer of the gate_pin’s *id* attribute. The pin index is persistent. The pin_index is the index of the gate_pin on the primitive module, where 0 is the index of the single output and integers larger than 0 correspond to the inputs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

primitive_name	Name of the primitive module associated with the gate on which the gate_pin exists. The primitive_name attribute values for all primitive types are listed in Table 11-6.
primitive_port_name	Name of the port on the primitive module. The primitive_port_name attribute values for all primitive types are listed in Table 11-6.
power_domain_name	Returns the name of the power domain in which the gate (instance) to which this gate_pin is attached exists. This attribute is inherited from the instance.
simulation_force_value	String that contains the simulation force value that is specified for a gate_pin with the add_simulation_forces command.
simulation_value	String that contains the simulation value on a gate_pin.
static_dft_control_point	Indicates the node was declared with a "add_dft_control_points -type static_dft_control_point" command and the process_dft_specification gates it with the specified dft signal.
static_dft_signal	A string attribute that is set on the node where a static DFT signal is defined. The value of the attribute is the name of the static DFT signal. This attribute has the -display_in_gui option set to on such that when the node is displayed in the GUI, a tooltip is shown letting you know that it is the location of the static DFT signal.

Built-In Primitives and Primitive Port Names

Table 11-6. Built-in Primitives and Primitive Port Names

Primitive Name	Functional Description	Primitive Port Names	
		Inputs	Outputs
BUF	Buffer	I	O
INV	Inverter	I	O
AND	AND gate	I0, I1, …, In	O
NAND	Negated AND	I0, I1, …, In	O
OR	OR gate	I0, I1, …, In	O

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-6. Built-in Primitives and Primitive Port Names (cont.)

Primitive Name	Functional Description	Primitive Port Names	
		Inputs	Outputs
NOR	Negated OR gate	I0, I1, …, In	O
XOR	Exclusive OR	I0, I1, …, In	O
NXOR	Negated Exclusive OR	I0, I1, …, In	O
DFF	D Flip-flop	S, R, CK (CK0, CK1, … CKn in case of multiple clock/data input pairs), D (D0, D1, ..., Dn in case of multiple clock/data input pairs)	Q
DLAT	D latch	S, R, CK (CK0, CK1, … CKn in case of multiple clock/data input pairs), D (D0, D1, ..., Dn in case of multiple clock/data input pairs)	Q
PI	Primary input	-	O
PO	Primary output	I	-
TIE0	Tie-0	I	O
TIE1	Tie-1	I	O
TIEX	Tie-X	I	O
TSD	Tri-state driver with active high at control input.	EN, D	O
BUS	Bus primitive. Drivers must include at least one TSH or switch gate.	I0, I1, …, In	O

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-6. Built-in Primitives and Primitive Port Names (cont.)

Primitive Name	Functional Description	Primitive Port Names	
		Inputs	Outputs
ZVAL	Z converter gate, converts Z to X: Placeholder gate inserted between output of the gate taking Z and the binary gate input.	I	O
WIRE	Wire	I0, I1, …, In	O
MUX	Multiplexer	S, D0, D1	O
TIEZ	Tie-Z	I0, I1, …, In	O
NMOS	Switch gate	G (gate), S (source)	D (drain)
ZHOLD	Z-hold enables modeling of tri-state nets such that they can retain the previous state when the net goes to a Z value.	I	O
PBUS	Pull-bus gate. Used when combining strong (ZW) bus and weak bus signals (e.g. TIE0) together. The strong value (input 1) always goes to the output, unless the value is Z, in which case the weak value (input 2) propagates to the output.	I0, I1	O
WBUS	Weak bus primitive	I0, I1, …, In	O
BUFZ	Buffer allowing Z value at input.	I	O

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-6. Built-in Primitives and Primitive Port Names (cont.)

Primitive Name	Functional Description	Primitive Port Names	
		Inputs	Outputs
XDET	X-detector gate: input ‘X’ results in output value ‘1’; otherwise output is ‘0’.	I	O
ZDET	Z-detector gate: input ‘Z’ results in output value ‘1’; otherwise output is ‘0’.	I	O
RAM	Read-write memory	S, R, WCK (WCK0, WCK1, … WCKw in case of multiple write ports), WEN (WEN0, WEN1, … WENw in case of multiple write ports), WADDR[0…i] (WADDR0[0…i], WADDR1[0…i], … WADDRw[0…i] in case of multiple write ports), WDIN[0…j] (WDIN0[0…j], WDIN1[0…j], … WDIN w[0…j] in case of multiple write ports), RCK, REN, RADDR[0…i] (RADDR0[0…i], RADDR1[0…i], … RADDRr[0…i] in case of multiple read ports)	Q
ROM	Read-only memory	CK, EN, ADDR[0…i] (ADDR0[0…i], ADDR1[0…i], … ADDRr[0…i] in case of multiple read ports)	Q

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-6. Built-in Primitives and Primitive Port Names (cont.)

Primitive Name	Functional Description	Primitive Port Names	
		Inputs	Outputs
RAM_OUT	Output of a memory, place holder for multiple data outputs of memories in the flat model.	I0, I1, …, In	O
ROM_OUT	Output of a memory, place holder for multiple data outputs of memories in the flat model.	I0, I1, …, In	O
FB_BUF	Feedback buffer: internal gate type to break combinational loops in the design.	I	O
PGEN	Pulse generator: circuitry that creates a pulse at its output when active.	I	O
CORE	Core (obsolete)	I0, I1, …, In	O
CORE_OUT	Output of core, place holder for multiple core outputs in the flat model (obsolete).	I0, I1, …, In	O

Inheritance

Built-in inherited attributes:

- During flattening, attributes defined on hierarchical pins (cell or design) are copied to the gate_pin objects when there is a gate_pin associated to the hierarchical pin. Those are fault sites on cell pins, and those pins that have an attribute that is set to a non-default value and whose -preserve_boundary_in_flat_model option is set to true.

User-defined attributes are inherited. See "Attribute Inheritance Behavior" on page 3840.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Commands

get_attribute_list	get_gate_pins
get_attribute_value_list	get_pins
report_attributes	get_name_list

Gate_pin Specification

A gate_pin object (gate_pin_object) is assigned a number ID rather than having an actual name. The number ID is a unique gate pin object id of the format “gate_id.gate_pin_id”. For example an output of an AND gate in the flattened design might have an ID of “1325.0”.

Related Topics

get_attribute_list

get_attribute_value_list

report_attributes

get_gate_pins

get_pins

get_name_list

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL Data Model

The ICL data model defines objects as one of the following four data types: icl_module, icl_instance, icl_port, and icl_pin. The icl_module and the icl_port object types are created as soon as ICL modules are read in using the read_icl command.

Each icl_module object has its list of icl_port objects. Once the current design is set using the set_current_design command, the ICL data model is elaborated downward from the current design and icl_instance objects are created for each instance of icl_module, and icl_pin objects are created for each port of the modules associated to the icl_instances. The icl_instance and icl_pin objects are hierarchical objects and therefore only exist after the current design is set.

The purpose of the Instrument Connectivity Language (ICL) is to describe the elements that comprise the IJTAG network as well as their logical (though not necessarily physical) connections to each other and to the instruments at the endpoints of the network. ICL bears a loose resemblance to a hierarchical netlist such as Verilog; it is organized by modules that may contain instances of other modules, and it describes the connections between the pins of the instances. However, it is important to note that ICL is not a complete netlist; connections are port-to-port rather than through nets, and ICL freely uses abstraction in order to omit the detailed physical construction of the circuitry — only the behavioral operation of the network is represented.

Beyond merely documenting the structure of the network, the purpose of ICL is to enable an operation called “test pattern retargeting.” With test pattern retargeting, the sequences of test patterns, written in PDL, that are to be applied at the instrument boundaries are mapped through the IJTAG network to a higher level of hierarchy and ultimately to chip level pins. Tessent Shell reads and stores ICL information in a data structure and uses that information to retarget the instrument-level test patterns into test patterns that can then be applied at a higher level.

If a Verilog or VHDL representation has also been read prior to running the set_current_design command, the port list on the design and the ICL module corresponding to the current design is compared to make sure that all ports declared in the ICL module also exist on the design module. The design module can have more ports than the ICL module. In fact, it is likely the design module has more ports because the ICL module only includes the ports with an IJTAG function while the design module includes all ports on the physical module. The testbench created for the current design uses the complete port list read in from the design view if it is supplied.

The ICL data model is a stand-alone representation of the design, separate and distinct from the design files. Object names may differ between the ICL data model and the design data model.

The ICL data model can be extracted from an existing design in Tessent Shell, loaded by itself, or with other design files. Additional attributes are required to bind the ICL objects (icl_module, icl_port, icl_pin, and icl_instance) to the design objects (module, port, pin, and instance). These attributes are named tessent_design_*. If they are not specified in the ICL files, Tessent sets them during ICL elaboration of the current design (whether RTL or gate-level).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In the case of an ICL network extracted by the tool from the existing design, all extracted ICL objects use the post-synthesis name when creating ICL modules. These must be valid ICL names, so ICL extraction replaces any special characters with underscores.

ICL Object Types **3766**
icl_instance 3766
icl_module 3770
icl_pin 3781
icl_port 3784
ICL File Examples With Complex Ports **3806**
Example of Complex Ports as Functional Clocks or Input Port Functions 3806
Example of Grouping ICL Port Definitions for Modeling 3808
ICL Model Attribute Name Examples 3811

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL Object Types

The icl_module, icl_instance, icl_port, and icl_pin object types are described on the following reference pages.

icl_instance 3766
icl_module 3770
icl_pin 3781
icl_port 3784

icl_instance

An icl_instance is a single instantiation of an icl_module. No icl_instance object exists until the set_current_design command is executed.

The icl_instance object type interacts with attributes as follows:

- The icl_instance attributes are automatically visible as read-only attributes on corresponding icl_instances if the icl_instance object type does not have the same attribute registered.
- Attributes defined with the applies_to_child_instances option set to On automatically become visible to all child instances of the icl_instance object.
- The object type icl_instance treats different physical instances as different objects, even if they are associated with the same "Instance" construct in the ICL file (this can happen if the instances have the same parent module, but the parent module is instantiated several times). This introduces ambiguity during write_icl, if an attribute is set to different values on different icl_instance objects associated with the same "Instance" construct. In case of ambiguity, the most recent usage of the commands set_attribute_value or reset_attribute_value determines the value of the attribute in the written ICL file.

For more information on attributes, see "Attributes" on page 3829.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Built-In Attributes

Table 11-7. Built-In Attributes on ICL Instance Object Type

change_rtl_design_instance_to_gate	A Boolean attribute which affects how the value of the ICL instance attribute “hierarchical_design_instance” is computed. When the tool needs the path to a design object in the RTL design during simulation (for example, a clock pin on a physical block) and one or more levels of hierarchy in that path use the IJTAG graybox view which is created from the quick synthesized view, then you must set the “change_rtl_design_instance_to_gate” attribute on the proper ICL instances. In particular, you must set it on any ICL instance whose “tessent_design_instance” attribute refers to a design object which is instantiated in one of those IJTAG graybox views and appears along the path to be computed. When the tool computes the “hierarchical_design_instance” attribute for an ICL instance, it may change some of the components in that path from the RTL name to the gate name. The components which are changed are those which are derived from the “tessent_design_instance” attribute of an ICL instance which has the “change_rtl_design_instance_to_gate” attribute set. Those ICL instances could include the one for which the “hierarchical_design_instance” attribute needs to be computed as well as any of its ICL instance ancestors. To use the “change_rtl_design_instance_to_gate” attribute, you must first reset that attribute on all ICL instances. Then, for a given ICL instance whose “hierarchical_design_instance” attribute is to be computed, set the “change_rtl_design_instance_to_gate” attribute on the proper ICL instances by using the criteria described above. After all the ICL instances of interest have been set, you can retrieve the “hierarchical_design_instance” attributes for the desired ICL instances. It is recommended that you reset all occurrences of this attribute once all paths of interest have been retrieved.
hierarchical_design_instance	A string attribute that returns the complete hierarchical instance path of a design instance to the top. Unlike tessent_design_instance, hierarchical_design_instance is a read-only attribute that is computed internally and contains the complete instance path of a design instance. The path is computed only if the tessent_design_instance attribute is set for each instance in the hierarchy, otherwise an empty string is returned. The tessent_design_instance attribute, on the other hand, is an ICL attribute that is either user-specified or inserted by ICL extraction, and contains only the relative portion of the instance path between a design instance and the design module which matches the ICL module in which the ICL instance is instantiated.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-7. Built-In Attributes on ICL Instance Object Type (cont.)

hierarchical_design_instance_is_valid	A boolean attribute that returns an indication of whether the value (design instance path) of the hierarchical_design_instance attribute exists in the current design. Tessent Shell computes the hierarchical_design_instance_is_valid attribute on demand, so it is a read-only attribute.
is_blackboxed_in_design	Boolean that specifies whether the ICL instance matches or is within a design instance that has been blackboxed using the add_black_boxes command, or read with the -interface_only option.
is_valid	Boolean that specifies whether the ICL instance still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the name of the icl_instance (value of the name attribute) without the hierarchical path.
master_module_name	String that specifies the value of the master_name on the icl_module object associated with the icl_instance.
module_name	String that specifies the name of the ICL module associated to the icl_instance with the @PV# suffix when the instantiation uses parameter overrides.
name	String that specifies the full hierarchical instance path of the icl_instance with '.' as a hierarchy separator.
object_type	Returns the object data type “icl_instance”.
parent_instance	String that specifies the instance name of the ICL instance in which the pin is instantiated. It is null for pins on icl_instances found within the current design.
tessent_clock_domain_labels	String that reports the label of clocks that were added prior to ICL extraction. The value is a list of *label design pin/port*. The attribute is added to ICL instances that represent asynchronous clocks, reference clocks, and branch clocks.
tessent_design_instance	An attribute populated by ICL extract when extracting the top level ICL module. The attribute reflects the hierarchical instance path of the design instance corresponding to the ICL instance at the time of ICL extraction. The design instance path is relative to the current design at the time the ICL module was extracted which is always the parent ICL module of the ICL Instance. To get the complete design instance path of an ICL instance, use the hierarchical_design_instance attribute. ICL Extraction sets the tessent_design_instance attribute to the pre-synthesis instance names whenever those names are available.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-7. Built-In Attributes on ICL Instance Object Type (cont.)

tessent_design_not_found	A boolean attribute that is set to true when the ICL data model is updated to reflect the netlist, and the tessent_design_instance attribute on the ICL instance cannot be matched to an instance in the netlist. The attribute is also set on the ICL instance if it was set on an ancestor ICL instance.

tessent_memory_cells_info	A string attribute describing a Tcl dictionary that gets set on every module for which there is a matched TCD Memory description and is identified as a wrapper that contains exactly one memory cell instance. The Tcl dictionary format for the attribute is as follows:
	`module_name {`
	`  Instances {instance_path}`
	`}`
	Where module_name is the wrapper module name and instance_path is the memory cell instance path relative to the wrapper module.

tessent_ignore_during_icl_verification	A string attribute that specifies whether to include the ICL instance in ICL verification patterns. The attribute has three possible values: true, false, and when_possible. False means include the ICL instance in the ICL verification patterns. True means exclude it. When_possible means exclude the ICL instance unless verification of other ICL objects requires including it. Refer to ICLNetworkVerify or create_icl_verification_patterns for more information about verification patterns.

Inheritance

Built-in attribute inheritance from associated icl_module objects:

- c0_constrained_internal_pin_list
- c0_constrained_pin_list
- c1_constrained_internal_pin_list
- c1_constrained_pin_list
- icl_extraction_time
- is_created
- partially_constrained_bus_pin_list
- port_group_list
- tessent_instrument_type
- test_setup_procfile

User-defined attributes are inherited. See “Attribute Inheritance Behavior” on page 3840.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Commands

get_attribute_list	get_icl_scope
get_attribute_value_list	get_name_list
get_icl_instances	report_attributes

ICL Instance Specification

An ICL instance object is specified by a complete hierarchical ICL instance pathname. For example: “/u1/u2/u3”.

Related Topics

get_attribute_list

get_attribute_value_list

get_icl_instances

get_icl_scope

get_name_list

report_attributes

icl_module

One module is designated as the target module using the set_current_design command. Instances and pins are hierarchical objects defined relative to the current module.

The following properties apply to the icl_module object type:

- An icl_module object has a simple, non-hierarchical name. There can be only one icl_module with a given name within a given namespace. There is only one global namespace for ICL in Tessent Shell.
- An icl_module can be parameterized and its parameters can be overridden during instantiation. For each combination of parameter overrides, an icl_module object corresponding to the parameterized view exists. The name of the parameterized view is uniquified with a @PV# suffix where # is a unique integer starting from 1 and incrementing per unique parameterized view.
- One master icl_module exists for each module definition in the input files. The master icl_module is the one that contains the ICL module definition with default parameter values.
- Attributes defined on icl_modules are automatically visible as read-only attributes on the instance objects when the same attribute name is not also registered on the icl_instance object type.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Attributes defined with the set_attribute_options -applies_to_child_instances switch set to On automatically become visible to all child instances of each instance of the icl_module.
- icl_module attributes are stored on all parameterized views of an icl_module. When an attribute value is set on an icl_module object, the attribute value is set on all icl_module objects having the same master_name within a name space.

For more information on attributes, see "Attributes" on page 3829.

Built-In Attributes

Table 11-8. Built-In Attributes on ICL Module Object Type

bsdl_file	An attribute that is attached to the ICL module by Tessent Shell, if the ICL module contains an AccessLink statement and the -bsdl_file option of the read_icl command has been used. The presence of the AccessLink statement in the ICL module instructs the tool to create ICL for the TAP controller and to integrate it into the IJTAG network. The BSDL file is required for the details of the TAP controller, such as instruction names and opcodes. The attribute bsdl_file holds the information about the BSDL file that provided those TAP controller specifications.
file_path_name	String that specifies the pathname to the file where a given module was read from. This attribute is null for modules created by ICL extract until they have been written.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-8. Built-In Attributes on ICL Module Object Type (cont.)

forced_high_input_port_list	Lists all input ports that were constrained to 1 during ICL extraction. Those constraints are automatically reapplied in the generated patterns. They are also verified to be properly simulated when running ICL extract at the next level. Tessent reports an I5 DRC violation if the simulated value on any of those pins is not 1.
	
	Tessent initially populates this attribute with the pre-synthesis names of the input ports. The tessent_design_rtl_gate_port_mapping attribute stores the mapping between the RTL name (pre_synthesis_name) and the post-synthesis name. Use this when regular port names are changed after real synthesis (for example, they are bit-blasted). The post-synthesis name can be escaped or special characters replaced with an underscore. Use this attribute to find the equivalent post-synthesis names when you go back to Tessent after real synthesis. The tessent_design_rtl_gate_port_mapping attribute is updated with the current post-synthesis names found from the current netlist.
	
	For complex ports constrained to 1 during ICL extraction, Tessent stores the equivalent post-synthesis names in the tessent_design_rtl_gate_port_mapping attribute. This attribute identifies the equivalent design ports in the quick synthesis view, or in the netlist view after logic synthesis. For example:
	
	`Module top {`
	`  // Created by ICL extraction`
	`  ClockPort top_intfA__clk_ {`
	`  Attribute tessent_design_gate_ports = "{\ top_intfA?.clk! }";`
	`  Attribute tessent_design_rtl_ports = "{\ top_intfA? .\clk! }";`
	`  }`
	`  DataInPort top_intfA__extraDataIn__[0] {`
	`  Attribute tessent_design_gate_ports = \ "{\top_intfA?.extraDataIn!! [0]}";`
	`  Attribute tessent_design_rtl_ports = \ "{\top_intfA? .\extraDataIn!! [4]}";`
	`  }`
	`  Attribute tessent_design_format = "sv2009";`
	`  Attribute forced_high_input_port_list = \ "{\top_intfA? .\extraDataIn!! [1]} {\top_intfA? .\ extraDataIn!! [3]}";`
	`  Attribute tessent_design_rtl_gate_port_mapping = \ "{\top_intfA? .\extraDataIn!! [1]} {\ top_intfA?.extraDataIn!! [1]}\ {\top_intfA? .\extraDataIn!! [3]} {\ top_intfA?.extraDataIn!! [3]}";`
	`  ...`
	`  }`

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-8. Built-In Attributes on ICL Module Object Type (cont.)

forced_high_internal_input_port_list	Lists all internal pins that were constrained to 1 during ICL extraction. Tessent reports those constraints in this attribute for debugging. They serve no other purpose. Tessent initially populates this attribute with the pre-synthesis names of the input pins. For internal pins constrained to 1 during ICL extraction, Tessent stores the equivalent post-synthesis names in the tessent_design_rtl_gate_port_mapping attribute. This attribute identifies the equivalent design pins in the netlist view after logic synthesis.
forced_high_output_port_list	Lists output ports of IJTAG instruments that must be forced high during ICL extraction in order to see all the connections between the instruments. A typical use of this attribute is for logictest testmode signals sourced by the ICL network. For example, if a TDR bit drives a TM signal and this TM signal puts the circuit in scan mode, you may need to force it to its inactive value for the rest of the ICL network to be traceable. Tessent initially populates this attribute with the pre-synthesis names of the output ports. For output ports forced high during ICL extraction, Tessent stores the equivalent post-synthesis names in the tessent_design_rtl_gate_port_mapping attribute. This attribute identifies the equivalent design ports in the quick synthesis view, or in the netlist view after logic synthesis.
forced_low_input_port_list	Lists all input ports that were constrained to 0 during ICL extraction. Those constraints are automatically reapplied and verified in the generated patterns running ICL extract at the next level. Tessent reports an I5 DRC violation if the simulated value on any of the ports is not 0. Tessent initially populates this attribute with the pre-synthesis names of the input ports. For input ports constrained to 0 during ICL extraction, Tessent stores the equivalent post-synthesis names in the tessent_design_rtl_gate_port_mapping attribute. This attribute identifies the equivalent design port in the quick synthesis view, or in the netlist view after logic synthesis.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-8. Built-In Attributes on ICL Module Object Type (cont.)

forced_low_internal_input_port_list	Lists all internal pins constrained to 0 during ICL extraction. These constraints are reported for debugging and serve no other purpose. Tessent initially populates this attribute with the pre-synthesis names of the input pins. For internal pins constrained to 0 during ICL extraction, Tessent stores the equivalent post-synthesis names in the tessent_design_rtl_gate_port_mapping attribute. This attribute identifies the equivalent design pins in the quick synthesis view, or in the netlist view after logic synthesis.
forced_low_output_port_list	Lists output ports of IJTAG instruments that must be forced low during ICL extraction to reveal all of the connections between the instruments. You typically use this attribute for logictest testmode signals sourced by the ICL network itself. For example, if you have a TDR bit driving a TM signal and this signals puts the circuit in scan mode, you may need to force it to its inactive value for the rest of the ICL network to be traceable. Tessent initially populates this attribute with the pre-synthesis names of the output ports. For output ports forced low during ICL extraction, Tessent stores the equivalent post-synthesis names in the tessent_design_rtl_gate_port_mapping attribute. This attribute identifies the equivalent design ports in the netlist view after logic synthesis.
icl_extraction_date	Reports the date and time when the ICL module was extracted. It is defined in the ICL module and created by ICL extraction.
ijtag_logical_connection	An attribute that creates a logical connection between source and destination ports. These attributes are transferred to the ijtag_logical_hier_connection_from_src and ijtag_logical_hier_connection_to_dst attributes on the matching module object. ICL extraction traces through those connections as if there was a buffer between the two ports. The syntax is attribute ijtag_logical_connection = “{src1 dst1} {src2 dst2} ...”; The source and destination ports may or may not be declared as ports in the ICL module but they must be declared with the proper direction in the design module object. To create a logical connection that is not between an input and output port of a module having an associated ICL module, use the add_ijtag_logical_connection command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-8. Built-In Attributes on ICL Module Object Type (cont.)

ijtag_logical_connection_inv	An attribute that creates an inverted logical connection between source and destination ports. These attributes are transferred to the ijtag_logical_hier_connection_from_src_inv and ijtag_logical_hier_connection_to_dst_inv attributes on the matching module object. ICL extraction traces through those connections as if there was a buffer between the two ports. The syntax is attribute ijtag_logical_connection_inv = "{src1 dst1} {src2 dst2} ..."; The source and destination ports may or may not be declared as ports in the ICL module but they must be declared with the proper direction in the design module object. To create a logical connection that is not between an input and output port of a module having an associated ICL module, use the add_ijtag_logical_connection command.
is_combinational	Boolean that is set to true when the ICL module consists only of combinational logic. When false, the Boolean specifies the module contains ScanRegisters, DataRegisters, AccessLink constructs, TAP state machines, or other instances of modules that have this attribute set to false. It is also false if there is a problem during evaluation, such as a circular instantiation. In all other cases, it is true.
is_created	Boolean that specifies whether the module was created by ICL extraction. This attribute is set to true for modules created by ICL extraction. This attribute is not exported to the ICL file. Once the ICL module is re-read, this attribute is false.
is_modified	Boolean that specifies whether an ICL module contains attributes that have been added or modified after the module was read in.
is_valid	Boolean that specifies whether the ICL module is deleted or still valid. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
keep_active_during_scan_test	Boolean that specifies that the given ICL module is a node that must remain active during scan test so as not to be disturbed and be usable for scan test_setup. See the create_dft_specification command description for more information on how it is used. The default is "true".
line_number	Integer that specifies a line number in the file where a given module was read from. This attribute is null for modules created by ICL extract until they have been saved and read back in.
master_name	String that specifies the name of the icl_module without the optional parameterized view suffix.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-8. Built-In Attributes on ICL Module Object Type (cont.)

name	String that specifies the name of the icl_module with an optional parameterized view suffix @PV#. ICL modules are defined from files read into the ICl database with the read_icl command. For example, the name of a parameterized view of the master icl_module ModA is ModA@PV1
object_type	Returns the object data type “icl_module”.
partially_forced_input_port_list	Lists all bus ports with at least one element in the forced_low_input_port_list and forced_high_input_port_list. This attribute is useful when creating the testbench using the ICL port list in order to get the complete bus range of the constrained bused ports. This attribute is not used when the Verilog or VHDL design module is read before setting the current design in the patterns -ijtag context because the top-level port list is then extracted from the design module.
pre_trace_stop_point_list	A Tcl list of IJTAG instrument ports that are to act as stop points for the hierarchical pre-tracer. During ICL extraction, pre-tracing ends for branches at these stop points. Use this attribute to avoid quick synthesis of a design module.
scan_interface_list	Lists all ScanInterface names defined within the ICL module.
tessent_boundary_scan_reg	A string attribute that identifies a ScanRegister inside the ICL module as the BOUNDARY register described in the BSDL file. See the section “Requirements on a TAP to be usable for BoundaryScan” in “BoundaryScan” on page 4582 for more information.
tessent_bypass_reg	A string attribute that identifies a ScanRegister inside the ICL module as the BYPASS register described in the BSDL file. See the section “Requirements on a TAP to be usable for BoundaryScan” in “BoundaryScan” on page 4582 for more information.
tessent_design_format	A string attribute that the tool automatically sets during ICL extraction. The attribute stores the format of the corresponding design module (for example, “vhdl_1993”).
tessent_design_id	A string attribute that is set on the current ICL module when it is created during ICL extraction to reflect the design_identifier associated with the current design. See the “set_context -design_identifier” command and switch for more information about this identifier.
tessent_design_level	A string attribute that is set on the current ICL module when it is created during ICL extraction to reflect the design_level associated with the current design. See the set_design_level command for more information about this value.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-8. Built-In Attributes on ICL Module Object Type (cont.)

tessent_design_module	An attribute populated by the extract_icl command when the tool extracts the top-level ICL module. The attribute reflects the name of the design module that corresponds to the ICL module at the time of ICL extraction. Use this attribute in cases where there is a different naming style between the design module name and the ICL module. For example, an escaped name.
tessent_device_id_reg	A string attribute that identifies a ScanRegister inside the ICL module as the DEVID register described in the BSDL file. See the section "Requirements on a TAP to be usable for BoundaryScan" in "BoundaryScan" on page 4582 for more information.
tessent_dft_function	A string attribute used to identify an ICL module having special meaning during DFT. For example, the create_dft_specification command uses this attribute to identify the Sib(sti) and Sib(sri) so that they can be found again during the next insertion pass.
tessent_edge_triggered_reset	A string attribute that identifies the module as a special edge-triggered reset module. This module must have exactly three ports: One ResetPort, one ToResetPort and one DataInPort. The attribute must have the value "falling" or "rising". Depending on the attribute value, when a corresponding rising or falling edge occurs on the DataInPort of the module, the IJTAG engine automatically activates the ToResetPort of the module for one cycle. If no edges are detected, the value of the ToResetPort is sourced from the ResetPort input of the module. For more information, see "Edge-Triggered Local Reset" in the How to Model Global Reset, Local Reset and Embedded TAPs section of the *Tessent IJTAG User's Manual*.
tessent_ignore_during_icl_verification	A string attribute that specifies whether to include the ICL module in ICL verification patterns. The attribute has three possible values: true, false, and when_possible. False means include the ICL module in the ICL verification patterns. True means exclude it. When_possible means exclude the ICL module unless verification of other ICL objects requires including it. Refer to ICLNetworkVerify or create_icl_verification_patterns for more information about verification patterns.
tessent_ignored_design_instances	A string attribute reported by ICL extract to reflect the list of design instances ignored during ICL extraction. These are the design instances with an ICL module for which the Boolean attribute ignore_during_icl_extraction has been set. The format of this attribute is a list of 3-element lists as shown here: {{<*IclModuleName*>} {<*DesignModuleName*>} {<*DesignInstancePath*>} ... }

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-8. Built-In Attributes on ICL Module Object Type (cont.)

tessent_ignore_for_ijtag_graybox	A string attribute that instructs analyze_graybox -ijtag to exclude ICL instances of the given ICL module from being preserved in the IJTAG graybox. Do not set this attribute to on without also setting tessent_ignore_during_icl_verification on the same ICL module because ICL verification simulations are done using the IJTAG graybox view.
	It can have the following values:
	• off — The Tessent ICL instruments that are instances of the specified module are included in the graybox.
	• on — The Tessent ICL instruments that are instances of the specified module are excluded from the graybox.
	• auto — The Tessent ICL instrument is excluded if one of the following is true:
	  • the tessent_instrument_type attribute is set to "mentor::memory_bist" and the tessent_instrument_subtype attribute is one of the following: memory_interface, controller, shared_bus_glue_logic.
	  • the tessent_instrument_memory_module is not set to "off".
	  • the tessent_instrument_type attribute is set to "mentor::memory_bisr" and the tessent_instrument_subtype is set to "fuse_box_interface".
	Any ICL instance that does not satisfy one of the previously listed three conditions is included.

tessent_instruction_reg	A string attribute that identifies a ScanRegister inside the ICL module as the INSTRUCTION register described in the BSDL file. See the section "Requirements on a TAP to be usable for BoundaryScan" in "BoundaryScan" on page 4582 for more information.
tessent_instrument_container	String that specifies the base name of the directory that contains the Tessent Instrument Description (TID) file and any other IP-specific data. This attribute is required for any ICL module that has IP-specific data.
tessent_instrument_subtype	String that identifies the sub-type of the ICL module for one of the known instrument types. The value of this attribute is only used when the tessent_instrument_type attribute is also set.
tessent_instrument_type	String that identifies the ICL module as one of the known instrument types within the tool. When this attribute is set, the tessent_signature attribute must also be set with a valid signature string.
tessent_is_physical_module	Boolean that is set on the top ICL module when the extract_icl command runs and the level was set to chip or physical_block using the set_design_level command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-8. Built-In Attributes on ICL Module Object Type (cont.)

tessent_memory_cells_info	A string attribute describing a Tcl dictionary that gets set on every module for which there is a matched TCD Memory description and is identified as a wrapper that contains exactly one memory cell instance. The Tcl dictionary format for the attribute is as follows:
	`module_name {`
	`  Instances {instance_path}`
	`}`
	Where module_name is the wrapper module name and instance_path is the memory cell instance path relative to the wrapper module.

tessent_repair_word_size	Used for physical blocks with BISR chains. This value reports the maximal repair word size used in the block. The repair word size is the number of bits read from the fuse box. This value is used to generate the compression hardware in the fuse box controller.
tessent_rtl_pre_trace_stop_port_list	Lists the names of ports on which the tool stops the ICL tracing during the ICL extraction pre-trace phase. This is useful to prevent the unnecessary synthesis of module during ICL extraction.
tessent_design_rtl_gate_port_mapping	A list of name pairs that matches the RTL and netlist port names. The list is composed of (RTL name, netlist name) pairs.
tessent_signature	A string that is used to identify that an ICL module as not been modified after its creation. An ICL module can be read in without a tessent_signature or with a valid tessent_signature but a module with an invalid signature generates an errors and is rejected during the read_icl command.
test_setup_procfile	Reports the filename of the procfile that contained the optional test_setup procedure used during ICL extraction. It currently is only informational to remind the user to reuse the same test_setup file during IJTAG pattern retargeting.
tessent_use_in_dft_specification	String that you can set on an ICL module in its ICL file to force the create_dft_specification command to ignore instances of this module. The legal value is auto or false.
tessent_variant_id	A string attribute that is set on the current ICL module when it is created during ICL extraction to reflect the variant ICL identifier set with extract_icl -variant_icl_id *variant_id*. See the -variant_icl_id switch of extract_icl for more information about this identifier.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-8. Built-In Attributes on ICL Module Object Type (cont.)

use_in_icl_extraction	String that specifies if the instances of the corresponding design modules are unconditionally used in ICL Extraction or if they are used only on-demand (corresponding design modules: design modules that have been mapped to the ICL module that owns this attribute). The attribute can have the following values: • always — All instances of the corresponding design modules are used as starting points for ICL Extraction (default) • when_reached_by_other_modules — An instance of a corresponding design module is used in ICL Extraction only if it has been reached by tracing from another instance or top module port.

Related Commands

get_attribute_list	get_icl_scan_interface_port_list
get_attribute_value_list	get_iproc_argument_default
get_icl_fanins	get_iproc_argument_list
get_icl_fanouts	get_iproc_body
get_icl_instances	get_iproc_list
get_icl_modules	get_name_list
get_icl_module_parameter_list	report_attributes
get_icl_module_parameter_value	report_icl_modules
get_icl_ports	report_module_matching
get_icl_scan_interface_list	report_module_matching_options

ICL Module Specification

An ICL module object is specified by an ICL module name which is a string. For example: “modA”.

Related Topics

get_attribute_list

get_attribute_value_list

get_icl_fanins

get_icl_fanouts

get_icl_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

get_icl_modules

get_icl_module_parameter_list

get_icl_module_parameter_value

get_icl_ports

get_icl_scan_interface_list

get_iproc_argument_default

get_iproc_argument_list

get_iproc_body

get_iproc_list

get_name_list

report_attributes

report_icl_modules

report_module_matching

report_module_matching_options

icl_pin

The ICL pin object (icl_pin) is an object on an icl_instance that can be connected to other icl_pin or icl_port objects. No ICL pin object exists until the set_current_design command is executed.

The icl_pin object type interacts with attributes as follows:

- Attributes defined on icl_port objects are automatically visible as read-only attributes on the associated icl_pin objects, unless the attribute is also registered for the icl_pin object type. For example, Tessent defines the tessent_design_gate_ports and tessent_design_rtl_ports attributes for both the icl_port and icl_pin object types.

For more information on attributes, see "Attributes" on page 3829.

Built-In Attributes

Table 11-9. Built-In Attributes on ICL Pin Object Type

base_name	String that specifies the name of the pin without the leaf name.
is_valid	Boolean that specifies whether the icl_pin still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-9. Built-In Attributes on ICL Pin Object Type (cont.)

leaf_name	String that specifies the name of the ICL pin (value of the name attribute) without the hierarchical path.
name	String that specifies the full hierarchical name of the icl_pin with '.' as a hierarchy separator.
object_type	Returns the object data type “icl_pin”.
parent_instance	String that specifies the name of the parent icl_instance of the icl_instance on which the icl_pin is on.
tessent_bisr_power_domain_name	This attribute is only specified for ICL pins of the BISR controller, which is part of the memory repair technology of Tessent MemoryBIST. The attribute is used to identify the power domain group label for each pin.
tessent_ignore_during_icl_verification	A string attribute that specifies to include the ICL pin in ICL verification patterns or not. The attribute has two values: true false. True means include the ICL pin in the ICL verification patterns. False means exclude it. Refer to the ICLNetworkVerify wrapper or the create_icl_verification_patterns command for more information about verification patterns.
tie_value	String that indicates the value to which the port is tied. Possible values are: 0, 1, X, or “” (empty string). The default is the empty string, which means the port is not tied to a value.

Inheritance

Built-in attribute inheritance from associated icl_port objects:

- bus_left_index
- bus_name
- bus_range_direction
- bus_right_index
- bus_width
- direction
- freq_multiplier
- freq_divider
- ijtag_function
- index

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- is_bus
- master_module_name
- module_name
- period
- port_group_list
- tessent_design_gate_ports

  This attribute is populated by the extract_icl command when the tool extracts the top-level ICL module. The attribute identifies the post-synthesis name of the design port corresponding to the ICL port. It identifies the design object in a quick synthesis view and, in no RTL mode, maps the ICL port to a design port in the post-synthesis netlist. Examples (using escaped identifiers):

  - Post-synthesis name: \data#[5]in [35]
  - Pre-synthesis name: \data#[5]in [2][1].a
  - Root bundle name: \data#[5]in
  - ICL port name: data__5_in[35]
  - tessent_design_rtl_ports: \data#[5]in [2][1].a
  - tessent_design_gate_ports: \data#[5]in [35]
- tessent_design_rtl_ports

  This attribute specifies the pre_synthesis_name of the design port corresponding to the ICL port at the time of ICL extraction. It identifies the design object in an RTL view, and in no RTL mode, maps the ICL port to a design port in the post-synthesis netlist. For example:

  - Post-synthesis name: data_in[35]
  - Pre-synthesis name: data_in[2][1].a
  - Root bundle name: data_in
  - ICL port name: data_in[35]
  - tessent_design_rtl_ports: data_in[2][1].a

User-defined attributes are inherited. See "Attribute Inheritance Behavior" on page 3840.

Pin Specification

An ICL pin object is specified by a complete hierarchical ICL pin pathname including the ICL pin. For example: /u1/u2/u3/Y refers to pin Y of the hierarchical instance /u1/u2/u3.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example With Complex Ports

Table 11-10. ICL Names and Associated Attributes

ICL Name	tessent_design_rtl_ports	tessent_design_gate_ports
u1.data_in[35]	data_in[2][1].a	data_in[35]
u1.data_in[34]	data_in[2][1].b[1]	data_in[34]
u1.data_in[33]	data_in[2][1].b[0]	data_in[33]

Related Topics

extract_icl

get_attribute_list

get_attribute_value_list

get_icl_fanins

get_icl_fanouts

get_icl_pins

get_icl_instances

get_iclock_list

get_iclock_option

get_name_list

report_attributes

report_ijtag_logical_connections

icl_port

Several properties apply to the icl_port object type.

- An icl_port object has a simple non-hierarchical name.
- An icl_port is an object on an icl_module and can have an input or output direction.
- One port object exists for each bit of a bused port so every bit of a bussed port can be attributed independently.
- Attributes defined on icl_port objects are automatically visible as read-only attributes on the associated icl_pin objects unless the attribute is also registered for the icl_pin object type.
- Attributes set on an icl_port of one icl_module are set on the same port of each icl_module having a common master_name. Because different parameterized views can have more or less icl_port objects when the size of a bussed icl_port is parameterized,

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

attributes on the other icl_module having the common master_name are only copies if the port object actually exists on the given parameterized view.

For more information on attributes, see "Attributes" on page 3829.

Built-In Attributes

The ICL port object (icl_port) represents the ICL view of a design port. Tessent derives the ICL port name from the post-synthesis name after removing all illegal characters. The attributes described in Table 11-11 are built-in for this object to match the ICL ports with design objects in RTL mode and after logic synthesis.

Table 11-11. Built-In Attributes on ICL Port Object Type

active_polarity	An integer attribute defined on the ResetPort and ToResetPort ICL port objects which reflects the value of the ActivePolarity property in the ICL definition.
associated_scan_interface	String attribute used to associate the following BISR data input ICL ports with a ScanInterface wrapper: • bisr_reset • bisr_select • bisr_mem_disable • parallel_in • serial_repair_enable Because you cannot specify DataInPorts inside the ScanInterface wrapper, the associated_scan_interface and tessent_bisr_function attributes enable the memory_bisr plugin to identify all BISR chain ports associated to each power domain. You can extract he BISR scan_in, scan_out, clock, and shift_en ICL ports from the ScanInterface wrapper.
bus_left_index	Integer that specifies the left index of the complete bus. Null for scalar ports.
bus_name	String that specifies the name of the bus without the index. <name> = bus_name [index] when is_bus is true; otherwise, bus_name is null.
bus_range_direction	String set to "up" or "down" that reflects the bus range direction.
bus_right_index	Integer that specifies the right index of the complete bus. Null for scalar ports.
bus_width	Integer that specifies the number of bits independent of whether left_index is smaller, larger than, or equal to right_index.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

compliance_value	This attribute specifies the compliance value of the ICL port and as such represents the COMPLIANCE_PATTERNS specification of an IEEE 1149.1 BSDL file. The attribute value can be specified as a sized or unsized binary number using legal ICL digits 0, 1 and X. If the attribute value is specified as a sized number, the size must match the size of the object. If the attribute value is specified as an unsized number, it is padded with 0s or Xs if necessary, dependent on the leftmost specified bit. The right-most bit (the least significant bit) of the attribute value represents the right-most bit of the port, the second right-most bit of the value refers to the second right-most bit of the port, and so on. This attribute is set automatically on all compliance enable ports, if the ICL top module contains an AccessLink statement of type STD_1149_1_2001 or STD_1149_1_2013. If the corresponding BSDL file contains more than one compliance pattern, the first compliance pattern is used to derive the attribute values. The compliance_value attribute determines the values that are applied to the ICL ports at the beginning of an IJTAG pattern set, see open_pattern_set.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

connection_rule_option	Specified inside the Port wrappers of the ICL modules and used to modify the connection rules for the corresponding port. You use one of the following values to achieve the described behavior. • allowed_no_source — Allowed on input ports of an ICL instrument to indicate that this input port is allowed to not connect to a valid source in the design. Typically, you use this for DataInPort and ScanInPorts of host ports that are not always used in all instantiations. This attribute is only relevant for ICL extraction and has no affect on ICL elaboration. When you set allowed_no_source on an ICL instrument port that requires input binding, and there is no connection to the port in the design, ICL extraction inserts an "InputPort *port* = 1'bX" statement in the ICL instrument instantiation for that port. This ensures compliance with the IEEE 1687 standard (checked by ICL49) and the port functions as intended by design (not connected). • allowed_tied — Allowed on input ports of an ICL instrument to indicate that this input port is allowed not to be connected to a valid source as long as the port is tied high or low. Typically, you use this for DataInPort and ScanInPorts of host ports which are not always used in all instantiations. • allowed_tied_low — Allowed on input ports of an ICL instrument to indicate that this input port is allowed not to be connected to a valid source as long as the port is tied low. You typically use this for DataInPort and ScanInPorts of host ports which are not always used in all instantiations. • allowed_tied_high — Allowed on input ports of an ICL instrument to indicate that this input port is allowed not to be connected to a valid source as long as the port is tied high. Typically, you use this for DataInPort and ScanInPorts of host ports which are not always used in all instantiations. • allowed_no_destination — Allowed on output ports of an ICL instrument to indicate that this output port is allowed not to be connected to a valid destination. You typically use this for DataOutPort and ScanOutPorts of host ports which are not always used in all instantiations.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

connection_rule_option (continued...)	• auxiliary_data — Identifies the DataOutPort as an active-high auxiliary data pin. When tracing from a port with connection rule option “auxiliary_data”, ICL extraction asserts all ports on all instances declared as an auxiliary_data or auxiliary_data_inverse to their inactive values, except for the current port which is not forced. This enables ICL extraction to trace through OR-like combinational logic that combines several active-high auxiliary data pins into one single data signal. That this logic is represented by a OneHotDataGroup in the generated ICL. Therefore, the involved DataOutPorts must have the “Enable” property or an equivalent enabling condition to pass the ICL75 check. • auxiliary_data_inverse — Identifies the DataOutPort as an active-low auxiliary data pin. When tracing from a port with connection rule option “auxiliary_data_inverse”, ICL extraction asserts all ports on all instances declared as an auxiliary_data or auxiliary_data_inverse to their inactive values, except for the current port which is not forced. This enables ICL extraction to trace through AND-like combinational logic that combines several active-low auxiliary data pins into one single data signal. That this logic is represented by a OneHotDataGroup in the generated ICL. Therefore, the involved DataOutPorts must have the “Enable” property or an equivalent enabling condition to pass the ICL75 check. • auxiliary_data_enable — Identifies the DataOutPort as an auxiliary data enable pin. When tracing a port with a forced_low_output_port_list or a forced_high_output_port_list attribute pointing to a port declared as an auxiliary_data_enable or auxiliary_data_enable_inverse, ICL extract asserts all ports on all instances declared as an auxiliary_data_enable or auxiliary_data_enable_inverse to their inactive values, except for the ones referenced by the current port which are forces to their active values instead. This enables ICL extract to extract DataOut ports connected through an AND OR mux like this: (d1 & en1) | (d2 & en2) | (d3 & en3); • auxiliary_data_enable_inverse — Identifies the DataOutPort as an auxiliary data enable inverse pin.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

connection_rule_option (continued...)	• must_connect_to_top_port — Allowed on input and output ports of an ICL instrument to indicate that this input or output port is only allowed to be connected to a primary input or output pin of the current design. For example, a ScanInPort is allowed to be sourced by a ScanOutPort of another instance. If this attribute is set to must_connect_to_top_port, it is only allowed to be sourced by a primary input port.
default_load_value	This attribute corresponds to the DefaultLoadValue that is specified on DataIn ports in the ICL. Possible values are 0, 1, or X, where X is specified when the DataIn port does not have a specified default load value.
differential_inv_of	An icl_port attribute that specifies a functional clock. The value is computed from the optional DifferentialInvOf keyword of the ClockPort or ToClockPort statement.
direction	Direction of the icl_port. Possible values are as follows: • input • output
exclude_from_sdc	An attribute that is used to exclude ICL module ports from the SDC. • When set to true on a ToClockPort, the extract_sdc command ignores the ICL port when tracing functional clocks and does not add a generated clock. Typically, this is a module port that has a feedthrough connection of a functional clock but is not a clock domain base. • When set to true on a TCKPort, the extract_sdc command ignores the ICL port as a source for any TCK-generated clock required by the DFT. Typically, this is a muxing module having two TCKPorts and one ToTckPort. In that case, use the exclude_from_sdc attribute to identify the TCKPort to exclude from tracing. The tool traces using the other TCKPort to find the source of TCK for the DFT instruments. The default is false.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

explicit_iwrite_only	An attribute used to specify that the value of an ICL data object cannot be changed by the retargeter unless it is the target of an explicit iWrite command. In the ICL, you can specify the attribute inside DataInPort, DataOutPort, ScanRegister and DataRegister elements. The attribute value can be specified as a sized or unsized binary number using legal ICL numbers 0 and 1, where a 1 activates the described functionality on the corresponding port bit, while 0 represents ordinary behavior. If you specify the attribute value as a sized number, the size must match the size of the object. If you specify the attribute value as an unsized number, it is padded with 0s if necessary. The right-most bit (the least significant bit) of the attribute value represents the rightmost bit of the port, the second right-most bit of the value refers to the second right-most bit of the port, and so on. You can also specify the attribute value using string values that are simply binary, decimal, or hexadecimal numbers. There can be no complex expressions if the attribute is specified as a string but concatenation is allowed. You can use this attribute to have full control over parts of the scan path configuration and prevent the retargeter from always using the shortest path.
forced_high_dft_signal_list	This attribute is specified inside the Port wrappers of the ICL modules and is used to list DFT signals that must be forced high during the tracing of the current port. The pins or ports that correspond to the listed DFT signals are simulated high and appended to the stable_after_setup simulation condition when tracing the associated port.
forced_low_dft_signal_list	This attribute is specified inside the Port wrappers of the ICL modules and is used to list DFT signals that must be forced low during the tracing of the current port. The pins or ports that correspond to the listed DFT signals are simulated low and appended to the stable_after_setup simulation condition when tracing the associated port.
forced_low_output_port_list	This attribute is specified inside the Port wrappers of the ICL modules and is used to list output ports on the IJTAG instrument that must be force low during the tracing of the current output port. The listed ports are simulated low and appended to the stable_after_setup simulation condition when tracing the associated output port. For example, tdoEnable typically needs to be forced active when tracing the tdo port of a TAP controller.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

forced_high_output_port_list	This attribute is specified inside the Port wrappers of the ICL modules and is used to list output ports on the IJTAG instrument that must be force high during the tracing of the current output port. The listed ports are simulated high and appended to the stable_after_setup simulation condition when tracing the associated output port. For example, tdoEnable typically needs to be forced active when tracing the tdo port of a TAP controller.
freq_multiplier	A positive integer value attribute at the icl_port. The value is computed from the optional FreqMultiplier keyword of the ICL ToClockPort statement. The default value is 1.
freq_divider	A positive integer value attribute at the icl_port. The value is computed from the optional FreqDivider keyword of the ICL ToClockPort statement. The default value is 1.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

function_modifier	A string that modifies the functionality of the port. Depending on the type of the port, different function modifiers can be applied. • CaptureEnPort and ToCaptureEnPort: The function modifier can have the value “CaptureShiftClock” or “CaptureShiftClockInv”. When one of these function_modifiers is specified on a CaptureEnPort, the port can only trace to ToCaptureEnablePort or a CaptureEnPort with one of the two function modifiers as well. When generating patterns for an ICL module and this module has a CaptureEnPort with function_modifier equal to CaptureShiftClock, the port is driven by a gated version of TCK that is only active during the capture and shift cycles. When generating patterns for an ICL module and this module has a CaptureEnPort with function_modifier equal to CaptureShiftClockInv, the port is driven by a gated version of inverted TCK that is only active during the capture and shift cycles. • CaptureEnPort and ToCaptureEnPort: The function modifier can have the value “capture_shift_enable”. This indicates an active-high control signal which is active during the Capture-DR, Capture-IR, Shift-DR and Shift-IR states of the TAP controller. • CaptureEnPort, ToCaptureEnPort, ShiftEnPort, ToShiftEnPort, UpdateEnPort and ToUpdateEnPort: The function modifier can have the value “ir” or “ir_dr” to indicate that the controlled scan interface is active during IR scan loads only or during both IR and DR scan loads. Without a function modifier the scan interface can be active during DR scan loads only. The specified capture, shift and update control ports of a scan interface must have the same function modifier. • CaptureEnPort, ToCaptureEnPort, ShiftEnPort and ToShiftEnPort: The function modifier can have the value “retimed_at_destination” to indicate that the corresponding control signal transitions must occur one TCK cycle earlier. This is necessary for generating patterns at the core level when the retiming elements for capture- or shift-enable have been moved from the TAP to each destination node. The function modifier must be applied consistently across the source and sink of a connection. On a ToCaptureEnPort and ToShiftEnPort the function modifier documents the lack of retiming elements on the controller side.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

function_modifier (continued)	• ClockPort and ToClockPort: The function modifier can have the value “sync_tester_clock” to indicate that the clock port is in sync with the tester clock. The function modifier can have the value “two_pin_serial_clock_port” to indicate that this port clocks the TwoPinSerialPort (TPSP) controller. The tool considers the clock as synchronous to the tester clock and uses it to generate the TCK clock of the IJTAG network. The frequency ratio between the TPSP clock and the generated TCK is 3:1. This function modifier is reserved exclusively for this usage. • DataInOutPort: The function modifier can have the value “two_pin_serial_inout_port” to indicate that this port transfers data in and out of the network through the TwoPinSerialPort controller. This function is reserved exclusively for this usage. • DataInPort and DataOutPort: The function modifier can have the values “tdo_enable_active_high” or “tdo_enable_active_low” to indicate an active high or low polarity of the corresponding tdo_en control signal. The function modifier can have the value “tap_fsm_state” that indicates the content of the 4-bit register that stores the state of the TAP controller’s finite state machine, or “tap_fsm_state_delayed” that is delayed by half a TCK cycle. This function modifier is required for the insertion of the Boundary Scan hardware. The SSN data path ports are also using this attribute in their ICL descriptions. The SSN function modifiers are “tessent_ssn_clock” and “tessent_ssn_bus_data”. Refer to “Understanding the ICL Modeling of the SSN Network” on page 4412 for a description of their usage. • UpdateEnPort and ToUpdateEnPort: The function modifier can have the value “update_clock”. The falling edge of this signal triggers the update functionality of a TDR directly. (Without this function modifier, the signal from the UpdateEnPort is combined with the falling edge of the TCK).
icl_extraction_port_trigger_list	String that contains a list of ports that must be present in the same ICL module. An ICL port with this attribute does not serve as a starting point for ICL extraction tracing from the very beginning. This port is used as starting point only if one of the ports listed in the attribute value has been reached by tracing from another instance or top module port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

ijtag_function	String that specifies the function of the icl_port; for example, “scan_in”. This is the ICL port function string converted to lowercase and the words connected with underscores.
index	Integer that specifies the index of the icl_port that is already part of the name when the icl_port is part of a bus.
is_bus	Boolean that specifies whether the icl_port is part of a bus. This attribute is set to true when the icl_port is part of a bus.
is_valid	Boolean that specifies whether the ICL port still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
launch_edge	String attribute that is defined on a port with function ScanOut to reflect the LaunchEdge of the ScanOutPort in ICL.
master_module_name	String that specifies the master_name of the icl_module object the icl_port belongs to.
module_name	String that specifies the name of the icl_module object the icl_port belongs to.
name	String that specifies the name of the icl_port including its bus index when it is an element of a bus.
object_type	Returns the object data type “icl_port”.
period	A string provides access to the "Period" specification of the ToClockPort in the ICL file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

read_latency	An attribute that can be specified inside a DataRegister element. If present, the IJTAG retargeting engine inserts the specified latency as a delay before the corresponding DataRegister is read through an iRead operation. The delay is inserted right before the operation that captures value of the DataRegister. When an operation captures the value of multiple DataRegisters with the read_latency attribute, the delays are merged into a single delay.Use one of the following values for the read_latency attribute: • tck <delay> : Indicates that the delay should last for <delay> cycles of the tester clock (TCK). • sck <icl_clock_port> <delay>: Indicates that the delay should last for <delay> cycles of the clock <icl_clock_port>. The clock has to be defined in the same module as the DataRegister with the read_latency attribute. The clock has to be previously traced with the iClock command. • time <time value>: Indicates that the delay should last for <time value>. The <time value> consists of a value and a time unit (s, ms, us, ns, fs or as), for example: 5us. When TCK is OFF, <delay> is converted to <delay>/tester_period TCK cycles rounded up to the next integer. If TCK is asynchronous free-running, TCK is ON and the <delay>/ tester_period must be a multiple of the TCK ratio.
scan_interface_list	Lists all ScanInterface names that the port belongs to.
tessent_bisr_function	String that reflects the function of the ICL port for the Tessent Memory Repair ICL module. Legal values for this attribute are: Reset, MemoryDisable, Select.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

tessent_clock_domain_labels	An attribute that reports the labels of clocks added before ICL extraction. The value is a list of design pin and port labels. Tessent adds the attribute to ICL port objects that represent asynchronous clocks, reference clocks, and branch clocks.
	
	Tessent populates the design pin and port labels in this attribute with the pre-synthesis names, and stores the potential post-synthesis names in the tessent_design_rtl_gate_port_mapping attribute. When the ICL models are loaded in no RTL mode after logic synthesis, the attribute tessent_design_rtl_gate_port_mapping is used to find and stores the new equivalent names.
	
	The tessent_clock_domain_labels attribute is also added to ICL instances to reference the equivalent design instance in both RTL mode and no RTL mode.
	
	Example:
	
	`ClockPort data_in[35] {`
	`  Attribute tessent_design_gate_ports = "{data_in[35]}";`
	`  Attribute tessent_design_rtl_ports="{data_in[2][1].ck[1]}";`
	`  Attribute tessent_clock_domain_labels = \`
	`  "ck0 {data_in[2][1].ck[1]}";`
	`  Attribute tessent_clock_periods = "all 10.00ns";`
	`}`
	
	`Instance add_clock_inst_0 of design1_add_clock_0 {`
	`  InputPort ClockIn = clka;`
	`  Attribute tessent_clock_domain_labels =`
	`  "clka_branch_label {clka_inst/INT.clk}";`
	`}`
	
	`attribute tessent_design_rtl_gate_port_mapping = \`
	`  "{clka_inst/INT.clk} {clka_inst/\INT.clk }" \`
	`  "{data_in[2][1].ck[1]} {data_in[35]}";`
	`…`
	`}`

tessent_clock_periods	A string attribute that is set on ports of the current ICL design during ICL extraction. It reports the period that was set on the clock port with the “add_clocks -period” command prior to running ICL extraction. The value of this attribute is the “all” keyword followed by the period. The “all” keyword is used to reflect the fact that the period is used for all test modes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

tessent_common_tdr_source	An attribute that can be defined in any DataInPort wrapper inside the ICL Module wrapper to specify an arbitrary common source name for the DataInPorts. During the execution of the create_dft_specification command, one Tdr is created with one DataOutPort for each unique source name such that all ports with the same common source name can be controlled by a single IJTAG signal. If "tessent_enable_group" is also specified, a separate Tdr is created for each unique group.
tessent_default_procedure_name	An attribute that you can define inside a ToClockPort element of an ICL module to describe the name of an iProc that should be called by default to initialize the PLL or clock divider modeled by the ICL module. As you can see in Example 2 of the ProcedureStep section, the create_icl_verification_patterns command automatically detects that the memory BIST controller makes use of a clock that goes through the ToClockPort and automatically infers a ProcedureStep wrapper to initialize the PLL or clock divider associated to the ToClockPort. The ProcedureStep includes the iProcArguments wrapper if you also defined the tessent_default_procedure_parameters attribute.
tessent_default_procedure_parameters	An attribute that can be defined inside a ToClockPort element of an ICL module to describe the arguments that should be passed to the proc referenced by the tessent_default_procedure_name attribute
tessent_design_gate_ports	An attribute populated by the extract_icl command when the tool extracts the top-level ICL module. The attribute identifies the post-synthesis name of the design port corresponding to the ICL port. It identifies the design object in a quick synthesis view, and, in no RTL mode, to map the ICL port to a design port in the post-synthesis netlist. Example (using escaped identifiers): • Post-synthesis name: \data#[5]in [35] • Pre-synthesis name: \data#[5]in [2][1].a • Root bundle name: \data#[5]in • ICL port name: data__5_in[35] • tessent_design_rtl_ports: \data#[5]in [2][1].a • tessent_design_gate_ports: \data#[5]in [35]
tessent_design_not_found	A boolean attribute that is set to true when the ICL data model is updated to reflect the netlist, and neither the tessent_design_gate_ports nor the tessent_design_rtl_ports attribute on the ICL port can be matched to a port in the netlist.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

tessent_design_rtl_ports	An attribute that specifies the pre_synthesis_name of the design port corresponding to the ICL port at the time of ICL extraction. This attribute identifies the design object in an RTL view, and in no RTL mode, to map the ICL port to a design port in the post-synthesis netlist. For example: • Post-synthesis name: data_in[35] • Pre-synthesis name: data_in[2][1].a • Root bundle name: data_in • ICL port name: data_in[35] • tessent_design_rtl_ports: data_in[2][1].a
tessent_dft_function	A string attribute used to identify ICL ports having special meaning during DFT.
tessent_dft_signal_default_value_in_all_test	String attribute that defines the value the DFT signal is to have in all test modes. Its value is “1”, or “0”. See the description of the register_static_dft_signal_names command for more information. This attribute is mandatory when the tessent_dft_signal_usage attribute is set to global_dft_control and illegal otherwise.
tessent_dft_signal_name	A string attribute that defines the name of the DFT signal the ICL port corresponds to. This attribute is set on ports of the TDR created by process_dft_specification when the add_dft_signals *dft_signal_name* -create_with_tdr command is used. It is set on port of the current design when the add_dft_signals *dft_signal_name* -source_node *port_name* -make_icl_port command is used. If you want to have the output ports of your own TDR be automatically identified as the source of DFT signals, you must identify them with this attribute and three more attributes described below. See Example 2 in the add_dft_signals command for a complete example.
tessent_dft_signal_reset_value	A string attribute that defines the value the DFT signal is to have in functional mode. Its value is “1” or “0”. See the description of the register_static_dft_signal_names command for more information. This attribute is mandatory when the tessent_dft_signal_name attribute is set on an ICL port.
tessent_dft_signal_scan_mode_type	A string attribute that defines the scan mode type of the DFT signal when the tessent_dft_signal_usage is set to scan_mode. See the description of the register_static_dft_signal_names command for more information. This attribute is mandatory when the tessent_dft_signal_usage attribute is set to scan_mode and illegal otherwise.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

tessent_dft_signal_usage	A string attribute that defines the usage of the DFT signal. Its value is global_dft_control, logic_test_control and scan_mode. See the description of the register_static_dft_signal_names command for the meaning of each usage. This attribute is mandatory when the tessent_dft_signal_name attribute is set on an ICL port.
tessent_dft_signal_value_in_pre_scan_drc	A string attribute that defines the value the DFT signal is to be forced to during pre-DFT/pre-Scan DRC. Its value is “1”, “0”, or “x”. See the description of the register_static_dft_signal_names command for more information. This attribute is mandatory when the tessent_dft_signal_usage attribute is set to global_dft_control or logic_test_control.
tessent_enable_group	An attribute that can be defined in any DataInPort wrapper inside the ICL Module wrapper to specify arbitrary group names for the DataInPorts. During the execution of the create_dft_specification command, one Tdr is created for each group of DataInPorts such that the group of ports can be controlled by IJTAG while leaving the ports in the other groups under functional control.
tessent_ignore_during_e14_drc (part 1 of 2)	This attribute controls the handling of ICL ports during the E14 check. If the test_setup procedure contains iCall commands, E14 also verifies the effect of the processed iRead and iWrite commands on the design. This verification can be suppressed by means of the “tessent_ignore_during_e14_drc” attribute. The attribute value can be specified as a sized or unsized binary number using legal ICL digits 0, 1, and X. If the attribute value is specified as a sized number, the size must match the size of the object. If the attribute value is specified as an unsized number, it is padded with 0s or Xs if necessary, dependent on the leftmost specified bit. The right-most bit (the least significant bit) of the attribute value refers to the right-most bit of the port, the second right-most bit of the value refers to the second right-most bit of the port, and so on. The value “1” instructs Tessent Shell to ignore the bit of the port during the E14 check, the values “0” and “X” represent the default behavior. **Note:** Do not attach this attribute to aliases or anything other than ICL ports or ICL ScanRegisters. ICL ports do not inherit attributes from associated aliases.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

tessent_ignore_during_e14_drc (part 2 of 2)	iRead commands in test_setup/test_end iCalls: If you specify an iRead command in one of the iProcs which are called in the test_setup/test_end procedure, then the specified iRead is retargeted to the top level. The value at the specified location can either be observed through a DataOutPort or it can be captured into a ScanRegister. In the latter case, the content of the ScanRegister is shifted out through a ScanOutPort. If the observed value does not match the value specified by means of the iRead command, you get an E14 violation. In both cases, the generic E14 violation refers to the top level port (DataOutPort or ScanOutPort), while the additional explanation gives hints about the original iRead target and/or the involved ScanRegister.The "compare" events originating from user-defined iRead commands can only be suppressed if the iRead target is an ICL pin/ICL port with the tessent_ignore_during_e14_drc attribute. It cannot be suppressed if the iRead target is a scan register or a data register. iWrite verification: Tessent IJTAG creates internal "compare" events to verify that the iWrite commands succeeded, that is, to verify that the specified value actually arrives at the design pin that is associated with the iWrite target. This works only if the iWrite target is an ICL pin/ICL port. It does not work with iWrite commands that address ICL ScanRegisters or ICL DataRegisters, because ICL registers cannot be mapped to design FFs. The "iWrite verification" can be suppressed by means of the attribute tessent_ignore_during_e14_drc.
tessent_ignore_during_icl_verification	A string attribute that specifies to include the ICL port in ICL verification patterns or not. The attribute has two values: true and false. True means include the ICL port in the ICL verification patterns. False means exclude it. Refer to the ICLNetworkVerify wrapper or the create_icl_verification_patterns command for more information about verification patterns.
tessent_memory_bist_function	String that reflects the function of the ICL port for Tessent MemoryBIST's ICL module of the memory module.
tessent_no_input_constraints	An attribute that affects only ICL ports of type DataInPort. If this attribute is not set on an ICL port with port function DataInPort or if it is set to "off", then the port is automatically constrained to C0 or C1 at the end of the test_setup procedure, according to the last value that has been assigned to it during the IJTAG retargeting of the iCall and iMerge commands in the test_setup procedure. If the attribute is set to "on", this behavior is suppressed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

tessent_pipeline_enable	This string attribute can only be used on ICL ports of type ScanInPort or ScanOutPort. It denotes an ICL data signal (DataInPort, DataOutPort, a single bit of a ScanRegister, DataMux, LogicSignal, Alias, and so on) inside the ICL module in which the scan port (ScanInPort or ScanOutPort) resides, or a relative path to an ICL data signal in one of the instances below the ICL module in which the scan port (ScanInPort or ScanOutPort) resides. Inverted ICL data signals can be denoted by means of a preceding tilde. The data signal is evaluated at the time when a scan load through the scan port (ScanInPort or ScanOutPort) happens. If the result is a logical "1"or if the tessent_pipeline_enable attribute is omitted, the pipeline stages specified by means of the tessent_pipeline_stages attribute are inserted into the scan path at the position of the scan port (ScanInPort or ScanOutPort). If the tessent_pipeline_enable attribute exists and the specified data signal evaluates to "0", the pipeline stages are not inserted. If the scan load through the scan port (ScanInPort or ScanOutPort) modifies the value of the data signal which the attribute refers to, the resulting behavior is undefined.
tessent_pipeline_stages	This integer attribute can only be used on ICL ports of type ScanInPort or ScanOutPort. It denotes the number of pipeline stages to be inserted at the position of the scan port (ScanInPort or ScanOutPort) during a scan load through this port. You can use this to deal with certain timing requirements that, otherwise, could not be fulfilled. These pipeline stages do not affect the sequence of values applied to the top ScanInPort or the sequence of values observed at the top ScanOutPort (the tool does not scan in or scan out values for those pipeline flops). The operation of the scan registers (and TDO enable signals) downstream of such a pipeline is assumed to be delayed accordingly (by means of pipeline stages in the control lines), such that the scan load values can be applied unaltered. The observation at the top ScanOutPort is delayed by the sum of pipeline stages on the active scan path, Tessent Shell adapts the pattern files accordingly. If required, the tool adds additional cycles to enable the scan load to complete. The pipelining behavior can be made switchable by means of the tessent_pipeline_enable attribute.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

tessent_reset_value	An attribute that you can define in any DataInPort wrapper inside the ICL Module wrapper to specify an expected value after a reset. During create_dft_specification, the TDRs added to control these DataInPorts have their reset value set to match the expected value requirement. You specify this attribute as a sized binary number composed of values “0”, “1”, or “X”, where “X” indicates the DataInPort does not have a specific reset value requirement.
tessent_ssn_clock_pulse_count	An attribute that is used to describe the number of clock pulses an SSN bus clock input is supposed to have. This number is computed using the requirement to have a consistent stream of data through the SSN datapath with all of its bus frequency dividers, bus frequency multipliers, pipelines, and scan hosts. Calculation of this attribute uses the assumption that all datapaths are in their default configurations.
tessent_ssn_clocks	An attribute that is used to describe the SSN bus clock port associated with a bus data in and bus data out port. The attribute is currently rule checked to only have one clock port name.
tessent_ssn_data_phases	An attribute that is used to describe the strobing phase of the destinations of each bus data in ports and the launch phase of the source of each bus data out ports. The phase is expressed with two integers. For example, {2 2} on a data in port means the destinations of the port strobes the data on the second edge every two edges. {1 4} on a data out port means the source launches the data on the first edge every four edges. Finally, {0.5 1} means the data is strobed on every falling edge of the clock. Each port can only have a single pair of integers.
tessent_ssn_datapath_bus_index	An attribute that is used to describe the ordering of the bus data bits when the bus is not composed of a single contiguous bus. For example, the attribute is specified if the DataIn ports are a list of scalars or more than one buses. The value of the attribute is a different integer for each bit of the input and output bus of each data path. Those integer are sorted from high to low when connected to the data path at the next level up using the DesignInstance wrapper. See “DesignInstance” on page 4441 for more information about this attribute.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

tessent_ssn_datapath_ids	An attribute used to describe the grouping of ports belonging to a common SSN data path. The SSN ports with the function_modifier attribute set to “tessent_ssn_bus_data” always only have one data path ID in the list as they can only be associated to one data path. The ports with the function_modifier attribute set to “tessent_ssn_bus_clock” can be associated to more than one SSN data path ID. See “DesignInstance” on page 4441 for more information about this attribute.
tessent_timing	An attribute that you can define in any DataInPort wrapper inside the ICL Module wrapper to specify the output timing a Tessent-inserted TDR requires to control the DataInPort. During create_dft_specification, the TDRs added to control these DataInPorts match the requested timing. You can specify the values “unlatched”, “normal”, and “scan_reconfiguration” that create_dft_specification matches to output_timing values of “unlatched”, “normal”, and “ijtag_scan_selection” respectively.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-11. Built-In Attributes on ICL Port Object Type (cont.)

tessent_use_in_dft_specification	String attribute that you can set to false on an ICL Port to force the create_dft_specification command to ignore this DataIn or DataOut port. You can add the attribute in the ICL source description or set using the set_attribute_value command. The legal values for this attribute are: auto, false, and auto_no_pi. When set or left to default to auto, the DataInPort is driven by a TDR DataOutPort if it is not seen to be already sourced by a DataOutPort or a primary input of the current design. The DataOutPorts are observed by a TDR DataInPort if it is not seen to be already observed by a DataInPort or a Primary output of the current design. See the description of the tessent_enable_group and tessent_common_tdr_source attributes above for a method to control the TDR assignment. The complete algorithm on how this works is described in the create_dft_specification command description. To have create_dft_specification ignore a complete ICL instance, use the "set_ijtag_instance_options -use_in_dft_specification off" command. When set to "auto_no_pi", the DataInPort is equipped with a local IJTAG control even if it is seen to already be sourced by a primary input. You want to use the "auto_no_pi" value for ports like the reset port of a PLL because they are often sourced by a system level reset port. Even though it is possible to toggle the system reset to reset the PLL, the effect of the reset pulse is felt by the entire system and you may affect already running parts of the circuit. When you provide local controllability, it becomes possible to reset the PLL without affecting the rest of the chip. See the description of create_dft_specification for more information about how DataInPorts are handled during the creation of the DFT specification.
write_latency	An attribute that can be specified inside a DataRegister element. If present, the IJTAG retargeting engine inserts the specified latency as a delay after each iApply that contained an iWrite operation to the DataRegister. When the iApply contains write operations to multiple DataRegisters with the write_latency attribute, the delays are merged into a single delay. For details on the possible values of the write_latency attribute, see the documentation of the values for the read_latency attribute.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Inheritance

Built-in attribute inheritance from associated icl_module objects:

- None

User-defined attributes are inherited. See "Attribute Inheritance Behavior" on page 3840.

Related Commands

get_attribute_list	get_iclock_option
get_attribute_value_list	get_name_list
get_icl_fanins	report_attributes
get_icl_fanouts	report_iclock
get_icl_ports	report_ijtag_logical_connections
get_iclock_list	

Port Specification

An ICL port object is an input, output, or inout interface of an ICL module, which is specified by a string. Examples of ICL port names are: "In1", "ClkA", and "Out3".

Related Topics

get_attribute_list

get_attribute_value_list

get_icl_fanins

get_icl_fanouts

get_icl_ports

get_iclock_list

get_iclock_option

get_name_list

report_attributes

report_iclock

report_ijtag_logical_connections

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL File Examples With Complex Ports

The following examples illustrate the correspondence between SystemVerilog files that include complex ports and the ICL files that are created by ICL extraction.

Example of Complex Ports as Functional Clocks or Input Port Functions 3806
Example of Grouping ICL Port Definitions for Modeling. 3808
ICL Model Attribute Name Examples. 3811

Example of Complex Ports as Functional Clocks or Input Port Functions

The following example illustrates how Tessent manages input port functions defined as complex ports during ICL extraction. It manages functional clocks defined as complex ports in the same way. The example also illustrates the use of the SystemVerilog interface data structure at the end point.

ICL extraction computes valid ICL names for the ICL ports from the post-synthesis names of the design ports. The tessent_design_gate_ports and tessent_design_rtl_ports attributes help to bind each ICL port to the equivalent design port object.

Tessent escapes post-synthesis names for SystemVerilog interface sub-bundle objects during ICL extraction. Because the ICL model does not accept escaped names, Tessent deletes the escape character (\) and replaces the dot character (.) with an underscore character (_) to create the new ICL names.

In a SystemVerilog interface port, you can specify multiple directions for each sub-bundle object. As a consequence, Tessent blasts these into several ports during synthesis. Further, at ICL extraction, such a port and the associated pin are blasted into several ICL ports and ICL pins in the ICL data model.

This example uses a simple design without memory BIST or other features. The input functional clocks in the example are grouped in the same structure, named Clock_t.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

top.sv

```
interface JtagInterface ();
    logic tck,reset,select,shift_en,capture_en,update_en,scan_in,scan_out;
    modport host (output tck, reset, select, shift_en, capture_en,
    update_en, scan_in, input scan_out);
    modport client (input tck, reset, select, shift_en, capture_en,
    update_en, scan_in, output scan_out);
endinterface

interface FunctionalInterface ();
    typedef struct packed {
       logic [1:0] MemClock;
       logic [1:0] FuncClock;
    } Clock_t;
    Clock_t ClockData;
    logic FuncRstSync, FuncRstAsync;
    logic [15:0] BusIn;
    logic [15:0] BusOut;
    modport Clocking (input ClockData, FuncRstSync, FuncRstAsync);
    modport Data (input BusIn, output BusOut);
endinterface

module top (FunctionalInterface.Clocking clockif, JtagInterface.host jif);
  ...
endmodule
```

top.icl

```
Module top {
   // Created by ICL extraction
   ClockPort clockif_ClockData[3] {
      Attribute tessent_design_gate_ports = "{\clockif.ClockData [3]}";
      Attribute tessent_design_rtl_ports =
"{clockif.ClockData.MemClock[1]}";
   }
   ClockPort clockif_ClockData[2] {
      Attribute tessent_design_gate_ports = "{\clockif.ClockData [2]}";
      Attribute tessent_design_rtl_ports =
"{clockif.ClockData.MemClock[0]}";
   }
   CaptureEnPort jif_capture_en {
      Attribute tessent_design_gate_ports = "{\jif.capture_en }";
      Attribute tessent_design_rtl_ports = "{jif.capture_en}";
   }
   ResetPort jif_reset {
      ActivePolarity 0;
      Attribute tessent_design_gate_ports = "{\jif.reset }";
      Attribute tessent_design_rtl_ports = "{jif.reset}";
   }
   ScanInPort jif_scan_in {
      Attribute tessent_design_gate_ports = "{\jif.scan_in }";
      Attribute tessent_design_rtl_ports = "{jif.scan_in}";
   }
   ...
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example of Grouping ICL Port Definitions for Modeling

ICL extraction creates ICL models in bit-blasted mode with all ICL ports represented bit-by-bit. If you do not want bit-blasting, you can write your own ICL models by grouping the ICL port definitions as shown in the following example.

corea.sv

```
typedef enum logic [1:0] {A, B, C, D} enum_t;
typedef struct packed {
  enum_t [1:0] state;
  logic ack;
} data_t;

interface MSBus (input logic clk);
  data_t data;
  modport Slave (input data);
  modport Master (output data);
endinterface

module coreb (input data_t data_in, MSBus.Slave OneBus, MSBus.Master
OtherBus);
endmodule

module corea (input logic clk, MSBus.Slave MB1, MSBus.Master MB2, input
data_t [1:0] data_in);
  MSBus ConnOneBus (clk);
  coreb ucoreb_1(.data_in(data_in[1]), .OneBus(MB1),
.OtherBus(ConnOneBus));
  coreb ucoreb_2(.data_in(data_in[0]), .OneBus(ConnOneBus),
.OtherBus(MB2));
endmodule
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

corea.icl

```
Module corea {
   // Created by ICL extraction
   ClockPort MB1_clk {
      Attribute tessent_clock_domain_labels = "MB1_clk \MB1.clk ";
      Attribute tessent_clock_periods = "all 10.00ns";
      Attribute tessent_design_rtl_ports = "MB1.clk";
      Attribute tessent_design_gate_ports = "\MB1.clk ";
   }
   ClockPort MB2_clk {
      Attribute tessent_clock_domain_labels = "MB2_clk \MB2.clk ";
      Attribute tessent_clock_periods = "all 13.20ns";
      Attribute tessent_design_rtl_ports = "MB2.clk";
      Attribute tessent_design_gate_ports = "\MB2.clk ";
   }
   ClockPort clk {
      Attribute tessent_clock_domain_labels = "clk clk";
      Attribute tessent_clock_periods = "all 13.20ns";
   }
   DataInPort data_in[9:0] {
      Attribute connection_rule_option = "allowed_no_source";
      Attribute tessent_design_rtl_ports =
          {{data_in[1].state[1][1]}
          {data_in[1].state[1][0]}
          {data_in[1].state[0][1]}
          {data_in[1].state[0][0]}
          {data_in[1].ack}
          {data_in[0].state[1][1]}
          {data_in[0].state[1][0]}
          {data_in[0].state[0][1]}
          {data_in[0].state[0][0]}
          {data_in[0].ack}};
   }
   DataInPort MB1_data[4:0] {
      Attribute connection_rule_option = "allowed_no_source";
      Attribute tessent_design_rtl_ports =
          {{MB1.data.state[1][1]}
          {MB1.data.state[1][0]}
          {MB1.data.state[0][1]}
          {MB1.data.state[0][0]}
          {MB1.data.ack}};
      Attribute tessent_design_gate_ports =
          {{\MB1.data [4]}
          {\MB1.data [3]}
          {\MB1.data [2]}
          {\MB1.data [1]}
          {\MB1.data [0]}};

   }
   DataOutPort MB2_data[4:0] {
      Attribute tessent_design_rtl_ports =
          {{MB2.data.state[1][1]}
          {MB2.data.state[1][0]}
          {MB2.data.state[0][1]}
          {MB2.data.state[0][0]}
          {MB2.data.ack}};
      Attribute tessent_design_gate_ports =
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
            {{\MB2.data [4]}
            {\MB2.data [3]}
            {\MB2.data [2]}
            {\MB2.data [1]}
            {\MB2.data [0]}};
      }
      Attribute test_setup_procfile = "";
      Attribute icl_extraction_date = "Tue Jul  4 10:10:48 PDT 2017";
      Attribute created_by_tessent_icl_extract = "true";
      Attribute tessent_design_id = "rtl1";
      Attribute tessent_design_level = "physical_block";
      Attribute tessent_is_physical_module = "true";

      Instance ucoreb_1 Of coreb {
         InputPort data_in[4] = data_in[9];
         InputPort data_in[3] = data_in[8];
         InputPort data_in[2] = data_in[7];
         InputPort data_in[1] = data_in[6];
         InputPort data_in[0] = data_in[5];
         InputPort OneBus_clk = MB1_clk;
         InputPort OneBus_data[4] = MB1_data[4];
         InputPort OneBus_data[3] = MB1_data[3];
         InputPort OneBus_data[2] = MB1_data[2];
         InputPort OneBus_data[1] = MB1_data[1];
         InputPort OneBus_data[0] = MB1_data[0];
         InputPort OtherBus_clk = clk;
         OutputPort OtherBus_data[4] = ucoreb_2.OneBus_data[4];
         OutputPort OtherBus_data[3] = ucoreb_2.OneBus_data[3];
         OutputPort OtherBus_data[2] = ucoreb_2.OneBus_data[2];
         OutputPort OtherBus_data[1] = ucoreb_2.OneBus_data[1];
         OutputPort OtherBus_data[0] = ucoreb_2.OneBus_data[0];
         Attribute tessent_design_instance = "ucoreb_1";
      }
      Instance ucoreb_2 Of coreb {
         InputPort data_in[4] = data_in[4];
         InputPort data_in[3] = data_in[3];
         InputPort data_in[2] = data_in[2];
         InputPort data_in[1] = data_in[1];
         InputPort data_in[0] = data_in[0];
         InputPort OneBus_clk = clk;
         InputPort OneBus_data[4] = ucoreb_1.OtherBus_data[4];
         InputPort OneBus_data[3] = ucoreb_1.OtherBus_data[3];
         InputPort OneBus_data[2] = ucoreb_1.OtherBus_data[2];
         InputPort OneBus_data[1] = ucoreb_1.OtherBus_data[1];
         InputPort OneBus_data[0] = ucoreb_1.OtherBus_data[0];
         InputPort OtherBus_clk = MB2_clk;
         OutputPort OtherBus_data[4] = MB2_data[4];
         OutputPort OtherBus_data[3] = MB2_data[3];
         OutputPort OtherBus_data[2] = MB2_data[2];
         OutputPort OtherBus_data[1] = MB2_data[1];
         OutputPort OtherBus_data[0] = MB2_data[0];
         Attribute tessent_design_instance = "ucoreb_2";
      }
   }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL Model Attribute Name Examples

The following tables show examples of attribute names in several contexts.

See "Example of Grouping ICL Port Definitions for Modeling" on page 3808 for the SystemVerilog RTL and the extracted ICL for these names.

RTL Views (Pre-Synthesis)

Table 11-12. ICL Module corea (RTL View: 4 Ports; ICL Module: 6 Equivalent Ports)

Active Name	pre_synthesis_name	tessent_design_gate_ports	tessent_design_rtl_ports	ICL Port Name
clk	clk	clk	clk	clk
MB1.clk	MB1.clk	\MB1.clk	MB1.clk	MB1_clk
MB1.data	MB1.data	\MB1.data	MB1.data	MB1_data
MB2.clk	MB2.clk	\MB2.clk	MB2.clk	MB2_clk
MB2.data	MB2.data	\MB2.data	MB2.data	MB2_data
data_in	data_in	data_in	data_in	data_in

Table 11-13. ICL Module coreb (RTL View: 3 Ports; ICL Module: 5 Equivalent Ports)

Active Name	pre_synthesis_name	tessent_design_gate_ports	tessent_design_rtl_ports	ICL Port Name
data_in	data_in	data_in	data_in	data_in
OneBus.clk	OneBus.clk	\OneBus.clk	OneBus.clk	OneBus_clk
OneBus.data	OneBus.data	\OneBus.data	OneBus.data	OneBus_data
OtherBus.clk	OtherBus.clk	\OtherBus.clk	OtherBus.clk	OtherBus_clk
OtherBus.data	OtherBus.data	\OtherBus.data	OtherBus.data	OtherBus_data

Quick Synthesis Views

Table 11-14. ICL Module corea (RTL View: 4 Ports; ICL Module: 6 Equivalent Ports

Active Name	pre_synthesis_name	tessent_design_gate_ports	tessent_design_rtl_ports	ICL Port Name
clk	clk	clk	clk	clk
\MB1.clk	MB1.clk	\MB1.clk	MB1.clk	MB1_clk
\MB1.data	MB1.data	\MB1.data	MB1.data	MB1_data
\MB2.clk	MB2.clk	\MB2.clk	MB2.clk	MB2_clk

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-14. ICL Module corea (RTL View: 4 Ports; ICL Module: 6 Equivalent Ports (cont.)

Active Name	pre_synthesis_name	tessent_design_gate_ports	tessent_design_rtl_ports	ICL Port Name
\MB2.data	MB2.data	\MB2.data	MB2.data	MB2_data
data_in	data_in	data_in	data_in	data_in

Table 11-15. ICL Module coreb (RTL View: 3 Ports; ICL Module: 5 Equivalent Ports)

Active Name	pre_synthesis_name	tessent_design_gate_ports	tessent_design_rtl_ports	ICL Port Name
data_in	data_in	data_in	data_in	data_in
\OneBus.clk	OneBus.clk	\OneBus.clk	OneBus.clk	OneBus_clk
\OneBus.data	OneBus.data	\OneBus.data	OneBus.data	OneBus_data
\OtherBus.clk	OtherBus.clk	\OtherBus.clk	OtherBus.clk	OtherBus_clk
\OtherBus.data	OtherBus.data	\OtherBus.data	OtherBus.data	OtherBus_data

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Scan Data Model

This section describes the scan models used for Hierarchical Scan Insertion and their built-in attributes.

Scan Element Object Type 3813
Scan Chain Family Object Type. 3820
Scan Mode Object Type 3820

Scan Element Object Type

The Scan Element object type is used to represent all scan path objects making up the scan segments and the scan chains.

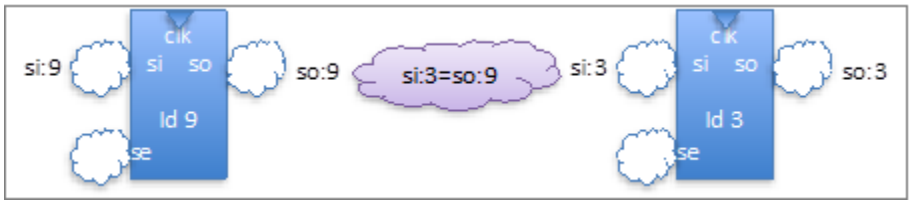
A scan element is either hierarchical (for example, scan chain, scan segment, shift register) or a leaf node, which models a single scan path of a cell or blackbox. The most common leaf scan element model is a single-bit scan cell, but multi-bit leaves like a flop-tray or a multi-chain cell can be modeled with one or more scan elements respectively. After DRC, there is one “active” scan element per scannable path of each cell, per declared scan segment and per scan chain. Most scan element leaves are populated from DRC results, but some get inferred by the planned insertion of certain types of modules like dedicated wrapper cells; these scan elements are deemed “virtual” as they do not exist yet but are assumed to exist for analysis and planning purposes.

The figure below is a simplified graphical representation of a scan_element. $id in si:$id and so:$id means that the group is taken from the id (node group) of the associated element. Other signals are classified into three different groups: at_speed_control, static_control, and clock. The scan element object holds a unique value for each of those group IDs.

The si:$id of one scan element connects to the si signal of its parent scan element when it is the first child element. It connects to the so of the previous element when it is not the first element. The so:$id of a hierarchical scan element connects to the so of its last child. The next figure

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

illustrates how the scan path equation “si:3 = so:9” is used to create a scan path between scan elements 9 and 3. Similarly, equations are used to control the scan enable of each scan element.



If specified, the si equation must include the signal type "si" as one of its inputs; similarly, the so equation must include the signal type "so" has one of its input. If specified, the clk equation must include the signal type "clk" or the operator <src> as one of its inputs.

Table 11-16. Built-in Attributes on Scan Element (SE) Object Type

Attribute name	Default	Description
class		standard, core, wrapper
clock_domains		The clock domain name or names of this element. This is the pseudo_port name or names if they exist.
cluster_name		String attribute assigned by the user. This attribute can be used to logically group scan elements. It is used by various commands to control scan insertion.
exists		Indicates if the scan element exists in the design netlist; a false value signifies that the element has not been implemented yet
has_children		Indicates if a hierarchical scan element has sub-scan elements defining it.
id		This is a unique integer that uniquely identifies a “scan_element” object. This id is only persistent within one run while in the analysis or insertion state.
is_non_scannable	False	Boolean that specifies that the scan element is non-scannable. Default is false.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-16. Built-in Attributes on Scan Element (SE) Object Type (cont.)

Attribute name	Default	Description
is_non_scannable_reason		A string that specifies why the scan_element is non-scannable, as follows: • s_rule_violation: The library cell has an unfixable s-rule violation. • non_scan_path_cell_in_hard_module: This attribute value is used for library cells that are inside a hard module that has only traced scan segments, and are not part of any traced segment. • clock_gater_cell: By default, memory elements inside a clock gater are not considered as scannable cells. However, you can still define a scan segment if the clock gater was implemented as a scannable cell. • add_nonscan_instance_control: The instance was marked as non-scannable as a result of using the "add_nonscan_instances -control" command in analysis mode. • segment_ignored_due_to_library_definition: The user-defined scan segment has been ignored because the internal scan paths that were defined in the Tessent shell library passed verification, so the native leaf_elements are used instead. • ignored_slice_due_to_segment_definition: You have defined a scan segment for one or more slices (scan paths) in the multi-slice library cell, but not for all of them. In this case, the tool uses the segment definitions and ignores the slices (scan paths) that do not have a corresponding segment definition. • driven_scan_enable: The scan_enable is connected to something that was not defined as a scan enable. • shadow_cell: This memory element was identified to be a shadow cell. It is loaded with a known value during scan shifting so there is no reason to convert it into a scan cell. Also, it only captures the value from the scan cell, so observability is also not an issue. • transparent_latch: This cell is a latch (or transparent latch), which makes it non-scannable.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-16. Built-in Attributes on Scan Element (SE) Object Type (cont.)

Attribute name	Default	Description
is_non_scannable_reason (continued)		• black_box: The memory element is a black box or is inside a black box. • inferred_during_system_mode_transition: This is used to mark cells that remain non-scannable due to previously unspecified conditions. It occurs very infrequently. • imported_from_tcd_scan: The object was inferred to be non-scannable because it has a Core/Scan TCD description and the property drc_from_boundary_only is equal to 1. • imported_from_icl: The instance was inferred to be non-scannable because it has a matched ICL description and the attribute keep_active_during_scan_test is not specified as false. • imported_from_bscan: The instance was inferred to be non-scannable because it is part of the boundary scan chain. • missing_scan_model: The instance was inferred to be non-scannable because it does not have a scan flop equivalent mapping. This only happens in the dft -scan context. • user_specified: The object was specified to be non-scannable using the add_nonscan_instances command. • imported_from_mbisr: The instance was inferred to be non-scannable because it is part of the memory bist controller. • multi_slice_cell_needs_mapping: The tool does not support mapping of multi-slice non-scan cells to scan cells. • segment_inside_segment_or_chain: This scan segment is part of a larger segment or chain so it is not directly usable. Gate pin-specific return values: • non_scan_path_element: This reason only applies to gate pins and is used to identify the memory elements (DFFs) that are not part of the scan path (and are thus not directly controllable and observable). • disabled_segment: This reason only applies to gate pins and is used to identify memory elements within a traced segment that fails DRC (for one of the previously mentioned reasons).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-16. Built-in Attributes on Scan Element (SE) Object Type (cont.)

Attribute name	Default	Description
length		This is the length of the scan element object (number of master sequential cells recursively found down the hierarchy).
mux_si_functional_source		Boolean indicating that the S6 rule check is skipped and any functional source on the scan-in of the scan element is multiplexed during scan chain insertion. This attribute is read-only.
mux_si_functional_source _reason		A string that specifies why mux_si_functional_source is set for a scan_element: • imported_from_tcd_scan — A *.tcd_scan* file specified the mux_si_functional_source property on this scan_element. • user_specified — The user-specified mux_si_functional_source on this scan element by setting the specified_mux_si_functional_source attribute. • If mux_si_functional_source is false, then mux_si_functional_source returns an empty string. This attribute is read-only
name		This is an optional name that was given when defining the “scan_element” object. This name is not necessarily unique and may be null.
no_si_pipelining		Boolean indicating that no pipelining element would get inserted at the beginning of a new scan chain if this element is the first element in the chain.
no_si_pipelining_reason		A string that specifies why no_si_pipelining is set for a scan_element, as follows: • imported_from_tcd_scan — A .tcd_scan file specified no_si_pipelining on this scan_element. • user_specified — The object specified no_si_pipelining by setting the specified_no_si_pipelining attribute. • If no_si_pipelining is false, then is_non-scannable_reason returns an empty string.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-16. Built-in Attributes on Scan Element (SE) Object Type (cont.)

Attribute name	Default	Description
power_domain_island		Returns the name of the power domain island associated with the scan_element. This attribute is inherited from its parent design object if a UPF or CPF file has been read in. Its value is computed using the design hierarchy and the power_domain_name attribute. For a definition of Power Domain Island, refer to the *Tessent Glossary*.
power_domain_name		The value of the power_domain_name attribute, if any. For a definition of Power Domain, refer to the *Tessent Glossary*.
shared_wrapper_cell_reason		A read-only string attribute. Only available on scan elements of class "wrapper" and state "usable". • direct — reached by PI fanout tracing or PO fanin tracing • feedback — feedback of the “direct” or user-specified shared wrappers • clock_gater_fanout — fanout of clock gater that is controlled with a primary input (see Clock Gaters and Wrapper Analysis in *Tessent Scan and ATPG User’s Manual*) • user_specified • inferred_from_clock_source — the instance is an OCC, the fast_clock port of the OCC is not directly sourced by a PI (that is, a PLL or clock divider), and the OCC drives at least one flop of type input or output wrapper. The wrapper_type is always generic when wrapper_type_reason is inferred_from_clock_source.
si_clock_domain		Clock domain for the first scan cell. This is the pseudo_port name if it exists.
si_clock_edge		Capture edge for the first scan cell.
so_clock_domain		Clock domain for the last scan cell. This is the pseudo_port name if it exists.
so_clock_edge		Capture edge for the last scan cell.
specified_mux_si_functional_source		A user attribute to set the combined attribute mux_si_functional_source.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-16. Built-in Attributes on Scan Element (SE) Object Type (cont.)

Attribute name	Default	Description
specified_no_si_pipelining		An attribute that prevents a pipelining element to get inserted at the beginning of a new scan chain if this element is the first element in the chain, even if the scan insertion options specifies -si_pipelining for that chain.
specified_wrapper_type	Off	An attribute that forces the wrapper type of a scan_element. Available values are: off, input, output, and generic. The default is off. • The result of the analyze_wrapper_cells command, which combines the specified_wrapper_type values with the inferred values, are reflected in the wrapper_type and shared_wrapper_cell_reason attributes described below. • When set or left to default to off, the wrapper_type value of each scan_element object is computed by the analyze_wrapper_cells command. • When you set the specified_wrapper_type attribute to input, output, or generic, the specification is kept during the analysis. • If you used the input value, the analyze_wrapper_cells command automatically finds all flip-flops fanning out to the scan_element you forced to input type and infer them as output wrapper type. • If you set the specified_wrapper_type attribute after wrapper analysis has been performed, the tool returns an error. Switching back to setup mode resets wrapper analysis and enables you to set the specified_wrapper_type attribute again.
state		ignored | unusable | unusable_child_mode | unresolved_child_mode | usable | usable_indirectly. Only “usable” scan elements can be specified as part of a population when defining a chain/segment using the “-include_...” option.
type		chain | segment | leaf_cell | shift_register | transparent_leaf_cell | inferred
used		Indicates if the scan element is used in at least one mode

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-16. Built-in Attributes on Scan Element (SE) Object Type (cont.)

Attribute name	Default	Description
wrapper_cell_capture_window_behavior		shift | hold | invert_hold | programmable_hold Read-only string attribute, only available on scan elements of class “wrapper” and state “usable”. See the description of set_wrapper_analysis_options -capture_window_behavior switch for details.
wrapper_cell_category		dedicated | shared Read-only string attribute, only available on scan elements of class “wrapper” and state “usable”.
wrapper_type		input | output | generic | ""

Scan Chain Family Object Type

The scan_chain_family is an intelligent container that controls the allocation of new scan chains from a specific population of scan element.

The command create_scan_chain_family is used to generate new chain families; the related commands delete_scan_chain_families and get_scan_chain_families are also available. The current introspection is currently very limited, but will eventually include the various scan insertion options specified for this particular population sample.

Table 11-17. Built in Attributes on Scan Chain Family Object Type

Attribute name	Default	Description
name		This is an optional name than was given when defining the “scan_chain_family” object. This name is not necessarily unique and may be null.
id		This is a unique integer that uniquely identifies a “scan_chain_family” object. This id is only persistent within one run while in the analysis or insertion state.

Scan Mode Object Type

The scan_mode object controls the allocation of scan chains for a specific scan mode.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The hierarchical scan engine lets you specify any number of scan modes. Typically, a mode applies to an entire scan element population or a large subset (for example, wrapper elements), and is populated by including scan_element objects directly and scan_chain_family objects.

Table 11-18. Built-in Attributes on Scan Mode Object Type

Attribute Name	Default	Description
name		This is an optional name than was given when defining the "scan_mode" object. This name is not necessarily unique and may be null.
id		This is a unique integer that uniquely identifies a "scan_mode" object. This id is only persistent within one run while in the analysis or insertion state.

Test Point Data Model

This section describes the Test Point data model and its built-in attributes.

Test points are inserted into the design to increase controllability and observability of otherwise hard to reach locations. For more information, see "What are Test Points?" in the *Tessent Scan and ATPG User's Manual.*

Table 11-19. Built-In Attributes on Test Point Object Type

Attribute name	Default	Description
name		A unique name that was given when defining the "test point" object.
id		A unique integer that uniquely identifies a "test_point" object. This id is only persistent within one run while in the analysis or insertion state.
location		A string representing the full design path to the observed/controlled gate.
shared_group_id		A string representing a population of test points that can potentially share the same driver flop.
shared_id		An integer indicating the test points that are driven by the same flop. This integer is unique per sharing group.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 11-19. Built-In Attributes on Test Point Object Type (cont.)

Attribute name	Default	Description
type		A string that specifies the type of test point. The possible values are: • control point • observe point
control_point_gate_type		A string that specifies the type of control point. The possible values are: • AND Control Point • OR Control Point
origin		A string that specifies how the test point was created. The possible values are: • user_specified — created by the user or imported from an external tool/script • design_analysis — created by the tool with the analyze_test_points command
power_domain_name		A string representing the power domain name of the test point location.
power_domain_island		A string representing the power domain island in which the test point is placed.
clock_connection_name		A string representing the full design path to the clock that is driving the test point flop.
enable_connection_name		A string representing the full design path to the connection that enables the test point.
flop_location		A string representing the full design path to the observed or controlled flop instance.
clock_domain		The clock domain name of the test point's flop. This is the pseudo_port name if it exists.
control_point_target		A string that specifies whether the control points is on the clock_path or data_path.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Configuration Data Model

This section describes the Configuration data model and its built-in attributes.

config_csv_wrapper 3823
config_data_wrapper 3824
config_property 3825
config_repeatable_property 3825
config_wrapper 3826

config_csv_wrapper

This section describes the built-in attributes for the config_csv_wrapper object type.

You use this metadata syntax to define a config_csv_wrapper in the CsvWrapper that contains comma separated value lists.

Built-In Attributes

expanded_tree_node	If you use the add_config_tab command to display a configuration data tree in the GUI, then you can open the nodes as if you clicked the [+] sign in the GUI. If you click the [+] in the GUI and you query the value of this attribute, it reflects that the node is open with a true value. If you save the configuration data to file using write_config_data, you see a +{ for all wrappers that were open at the time you issue the command. Reading this configuration data back using read_config_data restores the value of this attribute where the wrapper starting with +{ has this attribute true and those open with { have it false.
file_path_name	String attribute that specifies the pathname to the file where a given object was read using the read_config_data command. When the configuration is created in memory using commands such as add_config_element, the value of this attribute is null.
is_valid	Boolean that specifies whether the object still exists. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the leaf name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
line_number	String that specifies a line number in the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element, the value of this attribute is null.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

name	String that specifies the name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
object_type	Returns the object type of the configuration element, namely 'config_csv_wrapper' for this object type.

config_data_wrapper

This section describes the built-in attributes for the config_data_wrapper object.

You use this metadata syntax to define a config_data_wrapper in the DataWrapper that contains comma separated value lists.

Built-In Attributes

expanded_tree_node	If you use the add_config_tab command to display a configuration data tree in the GUI, then you can open the nodes as if you clicked the [+] sign in the GUI. If you click the [+] in the GUI and you query the value of this attribute, it reflects that the node is open with a true value. If you save the configuration data to file using write_config_data, you see a +{ for all wrappers that were open at the time you issue the command. Reading this configuration data back using read_config_data restores the value of this attribute where the wrapper starting with +{ has this attribute true and those open with { have it false.
file_path_name	String attribute that specifies the pathname to the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element, the value of this attribute is null.
is_valid	Boolean that specifies whether the object still exists. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the leaf name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
line_number	String that specifies a line number in the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element, the value of this attribute is null.
name	String that specifies the name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

object_type	Returns the object type of the configuration element, namely 'config_data_wrapper' for this object type.

config_property

This section describes the built-in attributes of the config_property object.

You use this metadata syntax to define a config_property wrapper in the Property wrapper that contains comma separated value lists.

Built-In Attributes

file_path_name	String attribute that specifies the pathname to the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element, the value of this attribute is null.
is_valid	Boolean that specifies whether the object still exists. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the leaf name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
line_number	String that specifies a line number in the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element, the value of this attribute is null.
name	String that specifies the name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
object_type	Returns the object type of the configuration element, namely 'config_property' for this object type.
value	String that returns the value of the property the same way get_config_value does. The value attribute is useful when you want to filter a collection of config property objects based on their current values.

config_repeatable_property

The section describes the built-in attributes for the config_repeatable_property object.

You use this metadata syntax to define a config_repeatable_property wrapper in the RepeatableProperty wrapper that contains comma separated value lists.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Built-In Attributes

file_path_name	String attribute that specifies the pathname to the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element, the value of this attribute is null.
is_valid	Boolean that specifies whether the object still exists. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the leaf name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
line_number	String that specifies a line number in the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element, the value of this attribute is null.
name	String that specifies the name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
object_type	Returns the object type of the configuration element, namely 'config_repeatable_property' for this object type.
value	String that returns the value of the property the same way get_config_value does. The value attribute is useful when you want to filter a collection of config property objects based on their current values.

config_wrapper

This section describes the built-in attributes for the config_wrapper object.

You use this metadata syntax to define a config_wrapper in the Wrapper that contains comma separated value lists.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Built-In Attributes

expanded_tree_node	If you use the add_config_tab command to display a configuration data tree in the GUI, then you can open the nodes as if you clicked the [+] sign in the GUI. If you click the [+] in the GUI and you query the value of this attribute, it reflects that the node is open with a true value. If you save the configuration data to file using write_config_data, you see a +{ for all wrappers that were open at the time you issue the command. Reading this configuration data back using read_config_data restores the value of this attribute where the wrapper starting with +{ has this attribute true and those open with { have it false.
file_path_name	String attribute that specifies the pathname to the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element, the value of this attribute is null.
is_valid	Boolean that specifies whether the object still exists. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the leaf name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
line_number	String that specifies a line number in the file where a given object was read from using the read_config_data command. When the configuration is created in memory using commands such as add_config_element, the value of this attribute is null.
name	String that specifies the name of the object. Refer to the get_config_value command for a precise description of what is the leaf name of a configuration object.
object_type	Returns the object type of the configuration element, namely 'config_wrapper' for this object type.
tree_node_state	String that specifies the tree node state. Possible values are as follows: • default • pass • maybe • fail

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 12
Attributes

Attributes are properties assigned to any object type. Attributes have names and values to associate relevant data with those objects. Attribute names may only include alphanumeric and underscore characters.

This section provides information on the following topics:

For detailed information on the attributes associated with each object type, see "Data Models" on page 3649.

Attribute-Related Commands **3829**
Attribute Types **3831**
Pre-Defined Attributes 3831
User-Defined Attributes 3831
Attribute Value Types **3831**
Attribute Filtering Equation Syntax **3832**
Glob and Regular Expression Pattern Matching Syntax **3834**
Hierarchical Name Matching With Escaped Identifiers **3838**
Attribute Inheritance Behavior **3840**
Inheritance Rules 3840

Attribute-Related Commands

You use the attribute-related Tessent Shell commands to create, manipulate, and query objects' attributes.

Table 12-1. Commands for Manipulating Attributes

Command	Description
get_attribute_list	Returns an alphabetically sorted Tcl list of attribute names.
get_attribute_option	Retrieves the current setting of an attribute's option.
get_attribute_value_list	Retrieves the value of an attribute on the specified design objects.
get_name_list	Returns the name(s) of the specified object(s).
register_attribute	Registers a new user-defined attribute.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 12-1. Commands for Manipulating Attributes (cont.)

Command	Description
report_attributes	Creates a human readable report for the set of attributes based on the usage options.
reset_attribute_value	Resets an attribute to its default value for the design objects specified in obj_spec.
set_attribute_options	Modifies options of any registered user-defined attribute.
set_attribute_value	Sets an attribute's value for the objects specified in object_spec.
unregister_attribute	Unregisters a user-defined attribute that was registered using the register_attribute command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Attribute Types

Tessent Shell supports two types of attributes: pre-defined attributes and user-defined attributes. Different object types have different pre-defined attributes. You can also create and specify user-defined attributes.

Pre-Defined Attributes 3831
User-Defined Attributes 3831

Pre-Defined Attributes

Pre-defined attributes exist on the object itself in the internal database. Because of this, pre-defined attributes do not need to be registered. Different object types have different pre-defined attributes. By default, these attributes are read-only.

Each data model has a list of pre-defined attributes that are described in sections "Hierarchical Design Data Model", "Flat Design Data Model", and "ICL Data Model".

User-Defined Attributes

User-defined attributes are registered with the register_attribute command before they can be used. The registration operation defines the attribute's value type and its default value and determines which design object type the attribute belongs to.

Attribute Value Types

Both pre-defined and user-defined attributes have an attribute value type.

Table 12-2. Attribute Value Types

Type	Description
Boolean	Case-insensitive. A Boolean. • Accepts any of the following for true: 1, true, yes, on. • Accepts any of the following for false: 0, false, no, off
Integer	Case-insensitive. An integer.
Double	Case-insensitive. Double-precision floating-point number.
String	Case-insensitive. A sequence of alphanumeric characters.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Attribute Filtering Equation Syntax

An attribute filtering equation is a series of relations joined together with the && (AND) and || (OR) operators. Parentheses are also supported. A basic relation compares or matches the value of an attribute with a constant expression through a relational operator.

Relational Operators

The relational operators are as follows:

==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
=~	Matches pattern
!~	Does not match pattern

Basic Relational Rules

The basic relational rules are as follows:

- Compare string attributes using any operator.
- Do not compare numeric attributes using pattern matching operators (=~ or !~).
- Compare Boolean attributes using only == and !=. The only possible value is true or false.
- Using an attribute without a relational operator in an equation is equivalent to saying att == true for a Boolean attribute or att != <defaultValue> for other types. You can also invert this default behavior by putting a ! in front of the attribute name. For example:

  ```
  is_something && !is_something_else
  ```

  translates to:

  ```
  is_something != <default> && is_something_else == <default>
  ```

  which mean the following when the attributes are of Boolean type:

  ```
  is_something == true && is_something_else == false
  ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Pattern Matching in the Filtering Attribute Equation

By default, the =~ and !~ filter operators use simple wildcard pattern matching (glob) with the "*" and "?" wildcards. When the -regexp option is specified for any command, the complete Posix Extended Regular Expression syntax is supported.

When you use the -regexp option, you should pay special attention to how you quote filter expressions. We recommend using rigid quoting with braces around regular expressions.

Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" to the beginning or end of the expression as necessary.

Refer to "Glob and Regular Expression Pattern Matching Syntax" for a complete description of the Tessent Shell pattern matching mechanism and a detailed description of the regular expression syntax.

Filter Equation Examples

This example shows different filtering attribute equations used to filter instance objects:

```
register_attribute -name Att1 -value_type boolean -object_types instance
register_attribute -name Att2 -value_type integer -default -1
register_attribute -name Att3 -value_type string -default ""

set_attribute_value u1 -name Att1
set_attribute_value u1 -name Att2 -value 3
set_attribute_value u1 -name Att3 -value "aba"

set_attribute_value u2 -name Att1
set_attribute_value u2 -name Att3 -value "abb"

#Matches only u1 because Att2 has default value on u2 so Att2 returns
#false
```

get_instance -filter {Att1 && Att2}

```
{u1}

#Matches both because Att3 starts with a in both
```

get_instance -filter {Att1 && Att3 =~ "a*"}

```
{u1 u2}

#Matches u2 because Att2 defaults to -1.
```

get_instances -filter {Att2 < 2}

```
{u2}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Glob and Regular Expression Pattern Matching Syntax

The introspection commands perform pattern matching in two distinct ways with subtle usage differences. The first way is with the *name_patterns* argument and it is specified as a list of patterns to match against object names. The second one is the use of the =~ and ~! operators within a filter attribute equation.

When specifying name_patterns, the hierarchy separator (/) is used to automatically split the matching patterns so that each pattern can be matched against the leaf name of each level of hierarchy separately. For example, the command "get_instance a*/b*" means to match the leaf instance starting with b in each instance whose name starts with a. Refer to section "Hierarchical Name Matching With Escaped Identifiers" for ways to deal with that in the presence of escape identifiers, especially when the escaped identifiers contains slashes.

When specifying the filter attribute equation, the / is treated as a literal within the pattern and the entire value of the attribute is searched by the entire pattern. For example, the command "get_instance -filter {name =~ "a*/b*"}" would match instance a1/b1 as well as a1/c1/b1 because a* matches "1/c1" as the / is just another literal character to match.

Tessent Shell supports two types of regular expression pattern matching:

- Perl
- POSIX Extended

Use the "-regex_style {extended | perl}" option of the set_tcl_shell_options command to select the style you want. The default is POSIX Extended.

In regular expressions syntax, most characters are treated as literals. That is, they match only themselves (for example, a matches "a"). The exceptions are called meta characters and are listed in Table 12-3.

Table 12-3. POSIX Extended Regular Expression Metacharacters

Meta Character	Description
.	The dot matches any single character. Within bracket expressions, the dot character matches a literal dot. For example: a.c matches "aac", "abc", and so on, but [a.c] matches only "a", ".", or "c".

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 12-3. POSIX Extended Regular Expression Metacharacters (cont.)

Meta Character	Description
[ ]	A bracket expression. Matches a single character that is contained within the brackets. For example, [abc] matches “a”, “b”, or “c”. [a-z] specifies a range that matches any lowercase letter from “a” to “z”. These forms can be mixed: [abcx-z] matches “a”, “b”, “c”, “x”, “y”, or “z”, as does [a-cx-z]. The - character is treated as a literal character if it is the last or the first character within the brackets: [abc-], [-abc]. Backslash escapes are not allowed. The] character can be included in a bracket expression if it is the first character: []abc]. The [character can appear anywhere.
[^]	Matches a single character that is not contained within the brackets. For example, [^abc] matches any character other than “a”, “b”, or “c”. [^a-z] matches any single character that is not a lowercase letter from “a” to “z”. As above, literal characters and ranges can be mixed.
^	Matches the starting position of the string. In Tessent Shell, all matching is rooted, which means the ^ character is inferred at the beginning of every expression when it is not already specified. For example, abc.* is equivalent to ^abc.* and both mean to match any string starting with abc.
$	Matches the ending position of the string. In Tessent Shell, all matching are rooted, which means that the $ character is inferred at the end of every expression when it is not already specified. For example, .*abc is equivalent to .*abc$ and both mean to match any string ending with abc.
( )	Defines a marked expression. The string matched within the parentheses can be recalled later (see next entry, \n). A marked expression can also be used to specify a sequence of characters that must be matched some number of times by following it by a *, + or {m.n} sequence. For example, (ab)* matches “”, “ab”, “abab”, “ababab”, and so on.
\n	Matches what the nth marked expression matched, where n is a digit from 1 to 9. For example (ab).*\1 matches any string that starts and ends with either a or b, such as aa, aca, aba, or bab but not acb and bca. A second example (ab)(cd).*\2\1 matches acffca but not acffaa because the second marked expression matched c and the first matched a.
*	Matches the preceding element zero or more times. For example: ab*c matches “ac”, “abc”, “abbbc”, and so on. [xyz]* matches “”, “x”, “y”, “z”, “zx”, “zyx”, “xyzzy”, and so on. (ab)* matches “”, “ab”, “abab”, “ababab”, and so on.
?	Matches the preceding element zero or one time. For example: ba? matches “b” or “ba”.
+	Matches the preceding element one or more times. For example: ba+ matches “ba”, “baa”, “baaa”, and so on.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 12-3. POSIX Extended Regular Expression Metacharacters (cont.)

Meta Character	Description
|	The choice operator matches either the expression before or the expression after the operator. For example: abc|def matches “abc” or “def”.
{m,n}	Matches the preceding element at least m and not more than n times. For example: a\{3,5\} matches only “aaa”, “aaaa”, and “aaaaa”. (ab){2,3} matches “abab” and “ababab”.

Table 12-4 lists the escaped characters that have special meaning in POSIX regular expression syntax:

Table 12-4. Special Characters in Regular Expressions

Escaped character	Description
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab

Table 12-5 lists the predefined special character sets that can be used in regular expressions:

Table 12-5. Special Characters Set in Regular Expression

Symbol	Content	Description
[:alnum:]	[A-Za-z0-9]	Alphanumeric characters
[:alpha:]	[A-Za-z]	Alphabetic characters
[:blank:]	[\t]	Space and tab
[:cntrl:]	[\x00-\x1F\x7F]	Control characters
[:digit:]	[0-9]	Digits
[:graph:]	[\x21-\x7E]	Visible characters
[:lower:]	[a-z]	Lowercase letters
[:print:]	[\x20-\x7E]	Visible characters and space character
[:punct:]	[\]\[!#$%&'()*+,./:;<=>?@\^_`{|}~-]	Punctuation characters
[:space:]	[\t\r\n\v\f]	Whitespace characters
[:upper:]	[A-Z]	Uppercase letters
[:xdigit:]	[A-Fa-f0-9]	Hexadecimal digits

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 1

The following list of examples illustrates the use of meta characters in matching expressions:

- .at matches any three-character string ending with “at”, including “hat”, “cat”, “bat”.
- [hc]at matches “hat” and “cat”.
- [^b]at matches all strings matched by .at except “bat”.
- ^[hc]at matches “hat” and “cat”, but only at the beginning of the string or line.
- [hc]at$ matches “hat” and “cat”, but only at the end of the string or line.
- \[.\] matches any single character surrounded by “[” and “]”, since the brackets are escaped. For example: “[a]” and “[b]”.
- [hc]+at matches “hat”, “cat”, “hhat”, “chat”, “hcat”, “ccchat”, and so on, but not “at”.
- [hc]?at matches “hat”, “cat”, and “at”.
- [hc]*at matches “hat”, “cat”, “hhat”, “chat”, “hcat”, “ccchat”, “at”, and so on.
- cat|dog matches “cat” or “dog”.
- [[:digit:]]+ matches “1”, and “32” but not “a1”.

Example 2

The following example matches all leaf instance names between 3 and 4 characters in length.

```
get_instances -filter { leaf_name =~ “[[:alnum:]_]{3,4}”} -regexp
```

```
{u1/u2/abc u1/u3/ab_1}
```

Example 3

The following example matches all instances having a leaf instance name with a prefix matching the value within a Tcl variable followed by 1 or many digits. In the example, the attribute equation cannot be enclosed in braces because it would prevent the Tcl shell from replacing the Tcl variable. Instead, the attribute equation is enclosed in double quotation marks and all special characters within the expression are escaped.

```
set prefix “MyString_”
get_instances -filter “leaf_name =~ \”${prefix}\[\[:digit:\]\]+\ “” -regexp
```

```
{u1/u3/MySting_1 u1/u2/MySting_75}
```

Example 4

The following example looks for all leaf instances with a prefix equal to abc inside all instances starting with u or U followed by one or more digits. Notice that the / is automatically inferred as a hierarchy separator causing the expression to be split in two and applied to each level of hierarchy separately.

```
get_instance {[Uu][[:digit:]]+/abc.*} -regexp
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
{u3/abcd U43/abc}
```

Hierarchical Name Matching With Escaped Identifiers

The slashes inside a name_patterns argument have special meaning; they are interpreted as hierarchy separators unless they are within escaped identifiers. For example, consider an instance whose name has two escaped identifiers as part of its hierarchical name: “u1/\u2/u3 /\u4/u5 /insta”. In Tessent Shell, all attribute, introspection, and editing commands can reference such a name with and without the escape characters and trailing spaces.

For more information, refer to “Glob and Regular Expression Pattern Matching Syntax” on page 3834.

Note

Legacy commands (commands that existed prior to v9.6) only support names from which the escape characters and trailing spaces have been removed. However, the legacy commands are upgraded to accept collections of objects instead of a simple list of names so you can pass in directly the output of commands such as get_pins and get _instances.

When the escape identifier and trailing spaces are removed, the example path “u1/\u2/u3 /\u4/u5 /insta” becomes “u1/u2/u3/u4/u5/insta”. As you can see, the four levels of hierarchy appear as six levels because the leaf instance names \u2/u3 and \u4/u5 look like two levels each.

Matching “u1/u2/u3/u4/u5/insta” against “u1/\u2/u3 /\u4/u5 /insta” requires searching for u1/\u2/u3 if no matches are found for u1/u2. You must apply this guessing approach recursively, which can be expensive when the escaped identifiers contain many slashes. More importantly, pattern matching on these names is difficult because the / is interpreted as a hierarchy separator which incorrectly splits the name_patterns into the wrong patterns to match against the level of hierarchies. For these reasons, we recommend you keep the escape identifiers in the name_patterns argument to prevent escaped slashes from being interpreted as hierarchy separators.

The following two examples both successfully match the instance with name “u1/\u2/u3 /\u4/u5 /insta”:

```
get_instances {{u1/\u2/u3   /\u4/u5 /insta}}
get_instances {{u1/\\u[0-9]+/u[0-9]+   /\\u[0-9]+/u[0-9]+   /insta}} -regexp
```

Because the get_instances command accepts a list of pattern names as input, two sets of braces are used to enclose the name_patterns argument. The Tcl shell performs a first evaluation to get the list items and consumes one set of braces; it performs a second evaluation on each item and consumes the second set of braces. Without two sets of braces, you would need to escape both bracket characters “[“ and “]” to keep them as part of the regular expression and prevent them from being interpreted as a Tcl command. Note how the “\” that is part of the escaped identifier

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is escaped by a second \ when the -regexp option is used. This is because the \ is a special character in Posix regular expression syntax and it must be escaped to be treated as a literal.

In the second example, “get_instances {{u1/\\u[0-9]+/u[0-9]+ /\\u[0-9]+/u[0-9]+ /insta}} -regexp”, the *leaf instance* matching insta, inside leaf instances matching “\u[0-9]+/u[0-9]+”, inside leaf instances matching “\u[0-9]+/u[0-9]+”, inside leaf instance matching u1 is returned. Note how the slashes within the escaped identifiers were not interpreted as hierarchy separators but as literals within the patterns.

The parser terminates the escape identifier string when it encounters a white space. For this reason, your patterns must not include any spaces; there is no reason to include spaces when matching instance path names.

If you need the Tcl shell to replace the Tcl variable within the regular expression, as in the example below, you must use double quotation marks instead of the inner set of braces and escape every special character. The \\ became \\\\ because the Tcl interpreter replaces \\\\ by \\, and the \\ is interpreted as a \ literal by the regexp parser.

Having to escape all special characters is very error prone. We recommend that you use the double sets of braces as shown above unless your pattern contains Tcl variables that must be replaced by the shell as shown below.

```
set range “0-9”
get_instances {“u1/\\\u\[$range\]+/u\[$range\]+ /\\\u\[$range\]+/u\[$range\]+ /insta”} \
        -regexp
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Attribute Inheritance Behavior

By default, attributes with the applies_to_child_instances option set to On are visible to all child instances below the current module. When you query the attribute value, the tool first looks to see if it is explicitly set for the specified instance. If the attribute is not set, it then traverses up the hierarchy and checks to see if the attribute is set in any of the parent instances. The first explicitly specified attribute value found is returned to you. The default value is returned if you did not explicitly set it at any level of the hierarchy.

Inheritance Rules 3840

Inheritance Rules

Attribute inheritance follows specific rules.

Normal Attribute Inheritance Rule

If the attribute is not registered for the current object type, the attribute can be inherited from a different object type. The individual data model descriptions (such as, Module, Instance, Port, and so on) provide an “Inheritance” section that lists the build-in inherited attributes.

Special Boolean Inheritance Rule

If a boolean attribute value is not specified for the current object, the attribute value can be inherited from an object of a different object type. To determine if a given attribute is specified for the current object, use the “get_attribute_value_list -is_specified” command and switch.

The following example demonstrates the boolean inheritance with a pin and a port, specifically the pin inheriting the attribute from its port:

```
register_attribute -name myboolattr -value_type boolean -object_types [list port pin]
set pin [get_pins i1/a]
set port [get_ports -of_pins $pin]
set_attribute_value $port -name myboolattr -value true // value is set to true on the port
set_attribute_value $pin -name myboolattr -value false // value is set to false on the pin
puts [get_attribute_value_list $pin -name myboolattr]
// no inheritance since the value is specified on the pin
```

```
false
```

```
report_attributes $pin
```

```
Attributes defined for object 'i1/a':
Name                     Value  Inheritance
----------------------   -----  -----------
…
myboolattr               false  -
…
```

```
puts [get_attribute_value_list $pin -name myboolattr -is_specified]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
1
```

reset_attribute_value $pin -name myboolattr // unspecified the value on the pin
puts [get_attribute_value_list $pin -name myboolattr] // value is inherited from the port

```
true
```

puts [get_attribute_value_list $pin -name myboolattr -is_specified]

```
0
```

report_attributes $pin

```
Attributes defined for object 'i1/a':
Name                     Value  Inheritance
----------------------   -----  -----------
…
myboolattr               true   port
…
```

Attribute Inheritance Reporting

Using the report_attributes command with a collection of objects lists an Inheritance column that shows if the value is set on the current object, or if the value is inherited from another object type. For example:

report_attributes [get_scan_elements -class wrapper]

```
Attribute Definition Report
Attributes defined for object '/\cnt_s_reg[0] ':
Name                     Value                     Inheritance
----------------------   ----------------------    -----------
at_speed_control_group   2                         -
cell_type                scan_cell                 module
```

Attributes defined with the applies_to_child_instances option set to “on” show the value that is inherited from a parent instance or icl_instance.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 13
Configuration-Based Specification

This chapter documents the configuration data syntax used to encode the DftSpecification, PatternsSpecification, and DefaultsSpecification information.

The configuration data syntax is composed of nested wrappers and properties that fully describe the DftSpecification, PatternsSpecification, and DefaultsSpecification/DftSpecification or DefaultsSpecification/PatternsSpecification information. You can create these specifications automatically using the create_dft_specification and create_patterns_specification commands, and they can be processed using the process_dft_specification and process_patterns_specification commands. You can edit and introspect the specifications with the commands listed in Table 13-1. The specifications can also be viewed and edited in the Config Data Browser window in Tessent Visualizer using the add_config_tab command.

This chapter also documents how to order boundary scan cells and a description of the BISR segment order file (described in the MemoryBisr command) that is used to order the BISR chains.

Table 13-1. Configuration Data Editing and Introspection Commands

Command	Description
add_config_element	Adds a configuration element in the configuration data.
add_config_message	Adds error, warning, or information messages to configuration elements.
add_config_tab	Adds one configuration tree tab to the Config Data Browser.
delete_config_element	Deletes one or more configuration elements. Only elements which can be added can be deleted.
delete_config_messages	Deletes error, warning or info message that were added to configuration elements using the add_config_message command.
delete_config_tabs	Deletes one or more configuration tree tabs from the Config Data Browser that were previously added using the add_config_tab or display_specification command.
get_config_elements	Returns a collection of configuration elements or a count of configuration elements when the -count option is used.
get_config_messages	Returns a list of message strings attached to a configuration element.
get_config_value	Returns a value associated with a configuration element based on the specified option.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-1. Configuration Data Editing and Introspection Commands (cont.)

Command	Description
move_config_element	Moves a configuration element from one location to another.
read_config_data	Reads a string, or the content of a list of configuration data files, into the Tessent Shell environment.
report_config_data	Reports the content of configuration wrappers in the transcript as it appears inside the file when the configuration data is written to a file using the write_config_data command.
report_config_messages	Reports messages associated with a configuration element. When the -hierarchical option is used, this command also reports the messages associated with elements below the specified wrapper.
report_config_syntax	Reports the legal configuration syntax for a specified configuration object.
set_config_value	Sets the value of an element in the configuration data.
write_config_data	Writes the configuration data currently in memory into a file. Use the -wrappers option to limit the data written to the file to specific wrappers.

This chapter uses the syntax conventions in Table 13-2 when documenting wrappers and properties used in configuration data.

Table 13-2. SyntaxConventions for Configuration Files

Convention	Example	Usage
Italic	***scan_in :*** *port_pin_name*;	An italic font indicates a user-supplied value.
Underline	lockup_cell_type : <u>latch</u> | dff;	An underlined item indicates the default value.
|	logic_level : <u>both</u> | high | low;	The vertical bar separates a list of values from which you must choose one. Do not include the bar in the configuration file.
...	port_naming : *port_naming*, ...;	Ellipses indicate a repeatable value. The comment "// repeatable" also indicates a repeatable value.
//	// default: ijtag_so	The double slash indicates the text immediately following is a comment and tells the tool to ignore the text.

DftSpecification Configuration Syntax 3846
DftSpecification 3848
TopModuleConnections 4551
PatternsSpecification Configuration Syntax 4555
PatternsSpecification 4556

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Patterns 4563
EnableGroupInfo 4687
SdfInfo 4689
LoadBoardInfo 4692
IjtagRetargetingOptions 4701
InSystemTest 4703

DefaultsSpecification Configuration Syntax 4707
DefaultsSpecification 4708
DefaultsSpecification/DftSpecification 4709
DefaultsSpecification/PatternsSpecification. 4751

Metadata Configuration Syntax 4757
Wrapper 4759
Property. 4768
RepeatableProperty. 4772
DataWrapper. 4774
CsvWrapper 4777
Id 4779
Value. 4782
RowIdElement 4787

DftSpecification and Complex Ports 4790
Embedded Boundary Scan Flow With Complex Pad IO Ports 4790

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DftSpecification Configuration Syntax

The DftSpecification is an open and flexible interface into Tessent Shell. The interface is designed to make all Tessent DFT instruments available within Tessent Shell. In addition, the interface will eventually enable third-party providers (including end-user in-house instruments) to integrate their instrument types into the DftSpecification process. This enables chip designers to benefit from a unified yet flexible platform that supports the definition and insertion of test-related instruments from an unlimited number of sources into their designs.

Using the configuration data syntax, you can specify a wide variety of DFT specifications in a single wrapper or in multiple DftSpecification wrappers, and generate and insert any type of instrument in one or multiple insertion passes.

Note

You can control the default values of many properties with the DefaultsSpecification/DftSpecification wrapper and the set_defaults_value command. The default values of properties marked with the notation "*DefSpec" in the usage line are controllable in this manner.

This section describes the configuration data syntax used inside Tessent Shell to specify DFT components that are to be built and, optionally, inserted into a design. This chapter covers these wrappers that exist inside the DftSpecification wrapper:

- EDT
- IjtagNetwork
- InSystemTest
- EmbeddedBoundaryScan
- BoundaryScan
- LogicBist
- LpctType3
- MemoryBisr
- MemoryBist
- OCC
- SSN
- RtlCells

DftSpecification 3848
EDT 3860
IjtagNetwork 3927

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

InSystemTest 4060
EmbeddedBoundaryScan 4093
BoundaryScan 4153
LogicBist 4217
LpctType3 4250
MemoryBisr 4262
MemoryBist 4310
OCC 4356
SSN 4391
RtlCells 4540

TopModuleConnections 4551

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DftSpecification

A wrapper that specifies the information used by the process_dft_specification command to build a series of instruments and, optionally, insert them into a design.

Usage

```
DftSpecification(module_name,id) {
  mini_occ_capture_trigger     : shift_en | capture_en;
                                             // default: shift_en
  rtl_extension                : string; //default: v
  gate_extension               : string; //default: vg
  reuse_modules_when_possible  : on | off | auto ; // *DefSpec
  use_rtl_cells                : on | off | auto ; // *DefSpec
  use_rtl_clock_dff_cell       : on | off | auto ; // *DefSpec
  use_synchronizer_cell_with_reset : on | off ;    // *DefSpec
  use_rtl_synchronizer_cell    : on | off | auto ; // *DefSpec
  use_rtl_clock_shaper_cell    : on | off | auto ; // *DefSpec
  dft_cell_selection_name      :  string ;
  persistent_clock_cell_prefix :  string ;
                                  //default: tessent_persistent_cell_
  persistent_cell_prefix       :  string ;
                                  //default: tessent_persistent_cell_
  allow_port_creation_on_current_design : on | off ;    // *DefSpec
  add_persistent_buffers_in_scan_resource_instruments : on | off ; //
*DefSpec
  independent_insertion_flow : on | off | auto ; // *DefSpec
  force_creation_of_rtl_cells  : cell_type, ... ;
            // legal : ActiveHighLatch, ActiveHighLatchReset,
            //         ActiveHighLatchSet, ActiveLowLatch,
            //         ActiveLowLatchReset, ActiveLowLatchSet,
            //         And2, Buf, ClkAnd2, ClkBuf, ClkGateAnd,
            //         ClkGateOr, ClkInv, ClkMux2, ClkOr2,
            //         ClkShaper, Inv, Mux2, NegedgeClockDff,
            //         NegedgeClockDffReset, NegedgeClockDffSet
            //         NegedgeDff, NegedgeDffReset
            //         NegedgeDffSet, Or2, PosedgeClockDff,
            //         PosedgeClockDffReset, PosedgeClockDffSet,
            //         PosedgeDff, PosedgeDffReset,
            //         PosedgeDffSet, PosedgeSynchronizer,
            //         PosedgeSynchronizerReset
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
    RtlCells {
    }
    BoundaryScan {
    }
    EDT {
    }
    EmbeddedBoundaryScan {
    }
    IjtagNetwork {
    }
    InSystemTest {
    }
    LogicBist {
    }
    LpctType3 {
    }
    MemoryBisr {
    }
    MemoryBist {
    }
    OCC {
    }
    SSN {
    }

    //Other instrument types to be added in future releases
  }
```

Description

A wrapper that specifies the information used by the process_dft_specification command to build a series of instruments and, optionally, insert them into a design.

The DftSpecification wrapper is uniquely named using two strings. The first string is the module_name of the design the DFT specification is for. The second string is a general purpose id that differentiates multiple DFT specifications for a given design module where each DftSpecification is for a different insertion pass.

Arguments

- *module_name*

  A string that identifies the design that the DFT specification is for. When running the process_dft_specification command, the name of the current design must match this string unless the -no_insertion option is specified with this command; if the -no_insertion option is used, it can also be specified using the -design_name design_name option.

- *id*

  A string that identifies which insertion pass the DFT specification is to be used for. When executing the process_dft_specification command, the id argument of the command is used to specify this id used in the DftSpecification name. If only one DftSpecification wrapper for a given module_name is loaded in memory, the process_dft_specification command uses it even if the id is not specified as an option. Using either “rtl” or “gate” as the id string

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is a good strategy to identify the instruments inserted in the RTL and those inserted in the synthesized netlist.

- mini_occ_capture_trigger : *shift_en* | *capture_en*;

  A property that specifies whether shift_en or capture_en is used as the trigger to generate programmable capture pulses. The capture_en version should be used when internal test logic generates a separate capture_en signal. The default is shift_en.

  Choose from one of the following:

  - shift_en — The mini-OCC controllers infer the shift_en port.
  - capture_en — The mini-OCC controllers infer the capture_en port.

- rtl_extension : *string* ;

  A string that specifies the extension used when writing out files containing Verilog RTL descriptions of the created instruments. The string must start with a letter and may be followed by any number of letters, numbers, underscores, and periods. The value of rtl_extension must be different than the value of gate_extension. The default value when unspecified is “v”.

- gate_extension : *string* ;

  A string that specifies the extension used when writing out files containing gate-level Verilog modules. This extension is used in the synthesis script of the instruments and when writing out the concatenated netlist. The gate_extension is only relevant in the -no_rtl context. The string must start with a letter and may be followed by any number of letters, numbers, underscores, and periods. The value of gate_extension must be different than the value of rtl_extension. The default value when unspecified is “vg”.

- reuse_modules_when_possible : on | off | auto ;

  A Boolean that specifies whether instrument modules should be reused if possible, or if different modules should be created for each instance. When the value is auto, it is considered to be On in the rtl context and Off in the no_rtl context. When inserting instruments into a netlist, it is customary to avoid repeated instances of any module as timing optimization requires them to be uniquified. The default value is auto.

  Criteria for not reusing modules include, but are not limited to, port presence, port naming, signal polarity, ICL attribute name and value, scan register capture value, scan register reset value, and so on.

- use_rtl_cells : on | off | auto ;

  A property that specifies whether RTL or gate-level cells are to be instantiated into the generated controllers in special locations.

  For example, if a controller needs a clock multiplexer, the tool does not describe the clock multiplexer in RTL but instead instantiates a cell. When the use_rtl_cells property is specified or inferred as “on”, the tool creates the clock multiplexer as a special RTL cell and instantiates this cell into your design. You can control the name of the ports for the cell such that later you can replace the RTL cell with a real cell using the replace_instances

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

command. Modules created by tools are often marked as "hard" to prevent accidental editing inside the modules. To replace the cell instances inside those modules, you may have to set the is_hard_module attribute to false on the parent module of the cell. Ensure that you limit the editing to only cell swapping as you do not want to make the RTL views different from their associated ICL or tcd_scan views. Refer to the RtlCells wrapper to change the port names.

When the use_rtl_cells attribute is specified or inferred as "off", the tool uses the cells in the cell library and instantiates these cells into your design. Refer to the get_dft_cell command and the dft_cell_selection_name property to understand which specific cell the tool uses.

When the default value of "auto" is used, the on value is inferred in this specific condition:

- The -rtl context is used.
- You have not loaded a tessent cell library using the read_cell_library command containing the needed cell.

The behavior of the "auto" value was changed in the Tessent v2017.2 release to favor using real cells even in rtl context when available. Using RTL cells is no longer recommended given the lack of design module boundary preservation in the modern synthesis tool.

- use_rtl_clock_dff_cell : on | off | auto ;

  A property that controls instantiation of an RTL clock DFF cell. When this is set to "on", it forces the tool to create and instantiate an RTL clock DFF cell, even when use_rtl_cells is set to "off" or when operating in the -no_rtl context. When this is set to "auto," the tool uses the use_rtl_cells setting to control its behavior, except that it reverts to "on" when the library does not have an identified clock DFF cell. If this is set to "off" and the requested clock DFF cell does not occur in the cell library, the tool reports an error and prompts you to set this property to "on".

- use_rtl_clock_shaper_cell : on | off | auto ;

  A property that controls instantiation of an RTL clock shaper cell. If you set this to "on", it forces the tool to create and instantiate an RTL clock shaper cell, even when use_rtl_cells is set to "off" or when operating in the -no_rtl context. If you set this to "auto," the tool uses the use_rtl_cells setting to control its behavior, except that it reverts to "on" if the library does not have an identified clock shaper cell. If this is set to "off" and the requested clock shaper cell does not occur in the cell library, the tool reports an error and prompts you to set this property to "on".

  The RTL clock shaper cell is implemented with two variants, ClkShaperResetActive and ClkShaperResetInactive. The tool implements the RTL clock shaper cell using ClksShaperResetActive by default. If test_clock_present and use_ssn_bus_clock_as_test_clock_bypass in the ScanHost node are both on, the tool uses ClkShaperResetInactive instead. ClkShaperResetInactive enables you to suppress the clock in inactive mode.

  The clock shaper cell is only used in the ScanHost node, and a more complete description of the clock shaper RTL cell appears in "RtlCells" on page 4540.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- use_synchronizer_cell_with_reset : on | off ;

  A property that, if you specify to "off", enables the tool to request a basic synchronizer cell that does not have a reset. The tool requests a synchronizer cell with reset by default if you specify this property to "on". If the requested synchronizer cell with reset is not found in the cell library, the tool returns an error and the you are prompted to set use_synchronizer_cell_with_reset to "off" to enable use of a basic synchronizer cell.

  This property is currently used by the OCC, MemoryBist, and LogicBist wrappers.

- use_rtl_synchronizer_cell : on | off | auto ;

  A property that when set to "on", forces the tool to create and instantiate an RTL synchronizer cell, even when use_rtl_cells is set to "off" or when operating in the -no_rtl context. When set to "auto", the tool follows the use_rtl_cells behavior. In the case when the property is set to "off" and the requested synchronizer cell is not found in the cell library, an error is issued and the user is prompted to set this parameter to "on".

  This property is currently used by the OCC, MemoryBist and LogicBist wrappers.

- dft_cell_selection_name : *string* ;

  A property that specifies the name of the dft_cell_selection wrapper from which the cells are selected. You only need to specify this property if all of the following apply:

  - RTL cells are not used
  - Your cell library has more than one dft_cell_selection wrapper
  - You have not specified a default dft_cell_selection wrapper using the set_cell_library_options -default_dft_cell_selection_name command

- persistent_clock_cell_prefix : *string* ; // default: tessent_persistent_cell_

  A property that specifies the leaf instance name prefix to use when instantiating a clock cell such as a clock multiplexer or a clock gating cell. The string must start with a letter and be followed by only letters, numbers, and underscores. When unspecified, it defaults to "tessent_persistent_cell_". These clock cells have a special prefix so that they are not removed during synthesis and layout. You can modify this string to match the prefix you normally use in your design flow; otherwise, use the following Design Compiler command in your synthesis and layout scripts:

  ```
  set_size_only -all_instance \
    [get_cells  tessent_persistent_cell_* -hierarchical \
    -filter "is_hierarchical==false"]
  ```

  Refer to the SDC proc described in the "Generating and Using SDC for Tessent Shell Embedded Test IP" in the *Tessent Shell User's Manual* for more information. Specifically, look at the Synthesis Step 3 in the "Example Design Compiler Synthesis Script".

- persistent_cell_prefix : *string* ; // default: tessent_persistent_cell_

  A property that specifies a leaf instance name prefix to use when instantiating a non-clock cell such as a buffer or an inverter. The string must start with a letter and be followed by

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

only letters, numbers, and underscores. When unspecified, it defaults to "tessent_persistent_cell_". These non-clock cells have a special prefix to ensure that they are not removed during synthesis and layout so that they can be used as anchor points for timing constraints or as start and end points in ScanDef. You can modify this string to match the prefix you normally use in your design flow; otherwise, use the following Design Compiler command in your synthesis and layout scripts:

```
set_size_only -all_instance \
  [get_cells  tessent_persistent_cell_* -hierarchical \
  -filter "is_hierarchical==false"]
```

Refer to the SDC proc described in the "Generating and Using SDC for Tessent Shell Embedded Test IP" in the *Tessent Shell User's Manual* for more information. Specifically, look at the Synthesis Step 3 in the "Example Design Compiler Synthesis Script".

- allow_port_creation_on_current_design : on | off ;

  A boolean value to enable the creation of DFT ports on the current design. You can use this if the design_level of the current design is not "chip."

  Typically, you cannot create ports on a physical block during bottom-up DFT insertion in a tiling design flow. These blocks are created in advance in a top-down fashion to create the chip-level netlist and floor plan the port placement of each tile.

  For this reason, the create_dft_specification command automatically sets the allow_port_creation_on_current_design property to "off" if you specify the "set_dft_specification_requirements -design_type" command to "tile."

  Tip

  See the process_top_module_connections command options to learn how to create your DFT ports in a top-down fashion within the tool. If you want to create them during the bottom-up DFT insertion flow, use the "set_config_value -unspecified" command to reset the allow_port_creation_on_current_design property back to the default "on" value before running the process_dft_specification command.

- add_persistent_buffers_in_scan_resource_instruments : on | off ;

  A property that specifies to add persistent buffers to relevant controller ports and internal pins of logic test instruments, such as the EDT, LogicBIST, and In-System Test controllers, and Tessent OCCs. The persistent buffers enable pattern generation automation using the TCD mapping flow when these logic test instruments are ungrouped during physical implementation.

- independent_insertion_flow : on | off | auto ;

  A property that specifies whether the tool uses the legacy concurrent insertion flow.

  Choose from one of the following:

  - on — The tool uses the independent insertion flow.
  - off — The tool uses the concurrent insertion flow.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- auto — The tool turns the independent insertion flow on when DFT signals or DftSpecification properties (such as SetLoadUnloadTimingOptions) related only to the independent insertion flow are present in the Tessent Shell command scripts. If these signals or properties are not present, this switch disables the independent insertion flow.

The following features are enabled when the independent_insertion_flow is on:

- You can add LBIST-ready EDT controllers to child design levels and reuse them as sub-blocks in other designs.
- The tool adds EDT controllers directly on the IJTAG network instead of adding them as a subnetwork of the LBIST controller's IJTAG network.
- The tool creates a simplified single chain using the existing IJTAG network chains, instead of creating a separate IJTAG interface for LBIST.
- The tool reuses any edt_clock and shift_capture_clock gaters so that OCC and EDT receive the expected clock pulse.
- The tool creates DFT control signals from the LBIST controller to the sub-blocks. These signals enable communication between the two passes when the tool inserts LBIST-ready blocks and the LBIST controller.
- The tool creates fully programmable LBIST load and unload waveforms during pattern retargeting.
- The DftSpecification/OCC/capture_trigger default is shift_en.
- The tool enables local TCK injection in both the LBIST and EDT modules.
- The tool creates LBIST reset as an asynchronous signal.

- force_creation_of_rtl_cells : *cell_type*, ... ;

  A property that specifies the creation of specific RTL cells so that you can use them in a custom insertion script such as those you would use inside the process_dft_specification.post_insertion procedure. Refer to Table 13-3 for valid <*cell_type*> values and the usages for the cells.

  Tip

  It is recommended to directly instantiate real cells into the RTL and use a specific leaf instance prefix to instruct the synthesis process to preserve it. If you want to use RTL cells in the RTL, it is recommended to use the RtlCells wrapper to control their port names and to ensure they match the exact naming of your library when you specify them. This makes it easy to simply rename the module in the instantiation to start using real cells in the netlist.

  If you want to use set_config_value to set this property with a list of cell names, use a Tcl list of cell names. The list of RTL cells results in a comma-separated list in the configuration data. See "Example 2" on page 2390 in the set_config_value command description.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-3. Valid cell_type Values and Usages

Cell Type	Usage
ActiveHighLatch	Requests creation of an RTL cell that implements the behavior of a normal active high latch. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
ActiveHighLatchReset	Requests creation of an RTL cell that implements the behavior of a normal active high latch having an asynchronous reset port. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports and the active polarity of the reset port.
ActiveHighLatchSet	Requests creation of an RTL cell that implements the behavior of a normal active high latch having an asynchronous set port. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports and the active polarity of the set port.
ActiveLowLatch	Requests creation of an RTL cell that implements the behavior of a normal active low latch. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
ActiveLowLatchReset	Requests creation of an RTL cell that implements the behavior of a normal active low latch having an asynchronous reset port. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports and the active polarity of the reset port.
ActiveLowLatchSet	Requests creation of an RTL cell that implements the behavior of a normal active low latch having an asynchronous set port. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports and the active polarity of the set port.
And2	Requests creation of an RTL cell that implements the behavior of a normal 2-input AND gate. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
Buf	Requests creation of an RTL cell that implements the behavior of a normal buffer gate. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
ClkAnd2	Requests creation of an RTL cell that implements the behavior of a 2-input AND gate intended to be used along the clock path. A clock cell is designed to minimize duty cycle degradation and has special isolation to minimize crosstalk. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
ClkBuf	Requests creation of an RTL cell that implements the behavior of a buffer gate intended to be used along the clock path. A clock cell is designed to minimize duty cycle degradation and has special isolation to minimize crosstalk. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-3. Valid cell_type Values and Usages (cont.)

Cell Type	Usage
ClkGateAnd	Requests creation of an RTL cell that implements the behavior of a clock gating cell that gates the clock low when not enabled. The cell has two enable ports that are both active high, one used by the functional mode and one used during scan test. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
ClkGateOr	Requests creation of an RTL cell that implements the behavior of a clock gating cell that gates the clock high when not enabled. The cell has two enable ports that are both active high, one used by the functional mode and one used during scan test. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
ClkInv	Requests creation of an RTL cell that implements the behavior of an invert gate intended to be used along the clock path. A clock cell is designed to minimize duty cycle degradation and has special isolation to minimize crosstalk. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
ClkMux2	Requests creation of an RTL cell that implements the behavior of a 2-input multiplexer intended to be used along the clock path. A clock cell is designed to minimize duty cycle degradation and has special isolation to minimize crosstalk. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
ClkOr2	Requests creation of an RTL cell that implements the behavior of a 2-input OR gate intended to be used along the clock path. A clock cell is designed to minimize duty cycle degradation and has special isolation to minimize crosstalk. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
ClkShaper	Requests creation of an RTL cell that implements the behavior of a special clock shaper cell. Refer to "RtlCells" on page 4540 for a description of the content of the clock shaper cell. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports and the active polarity of the reset port.
Inv	Requests creation of an RTL cell that implements the behavior of a normal inverter gate. Such cells are not intended to be used along the clock path; use a ClockInv cell instead. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-3. Valid cell_type Values and Usages (cont.)

Cell Type	Usage
Mux2	Requests creation of an RTL cell that implements the behavior of a normal 2 input multiplexer gate. Such cells are not intended to be used along the clock path; use a ClkMux2 cell instead. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
NegedgeClockDff	Requests creation of an RTL cell that implements the behavior of a negative edge D-type flip-flop that is intended to be used along the clock path to source a divided clock. A clock cell is designed to minimize duty cycle degradation and has special isolation to minimize crosstalk. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
NegedgeClockDffReset	Requests creation of an RTL cell that implements the behavior of a negative edge D-type flip-flop having an asynchronous reset port. The flip-flop is intended to be used along the clock path to source a divided clock. A clock cell is designed to minimize duty cycle degradation and has special isolation to minimize crosstalk. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports and the active polarity of the reset port.
NegedgeClockDffSet	Requests creation of an RTL cell that implements the behavior of a negative edge D-type flip-flop having an asynchronous set port. The flip-flop is intended to be used along the clock path to source a divided clock. A clock cell is designed to minimize duty cycle degradation and has special isolation to minimize crosstalk. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports and the active polarity of the set port.
NegedgeDff	Requests creation of an RTL cell that implements the behavior of a normal negative edge D-type flip-flop. Such cells are not intended to be used along the clock path; use a NegedgeClockDff instead. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
NegedgeDffReset	Requests creation of an RTL cell that implements the behavior of a normal negative edge D-type flip-flop having an asynchronous reset port. Such cells are not intended to be used along the clock path; use a NegedgeClockDffReset instead. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports and the active polarity of the reset port.
NegedgeDffSet	Requests creation of an RTL cell that implements the behavior of a normal negative edge D-type flip-flop having an asynchronous set port. Such cells are not intended to be used along the clock path; use a NegedgeClockDffSet instead. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports and the active polarity of the set port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-3. Valid cell_type Values and Usages (cont.)

Cell Type	Usage
Or2	Requests creation of an RTL cell that implements the behavior of a normal 2-input OR gate. Such cells are not intended to be used along the clock path; use a ClockOr2 instead. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
PosedgeClockDff	Requests creation of an RTL cell that implements the behavior of a positive edge D-type flip-flop that is intended to be used along the clock path to source a divided clock. A clock cell is designed to minimize duty cycle degradation and has special isolation to minimize crosstalk. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
PosedgeClockDffReset	Requests creation of an RTL cell that implements the behavior of a positive edge D-type flip-flop having an asynchronous reset port. The flip-flop is intended to be used along the clock path to source a divided clock. A clock cell is designed to minimize duty cycle degradation and has special isolation to minimize crosstalk. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports and the active polarity of the reset port.
PosedgeClockDffSet	Requests creation of an RTL cell that implements the behavior of a positive edge D-type flip-flop having an asynchronous set port. The flip-flop is intended to be used along the clock path to source a divided clock. A clock cell is designed to minimize duty cycle degradation and has special isolation to minimize crosstalk. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports and the active polarity of the set port.
PosedgeDff	Requests creation of an RTL cell that implements the behavior of a normal positive edge D-type flip-flop. Such cells are not intended to be used along the clock path; use a PosedgeClockDff instead. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
PosedgeDffReset	Requests creation of an RTL cell that implements the behavior of a normal positive edge D-type flip-flop having an asynchronous reset port. Such cells are not intended to be used along the clock path; use a PosedgeClockDffReset instead. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports and the active polarity of the reset port.
PosedgeDffSet	Requests creation of an RTL cell that implements the behavior of a normal positive edge D-type flip-flop having an asynchronous set port. Such cells are not intended to be used along the clock path; use a PosedgeClockDffSet instead. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports and the active polarity of the reset port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-3. Valid cell_type Values and Usages (cont.)

Cell Type	Usage
PosedgeSynchronizer	Requests creation of an RTL cell that implements the behavior of a positive edge synchronizer cell, which consists of two D-type flip-flops in series. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports.
PosedgeSynchronizerReset	Requests creation of an RTL cell that implements the behavior of a positive edge synchronizer cell, which consists of two D-type flip-flops in series. Both flip-flops have asynchronous reset ports. Use the wrapper of the same name in the RtlCells wrapper to control the names of its ports and the active polarity of the reset port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

EDT

A wrapper that specifies the information used by the process_dft_specification command to build the EDT IP and, optionally, insert the IP into a design.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    ijtag_host_interface : ijtag_leaf_node_name | none ;
    LogicBistOptions { // use to specify common options across
    }                  // all controllers
    ControllerChain {
    }
    Connections {
    }
    Controller(id) {
    }
  }
}
```

Description

A wrapper that specifies the information used by the process_dft_specification command to build the EDT IP and, optionally, insert the IP into a design.

Arguments

- ijtag_host_interface : *ijtag_leaf_node_name* | none ;

  This optional property is used to specify the host scan interface that the EDT controllers connect to. This property has an override inside the Controller wrapper.

  The *node_id* value must have the format Sib(*id*), Tap(*id*)/HostIjtag(*id*), HostScanInterface(*id*), ScanMux(*id*)/Input(0|1) and must match the leafname of a wrapper in the IjtagNetwork wrapper.

  To have the IjtagNetwork created with a Sib node for the service of the EDT node, use the -sri_sib_list switch when calling the create_dft_specification command. For example, use -sri_sib_list {edt occ} when inserting the logic test hardware.

  When you specify ijtag_host_interface with a value pointing to an IJTAG host scan interface during process_dft_specification, a TDR is added that drives the static external controls of the EDT. If you have also specified static external controls which are tied to top level ports or pins, the TDR connections to the EDT static external controls override the specified static external controls.

  When the property is unspecified, it defaults to none which may be the inherited values by all controller wrappers if the ijtag_host_interface property is not specified locally.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

LogicBistOptions

This wrapper groups the LogicBist-related options set global to all Controllers.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    LogicBistOptions {
      present              : on | off | auto;
      capture_per_cycle    : on | off | auto ;
      prpg_reference_seed  : hex ; // default: 1
      self_test            : on | off | auto ;
      ShiftCycles {
      }
      WarmupPatternCount {
      }
    }
  }
}
```

Description

Use this wrapper to set LogicBIST-related options for the EDT controllers.

To use the independent TK/LBIST insertion flow, set the “present” property to “on” and do not specify a DftSpecification/LogicBist wrapper.

Arguments

- present : on | off | auto ;

  An optional property that specifies whether the hybrid EDT/LBIST controllers are generated. When this property is set to on and a DftSpecification/LogicBist/Controller wrapper is not specified, the EDT is set for LBIST-ready hybrid EDT/LBIST insertion. When this property is set to off, then the controller is a standard EDT controller (non-hybrid) that is not usable by LBIST.

  By default, auto resolves to on when a DftSpecification/LogicBist/Controller is specified; otherwise it resolves to off. The default is auto.

- capture_per_cycle : on | off | auto ;

  An optional property that specifies whether observation scan is used in the LBIST-ready block. By default, auto resolves to off when the capture_per_cycle DFT signal is not added; otherwise it is on.

  When capture_per_cycle is set to on and the capture_per_cycle_dynamic_en does not exist, the signal is automatically inferred on the port when the current design level is sub-block or physical block, and inferred on a buffer with input tied to zero when the current design level is chip.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- prpg_reference_seed : *hex* :

  An optional property that specifies the starting seed for all EDT controllers in the DftSpecification. The specified hex value for prpg_reference_seed is used as the starting seed for the first hybrid EDT, a permutation of the specified seed is used as the starting seed for the second controller, and so forth. The starting seed is used to find the actual faultsim starting seed that is computed based on warmup patterns and hw_default shift length. This property can be overwritten by the prpg_seed property on a per-controller basis.

- self_test : on | off | <u>auto</u> ;

  An optional property that specifies whether to add self-test logic. By default, auto resolves to off when the lbist_self_test_en signal is not added; otherwise it is on.

  When self_test is set to on and the lbist_self_test_en does not exist, the signal is automatically inferred on the port when the current design level is sub-block or physical block, and inferred on the buffer with input tied to zero when the current design level is chip.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ShiftCycles

A wrapper that configures the shift cycles for the hybrid TK/LBIST EDT common across all controllers.

Usage

```
DftSpecification (module_name,id) {
  EDT {
    LogicBistOptions {
      ShiftCycles {
        max               : int ;
        hardware_default : int ;   // default: max
      }
    }
  }
}
```

Description

A wrapper the configures the shift cycles for the hybrid TK/LBIST EDT common across all controllers.

Arguments

- max : *int* ;

  This property specifies the maximum number of shift cycles for the controller.

- hardware_default : *int* ;

  This property specifies the default maximum number of shift cycles for the controller. The default is the hardware maximum.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

WarmupPatternCount

This wrapper configures the hybrid TK/LBIST warm-up pattern count for all EDT controllers.

Usage

```
DftSpecification (module_name,id) {
  EDT {
    LogicBistOptions {
      WarmupPatternCount {
        max              : int ; // default: 255
        hardware_default : int ; // default: 0
      }
    }
  }
}
```

Description

This wrapper configures the hybrid TK/LBIST warm-up pattern count for all EDT controllers. See “Warm-Up Patterns” in the *Hybrid TK/LBIST Flow User’s Manual* for complete information.

Arguments

- max : *int* ;

  This property specifies the maximum number of warm-up patterns for the controller. The default is 255.

- hardware_default : *int* ;

  This property specifies the default maximum number of warm-up patterns for the controller. The default is 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ControllerChain

This wrapper is used to add and specify properties for the EDT Controller Chain Mode (CCM).

Usage

```
Edt {
  ControllerChain {
    present                     : on | off ;    // default: off
    clock                       : enum ;        // legal  : tck, edt_clock
                                                // default: edt_clock
    segment_per_instrument      : on | off ;    // default: on
    max_segment_length          : int | unlimited ; // int >= 32
    Interface {
      enable                    : port_name ;   // default: ccm_en
      scan_en                   : port_name ;   // default: scan_en
      scan_in                   : port_name ;   // default: scan_in
      scan_out                  : port_name ;   // default: scan_out
    }
    Connections {
      scan_en                   : port_pin_name ;
                                    // default: OptionalDftSignal(scan_en)
      controller_chain_enable   : port_pin_name ;
                        // default : OptionalDftSignal(controller_chain_mode)
      controller_chain_scan_in  : port_pin_name ;
                                    // default: control_chain_%s_scan_in
      controller_chain_scan_out : port_pin_name ;
                                    // default: control_chain_%s_scan_out
    }
  }
}
```

Description

For hybrid controllers (that is, if the LogicBist/Controller wrapper is present in the specification), this wrapper inherits its values from the LogicBist/Controller/ControllerChain wrapper. Otherwise, for non-hybrid controllers, the values are as specified in the following argument descriptions. If you specify both wrappers and there is a conflict in the value settings, the tool reports an error.

Arguments

- present : on | off ;

  This property specifies whether to enable or disable the CCM. The default is off. When this is turned on, the tool creates RTL scan chain segments within the controller blocks. You can then connect the CCM segments to form scan chains and generate ATPG patterns for this mode to test the controller logic.

- clock : *enum* ;

  This property specifies the clock for the tool to use during CCM. For non-hybrid controllers, the only legal value is edt_clock. Hybrid controllers can use tck.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- segment_per_instrument : on | off ;

  An optional switch and literal pair that specifies whether, during IP insertion, the tool automatically connects controller scan chain segments within the IP blocks into one scan chain, or leaves them as segments. The default is on (leave as segments).

  This option must be set to on in the following circumstances:

  - When an integer value is set for the max_segment _length property.
  - When the independent_insertion_flow property is enabled.

    You must segment the CCM chains if LBIST hosts the IJTAG subnetwork with no hybrid EDT placements.

- max_segment_length : *int* | unlimited ;

  This property specifies the maximum controller scan chain segment length in Controller Chain Mode. The minimum segment length is 32. The default is unlimited, meaning a single scan chain segment for each instrument. When the total number of scan cells is fewer than the specified length, the tool creates a single chain.

- Interface/enable : *port_name* ;

  A property that specifies the name of the controller chain enable port. The default value is ccm_en.

- Interface/scan_en : *port_name* ;

  A property that specifies the name of the controller chain scan enable port. The default value is scan_en.

- Interface/scan_in : *port_name* ;

  A property that specifies the name of the controller chain scan in port. The default value is ccm_scan_in.

- Interface/scan_out : *port_name* ;

  A property that specifies the name of the controller chain scan out port. The default value is ccm_scan_out.

Note

The tool uses the Connections properties only when segment_per_instrument is off. You cannot have any scan chain configuration other than one scan chain connected per controller. For more complex setups, use the scan insertion tool.

- Connections/scan_en : *port_pin_name* ;

  An optional property that specifies the connection for the controller scan enable pin. This pin is only used when “present” is set to on. The default is OptionalDftSignal(scan_en). When the edt_clock is created from test_clock, this should specify a valid pin or port name and may not be connected to a constant.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Connections/controller_chain_enable : *port_pin_name* ;

  An optional property that specifies the connection for the controller chain enable pin. This pin is only used when “present” is set to on. The default is OptionalDftSignal(controller_chain_mode). When the edt_clock is created from test_clock, this should specify a valid pin or port name and may not be connected to a constant.

- Connections/controller_chain_scan_in : *port_pin_name* ;

  An optional property that specifies the connection for the controller chain scan-in pin. This pin is only used when “present” is set to on. The default is control_chain_<*%s*>_scan_in. The <*%s*> value is replaced by the name of the controller in the default value.

- Connections/controller_chain_scan_out : *port_pin_name* ;

  An optional property that specifies the connection for the controller chain scan-out pin. This pin is only used when “present” is set to on. The default is control_chain_<*%s*>_scan_out. The <*%s*> value is replaced by the name of the controller in the default value.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Connections

This wrapper is used to specify EDT control pin connections for each EDT controller.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Connections {
      edt_clock       : port_pin_name | DftSignal(edt_clock) |
                        OptionalDftSignal(edt_clock);
      edt_slave_clock : port_pin_name ; // default: edt_slave_clock
      edt_update      : port_pin_name | DftSignal(edt_update) |
                        OptionalDftSignal(edt_update);
      edt_reset       : port_pin_name ; // default: edt_reset
      StaticExternalControls {
      }
    }
  }
}
```

Description

This wrapper is used to specify EDT control pin connections for each EDT controller.

Arguments

- edt_clock : *port_pin_name* ;

  An optional property used to specify a source node to connect the edt_clock port to. This is equivalent to the "set_edt_pins clock" command.

  The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the set_design_level command, is not "chip" and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it is created as an input port during insertion.

  When design level is "chip", specified port names must already be connected to a pad cell. The connection automatically made to the internal side of the input pad buffer. You can specify OptionalDftSignal(edt_clock) or DftSignal(edt_clock) as the value, and it is interpreted to request a connection to the DFT signal edt_clock as it was added using the add_dft_signals command. When using OptionalDftsignal(edt_clock), you do not get an error if the DFT signal edt_clock was not added; a connection to the port called "edt_clock" is made instead and the port is created if not present unless design level is equal to "chip" in which case you get an error.

  When the property is unspecified, its value defaults to OptionalDftSignal(edt_clock).

- edt_slave_clock : *port_pin_name* ;

  An optional property that specifies the EDT slave clock. This is equivalent to the "set_edt_pins slave_clock" command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When you generate level-sensitive clocking (latch-based EDT IP), the EDT IP contains a master clock and a slave clock to connect. See "TestKompress Compression Logic" in the *Tessent TestKompress User's Manual*. The default is edt_slave_clock.

- edt_update : *port_pin_name* ;

  An optional property used to specify a source node to connect the edt_update port to. This is equivalent to the "set_edt_pins update" command.

  The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the set_design_level command, is not "chip" and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it is created as an input port during insertion.

  When design level is "chip", specified port names must already be connected to a pad cell. The connection is automatically made to the internal side of the input pad buffer. You can specify OptionalDftSignal(edt_update) or DftSignal(edt_update) as the value, and it is interpreted to request a connection to the DFT signal edt_update as it was added using the add_dft_signals command. When using OptionalDftsignal(edt_update), you do not get an error if the DFT signal edt_update was not added; a connection to the port called "edt_update" is made instead and the port is created if not present unless design level is equal to "chip" in which case you get an error.

  When the property is unspecified, its value defaults to OptionalDftSignal(edt_update).

- edt_reset : *port_pin_name* ;

  An optional property that specifies the port or pin name to be used for the EDT reset. This is equivalent to the "set_edt_pins reset" command.

  The default is edt_reset. The edt_reset signal is an asynchronous reset for all the sequential elements in the EDT logic. For more information, see "EDT Logic Reset" in the *Tessent TestKompress User's Manual.*

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

StaticExternalControls

This wrapper specifies the connections for the static EDT control signals.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Connections {
      StaticExternalControls {
        edt_bypass                 : port_pin_name | 0 | 1 ;
        edt_single_bypass_chain    : port_pin_name | 0 | 1 ;
        edt_configuration          : port_pin_name | 0 | 1 ;
        edt_low_power_shift_enable : port_pin_name | 0 | 1 ;
      }
    }
  }
}
```

Description

This wrapper specifies the connections for the static EDT control signals.

Arguments

- edt_bypass : *port_pin_name* | 0 | 1 ;

  An optional property that specifies the connection to the edt_bypass static EDT control pin. This is equivalent to the "set_edt_pins bypass" command.

  The edt_bypass pin connects to the specified port_pin_name, or is tied to high or low using the 1 or 0 symbol. The default is 0.

- edt_single_bypass_chain : *port_pin_name* | 0 | 1 ;

  An optional property that specifies the connection to the edt_single_bypass_chain static EDT control pin. This is equivalent to the "set_edt_pins single_bypass_chain" command.

  The edt_single_bypass_chain pin connects to the specified port_pin_name, or is tied to high or low using the 1 or 0 symbol. The default is 0.

- edt_configuration : *port_pin_name* | 0 | 1 ;

  An optional property that specifies the connection to the edt_configuration static EDT control pin. This is equivalent to the "set_edt_pins configuration" command.

  The edt_configuration pin connects to the specified port_pin_name, or is tied to high or low using the 1 or 0 symbol. The default is 0.

- edt_low_power_shift_enable : *port_pin_name* | 0 | 1 ;

  An optional property that specifies the connection to the edt_low_power_shift_enable static EDT control pin. This is equivalent to the "set_edt_pins low_power_shift_en" command.

  The edt_low_power_shift_enable pin connects to the specified port_pin_name, or is tied to high or low using the 1 or 0 symbol. The default is 1.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Controller

Use this wrapper to add and specify properties for every EDT controller generated with process_dft_specification.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      ijtag_host_interface           : ijtag_leaf_node_name | none ;
      longest_chain_range            : min, max   ;
      scan_chain_count               : int        ;
      input_channel_count            : int        ;
      output_channel_count           : int        ;
      separate_control_data_channels : on | off   ;
      parent_instance                : name       ;
      leaf_instance_name             : name       ;
      connect_bscan_segments_to_lsb_chains : on | off | auto ;
      edt_bypass_change_edge_clock   : prefer_edt_clock | scan_clock ;
      LVxMode {
        present                      : on | off | auto ;
        enable_one_chain             : on | off ;
        chain_output_masking_disable : on | off ;
      }
      Interface {
      }
      BypassChains {
      }
      Compactor {
      }
      Clocking {
      }
      HighCompressionConfiguration {
      }
      ShiftPowerOptions {
      }
      LogicBistOptions {
      }
      Connections {
      }
      Decompressor {
      }
    }
  }
}
```

Description

The process_dft_specification command uses the settings from this wrapper to generate EDT controllers. Every Controller needs to have a unique simple name. You must use a name that starts with an alphanumeric character and contains no special characters except for underscores (_) in the middle of the controller name. The name must not end with an underscore or contain

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

two consecutive underscores. You can also use an integer to specify Controller names. For example, the following controller names are valid:

```
Controller(1)
Controller(c1)
Controller(controller_1)
```

Arguments

- ijtag_host_interface : *ijtag_leaf_node_name* | none ;

  An optional property that connects the EDT static external controls through a test data register (TDR). This wrapper inherits the ijtag_host_interface value from the top level. If you specify it at this level, it overrides the ijtag_host_interface specified at the top level.

  The *node_id* value must have the format Sib(*id*), Tap(*id*)/HostIjtag(*id*), HostScanInterface(*id*), ScanMux(*id*)/Input(0|1) and must match the leafname of a wrapper in the IjtagNetwork wrapper.

  To have the IjtagNetwork created with a Sib node for the service of the EDT node, use the -sri_sib_list switch when calling the create_dft_specification command. For example, use -sri_sib_list {edt occ} when inserting the logic test hardware.

  When you specify ijtag_host_interface with a value pointing to an IJTAG host scan interface during process_dft_specification, it adds a TDR that drives the static external controls of the EDT. If you have also specified static external controls that are tied to top level ports or pins, the TDR connections to the EDT static external controls override the specified static external controls.

  When you do not specify this property, it inherits its value from the ijtag_host_interface property found in the EDT wrapper.

- longest_chain_range : *min*, *max* ;

  A required property, for each controller, that specifies the range (min and max values) for the length of the longest scan chain. Use this switch to avoid having to regenerate the logic for minor changes.

- scan_chain_count : *int* ;

  A required property, for each controller, that specifies the number of scan chains for the EDT controller. It provides the size for the scan chain bus ports. The tool reports an error if this property is missing or has no value.

- input_channel_count : *int* ;

  A required property, for each controller, that specifies the number of EDT input channels in the generated EDT controller. It provides the size for the input channel bus port. The tool reports an error if this property is missing or has no value.

- output_channel_count : *int* ;

  A required property, for each controller, that specifies the number of EDT output channels in the generated EDT controller. It provides the size for the output channel bus port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- separate_control_data_channels : on | off ;

  An optional property that specifies whether control data channel separation is used in the generated EDT controller. The default is off. When this property is on, the tool automatically determines the number of control channels and inserts control bits behind the control channels only. This is necessary for channel sharing between non-identical blocks.

- parent_instance : *name* ;

  An optional property that specifies where in the design to insert the EDT controller. If you do not specify it, process_dft_specification creates the controller instance at the top level of the current design.

- leaf_instance_name : *name* ;

  An optional property that specifies the EDT leaf instance name. If you do not specify it, the name is "*<module_name>*_*<dft_spec_id>*_tessent_*<edt_id>*_inst". The *<dft_spec_id>* is the ID specified for the DftSpecification wrapper and *<edt_id>* is the ID specified for the EDT Controller wrapper.

- connect_bscan_segments_to_lsb_chains : on | off | auto ;

  An optional property that specifies whether to connect boundary scan chain segments to the EDT IP automatically if you are inserting boundary scan in the same pass or a previous insertion pass. When this property has a value of auto and a single EDT controller is present in the configuration data, the insertion phase connects the boundary scan chain segments to the EDT controller.

  When boundary scan chain segments are created, the boundary scan instrument dictionary is populated with information about them. When you want to connect the EDT controller to boundary scan chain segments, the insertion phase collects boundary scan chain segment information from the instrument dictionary and uses the data to perform connections. For more information on the instrument dictionary, see get_instrument_dictionary.

  The tool does not permit more than one EDT controller to connect to boundary scan chain segments. When there is more than one EDT controller present, you must specify which EDT controller to connect to the boundary scan chain segments. Set this property to "on" for the EDT controller that you want to connect. When all EDT controllers have this property set to "auto", or when more than one EDT controller has this property set to "on", the tool reports an error during the validation phase.

  The width of the EDT scan input and output bus on the EDT controller module is set with the scan_chain_count property value. If boundary scan chain segments exist and are to be connected to the EDT controller, the width of the bus is incremented by the number of boundary scan segments.

  For example, if you have 100 scan chains and 5 boundary scan chain segments, the resulting bus width on the EDT controller is 105 bits. This ensures there are enough bits when connecting scan chains from scan insertion.

  As suggested by the name for this property, connect_bscan_segments_to_lsb_chains, the boundary scan chain segments are connected to the LSB bits of the EDT scan bus. Each

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

boundary scan chain segment connection is made to the corresponding bit of the bus, so the first segment is connected to the first bit, and so on.

The default value for this property is auto.

- edt_bypass_change_edge_clock : prefer_edt_clock | scan_clock ;

  An optional property that specifies the clock used preferentially for the bypass chain change edge cell. By default, when a chain ends with a trailing edge (TE) and the bypass chain begins with a leading edge (LE), the cell is clocked by the scan clock, which is brought in from the top-level EDT using a dedicated input port. Use this switch to request that the tool preferentially use edt_clock when available.

  The edt_clock may not be available in the following situations:

  - When there are any bypass-mode input change edge cells. In this case, using the edt_clock may lead to clock skew problems and a potential T24 DRC violation.
  - When the output change edge cell becomes a copy cell.
  - When there are bypass lockup cells already driven by the scan clock.

  In these cases, the tool uses the scan clock even when edt_clock is requested, and it reports a warning to notify you that this has occurred.

  The default value for this property is scan_clock.

- LVxMode/present : on | off | auto ;

  An optional property that enables generation of the LVX hardware for the EDT controller. Unless otherwise specified, the default behavior is also to generate the enable_one_chain hardware with the LVX hardware. The default value, “auto”, generates LVX hardware if the LVxMode wrapper is specified.

  For more information, refer to the “Creating EDT With LVX Hardware” section in the Streaming Scan Network (SSN) chapter of the *Tessent Shell User’s Manual*.

- LVxMode/enable_one_chain : on | off ;

  An optional property that enables generation of the enable_one_chain hardware with the LVX hardware. This hardware enables you to observe individual chains when the LVX pattern is applied. Without this hardware, the LVX hardware does not have enough granularity to block individual chains.

  For more information, refer to the “Creating EDT With LVX Hardware” section in the Streaming Scan Network (SSN) chapter of the *Tessent Shell User’s Manual*.

- chain_output_masking_disable : on | off ;

  An optional property that enables generation of a Test Data Register (TDR) that disables EDT chain masking without the need for an extra scan load and pulsing edt_update.

  The default value for this property is “off”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

This example sets connect_bscan_segments_to_lsb_chains to auto, which results in the EDT controller being connected to the boundary scan chain segments.

```
DftSpecification(CoreA,rtl) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Tap(1) {
        HostBscan {}
      }
    }
  }
  BoundaryScan {
    ijtag_host_interface : Tap(1)/HostBscan ;
    max_segment_length_for_logictest : 10 ;
  }
  EDT {
    Controller(c1) {
      longest_chain_range : 100, 110 ;
      scan_chain_count : 100 ;
      input_channel_count : 2 ;
      output_channel_count : 2 ;
      connect_bscan_segments_to_lsb_chains : auto ;
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Interface

This wrapper specifies the port names for the interface of the EDT controller, including IJTAG port names.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      Interface {
        edt_clock            : port_name ; // default: edt_clock
        edt_slave_clock      : port_name ; // default: edt_slave_clock
        edt_update           : port_name ; // default: edt_update
        edt_reset            : port_name ; // default: edt_reset
        edt_channels_in_bus  : bus_name  ; // default: edt_channels_in
        edt_channels_out_bus : bus_name  ; // default: edt_channels_out
        edt_bypass_change_edge_clock : port_name ;
                                  // default: edt_bypass_change_edge_clock
        IjtagScanInterface {
          static_signals_driven : always | as needed // default: as_needed
          tck                 : port_name; // default: ijtag_tck
          reset               : port_name; // default: ijtag_reset
          select              : port_name; // default: ijtag_sel
          capture_en          : port_name; // default: ijtag_ce
          shift_en            : port_name; // default: ijtag_se
          update_en           : port_name; // default: ijtag_ue
          scan_in             : port_name; // default: ijtag_si
          scan_out            : port_name; // default: ijtag_so
        }
        StaticExternalControls {
        }
        LogicBist {
          reset               : port_name; // default: lbist_reset
          enable              : port_name; // default: lbist_en
          prpg_en             : port_name; // default: lbist_prpg_en
          misr_en             : port_name; // default: misr_accumulate_en
          low_power_shift_en : port_name;
                                   // default: lbist_low_power_shift_en
          self_test_en        : port_name; // default: self_test_en
          misr                : port_name; // default: - (not created)
        }
      }
    }
  }
}
```

Description

You can use this wrapper to specify the names of the EDT controller ports. Each property has a default value, as described in the configuration data syntax. These port names are for non-static EDT signals. The IjtagScanInterface wrapper specifies the IJTAG control port names. The StaticExternalControls wrapper specifies static EDT signal port names for the EDT controller. The LogicBist wrapper specifies the EDT LBIST interface port names on a per-controller basis.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Instance names inside the Interface wrapper must start with a letter and consist only of alphanumerics and underscores.

Arguments

- edt_clock : *port_name* ;

  An optional property that specifies the name of the port that supplies a clock to the EDT controller. The default is edt_clock. See the edt_clock entry in Table 3-13.

- edt_slave_clock : *port_pin_name* ;

  An optional property that specifies the name of the EDT controller slave clock port. When you generate level-sensitive clocking (latch-based EDT IP), the EDT IP contains a master clock and a slave clock to connect. The default is edt_slave_clock. See "TestKompress Compression Logic" in the *Tessent TestKompress User's Manual*.

- edt_update : *port_name* ;

  An optional property that specifies name of the update enable port of the EDT controller. The default is edt_update. See the edt_update entry in Table 3-13.

- edt_reset : *port_name* ;

  An optional property that specifies the name of the asynchronous reset port for the EDT controller. This property applies when the controller clocking reset signal is asynchronous. The default is edt_reset.

- edt_channels_in_bus : *bus_name* ;

  An optional property that specifies the name of the input channels bus port for the EDT controller. The default is edt_channels_in.

- edt_channels_out_bus : *bus_name* ;

  An optional property that specifies the name of the output channels bus port for the EDT controller. The default is edt_channels_out.

- edt_bypass_change_edge_clock : *port_name* ;

  An optional property that specifies the name of the input port created when adding retiming cells to the EDT bypass module. The default is edt_bypass_change_edge_clock.

- IjtagScanInterface/static_signals_driven : always | <u>as_needed</u> ;

  An optional property that specifies whether the tool generates TDRs for static control signals inside the EDT. The default value, as_needed, uses the following criteria to determine whether the IJTAG scan interface must be generated:

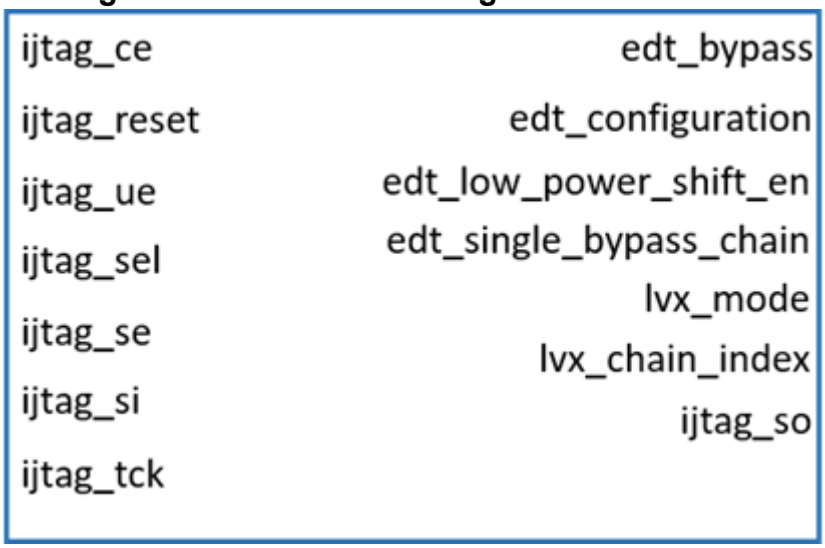
  - When LVX is enabled, the tool generates the IJTAG scan interface and the *_ijtag_registers module for hosting LVX TDRs. Any static control signals continue to come through the input ports of the EDT controller.
  - When LVX is not enabled, the tool does not generate the IJTAG scan interface.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When you set this property to “always”, the tool generates all TDRs, including any static control signals and LVX TDRs, inside the EDT controller with an IJTAG scan interface.

The <*edt_rtl_prefix*>_ijtag_registers module hosts all TDRs and provides output signals for connecting to other modules, as shown in Figure 13-1. When this module is present, there are no input ports on the EDT module for the affected static control signals.

Figure 13-1. Module Hosting TDRs Inside EDT

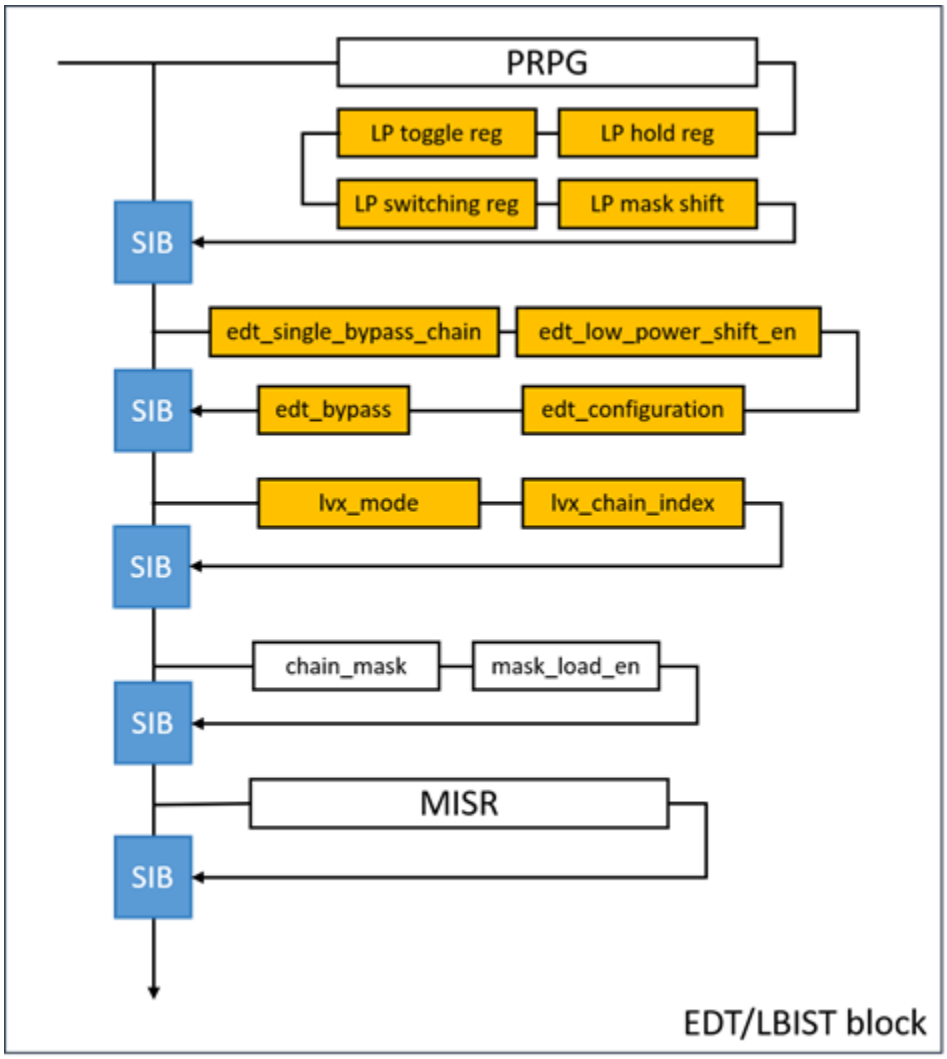


The EDT SIB that hosts the TDRs occurs inside the EDT controller when the tool places the TDRs inside the EDT controller. When LVX is enabled, the tool adds a SIB for the TDRs alongside the SIB for LVX TDRs inside the EDT controller. When LVX is not enabled, the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

tool adds a new SIB inside the <*edt_rtl_prefix*>_ijtag_registers module to host the TDRs. The following examples shows the IJTAG network for a hybrid EDT/LBIST controller:

Figure 13-2. IJTAG Network With TDRs Hosted in a Hybrid EDT/LBIST Controller



- IjtagScanInterface/tck : *port_name*;

  An optional property that specifies the name of the test clock IJTAG control signal. The default is ijtag_tck.

- IjtagScanInterface/reset : *port_name*;

  An optional property that specifies the name of the reset IJTAG control signal. The default is ijtag_reset.

- IjtagScanInterface/select : *port_name*;

  An optional property that specifies the name of the select IJTAG control signal. The default is ijtag_sel.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- IjtagScanInterface/capture_en : *port_name*;

  An optional property that specifies the name of the capture enable IJTAG control signal. The default is ijtag_ce.

- IjtagScanInterface/shift_en : *port_name*;

  An optional property that specifies the name of the shift enable IJTAG control signal. The default is ijtag_se.

- IjtagScanInterface/update_en : *port_name*;

  An optional property that specifies the name of the update enable IJTAG control signal. The default is ijtag_ue.

- IjtagScanInterface/scan_in : *port_name*;

  An optional property that specifies the name of the scan in IJTAG control signal. The default is ijtag_si.

- IjtagScanInterface/scan_out : *port_name*;

  An optional property that specifies the name of the scan out IJTAG control signal. The default is ijtag_so.

- LogicBist/reset : *port_name* ;

  An optional property that specifies the name of an input port that indicates when the synchronous reset of LogicBIST controller occurs. The default is lbist_reset.

- LogicBist/enable : *port_name* ;

  An optional property that specifies the name of an input port that indicates when LogicBIST is enabled. The default is lbist_en.

- LogicBist/prpg_en : *port_name* ;

  An optional property that specifies the name of an input port that indicates when the PRPG is active during the scan load/unload. The default is lbist_prpg_en.

- LogicBist/misr_en : *port_name* ;

  An optional property that specifies the name of an input port that indicates when the MISR is to accumulate the scan unload data. The default is misr_accumulate_en.

- LogicBist/low_power_shift_en : *port_name* ;

  An optional property that specifies the name of an input port that enables low power shift. The default is lbist_low_power_shift_en.

- LogicBist/self_test_en : *port_name* ;

  An optional property that specifies the name of the self-test enable port. This property is only used when the Controller/self_test property is set to on. The default port name is self_test_en.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- LogicBist/lbist_misr : *port_name* ;

  An optional property that specifies the name of the lbist_misr parallel output port. By default, the port is not created if the port_name is “-” or the property is not specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

StaticExternalControls

This wrapper specifies the port names for the static EDT signals on the interface of the EDT controller.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      Interface {
        StaticExternalControls {
          edt_bypass                  : port_name ;
                                       // default: edt_bypass
          edt_single_bypass_chain     : port_name ;
                                       // default: edt_single_bypass_chain
          edt_configuration           : port_name ;
                                       // default: edt_configuration
          edt_low_power_shift_enable  : port_name ;
                                       // default: edt_low_power_shift_en
        }
      }
    }
  }
}
```

Description

This wrapper specifies the names of static EDT signals.

Arguments

- edt_bypass : *port_name* ;

  An optional property that specifies the name of the bypass port for the EDT controller. See "Compression Bypass Logic" in the *Tessent TestKompress User's Manual*. The default is edt_bypass.

- edt_single_bypass_chain : *port_name* ;

  An optional property that specifies the name of the single bypass chain port for the EDT controller. See "Compression Bypass Logic" in the *Tessent TestKompress User's Manual*. The default is edt_single_bypass_chain.

- edt_configuration : *port_name* ;

  An optional property that specifies the EDT configuration port for the EDT controller. See "Dual Compression Configurations" in the *Tessent TestKompress User's Manual*. The default is edt_configuration.

- edt_low_power_shift_enable : *port_name* ;

  An optional property that specifies low-power shift enable port for the EDT controller. See "Low-Power Shift" in the *Tessent TestKompress User's Manual*. The default is edt_low_power_shift_enable.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BypassChains

This wrapper is used to specify the EDT bypass chain configuration in the generated EDT controller.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      BypassChains {
        present                  : on | off ;
        bypass_chain_count       : int | auto ;
        single_bypass_chain      : on | off | auto ;
        BypassChain(id) { // legal: 1..maxposint
          scan_chain_range_list : scan_chain_range, ... ;
        }
      }
    }
  }
}
```

Description

This wrapper is used to specify the EDT bypass chain configuration in the generated EDT controller. For more details, see "Compression Bypass Logic" in the *Tessent TestKompress User's Manual*.

Arguments

- present : on | off ;

  An optional property that specifies whether the bypass chain is present. The default is on.

- bypass_chain_count : *int* | auto ;

  An optional property that specifies the bypass chain count. The default (auto) is the input or output channel count, whichever is smaller. If you generate dual-configuration EDT and it connects to the ScanHost with scan_signals_bypass capability and the use_high_compression_channel_count_in_bypass_mode property is set to "on", the "auto" value is the input or output channel count from the high-compression EDT configuration, whichever is smaller.

- single_bypass_chain : on | off | auto ;

  An optional property that specifies whether a single bypass chain is present. When performing independent TK/LBIST insertion, this property also specifies whether or not to generate new single chain mode logic in EDT.

  The default is auto, with the following behavior:

  - If a LogicBist wrapper is not specified:
    - For the independent TK/LBIST insertion flow, where LogicBist/Options/present is on, this property resolves to on.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Otherwise, it resolves to off.

- If a LogicBist wrapper is specified, this property takes the value of the LogicBist/Controller/SingleChainForDiagnosis/present property.

Note

Even if the SingleChainForDiagnosis/present property is on, you cannot run EDT in single_bypass_chain mode. When you are inserting the LogicBIST IP (hybrid TK/LBIST flow), Tessent Shell concatenates all scan chains across all EDT controllers so that there is one single bypass chain for the hybrid TK/LBIST controller.

- BypassChain(*id*)

  A repeatable wrapper that specifies the bypass chain connections. The *id* value for this wrapper is used to specify the actual bypass chain index. You must specify the index within the bypass chain count limits. You cannot specify twice the same value for a single EDT controller. The *scan_chain_range_list* property inside this wrapper is used to specify the scan chains that create a specific bypass chain.

- BypassChain(*id*)/scan_chain_range_list : *scan_chain_range*, ... ;

  A required property for each BypassChain that specifies a list of scan chains by their index value. These scan chains are used to form the bypass chains.

  The syntax for this property is a list of comma-separated range values. The format for the ranges is "*int*:*int*". The order of the specified scan chains has impact on the order of scan chain connections in the generated EDT IP. The specified scan chain index values must fall within the limits specified by the scan_chain_count property of the EDT controller.

  It is also important to remember that a single scan chain can be assigned to its connection only once and that all scan chains need to have a connection assigned. Otherwise, a validation error is reported.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

This example shows the BypassChains wrapper with the chain connections specified:

```
DftSpecification(CoreA,rtl) {
  EDT {
    Controller(c1) {
      scan_chain_count : 16 ;
      input_channel_count : 2 ;
      output_channel_count : 2 ;
      longest_chain_range : 100, 110 ;
      BypassChains {
        present: on ;
        bypass_chain_count: 2 ;
        BypassChain(1) {
          scan_chain_range_list: 1:2, 4:10 ;
        }
        BypassChain(2) {
          scan_chain_range_list: 3, 11:16 ;
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Compactor

Use this wrapper to add and specify properties for the EDT compactor.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      Compactor {
        type                                : xpress | basic ;
        pipeline_logic_levels_in_compactor : int | off ;
        change_edge_at_compactor_output : any |
            leading_edge_of_edt_clock | trailing_edge_of_edt_clock ;
        CompactorConnection(id) { // legal: 1..maxposint
          scan_chain_range_list : scan_chain_range, ... ;
        }
      }
    }
  }
}
```

Description

This wrapper is used to add and specify properties for the EDT compactor.

Arguments

- type : xpress | basic ;

  An optional property that determines which compactor is used in EDT logic. This is equivalent to the "-COMpactor_type" argument in the set_edt_options command. The possible values are:

  - **xpress** — The Xpress compactor, which increases compression, especially for designs that generate unknown (X) values. This is the default value.
  - **basic** — The basic compactor is the first-generation compactor.

  For more information, refer to "Understanding Compactor Options" in the *Tessent TestKompress User's Manual*.

- pipeline_logic_levels_in_compactor : *int* | off ;

  An optional property that enables pipeline stages. This is equivalent to the "-PIpeline_logic_levels_in_compactor" argument in the set_edt_options command. The possible values are:

  - **off** — Disables pipeline stages in EDT logic. This is the default value.
  - ***<integer>*** — Includes pipeline stages in the EDT logic compactor(s) and specifies the maximum number of logic levels (2-input XOR gates) allowed in the compactors before inserting a pipeline stage.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- change_edge_at_compactor_output

  { any | leading_edge_of_edt_clock | trailing_edge_of_edt_clock }

  An optional property that specifies which edge the compactor output data changes on. This is equivalent to the "-CHange_edge_at_compactor_output" argument in the set_edt_pins command.

  Note

  The change edge of a digital storage element is described relative to its clock and can be Leading Edge (LE) or Trailing Edge (TE).

  During EDT generation with the DftSpecification, all chains are assumed to start with LE timing and end with TE timing relative to their shift clock.

  If you specify the trailing edge, the tool adds a LE-clocked pipeline stage followed by a TE-clocked lockup cell by default.

  If you specify the leading edge, the tool adds a LE-clocked pipeline stage.

- CompactorConnection(*id*)

  A repeatable wrapper that specifies compactor connections.

  The *id* value of this wrapper is used to specify the actual EDT output channel index. You must specify an index within the EDT output channel count limits. You cannot specify the same value twice for a single EDT controller. The *scan_chain_range_list* property inside this wrapper is used to specify the scan chains that are connected to the EDT output channel.

- CompactorConnection(*id*)/scan_chain_range_list : *scan_chain_range*, ... ;

  A required property for each CompactorConnection that specifies a list of scan chains by their index values. Use this property to specify which scan chains are compacted into which EDT channel.

  The syntax for the property value is a list of comma-separated range values. The ranges are of the format "*int*:*int*". The order of the specified scan chains impacts the order of scan chain connections in the generated EDT IP. The specified scan chain index values must be within the limits specified by the scan_chain_count property of the EDT controller.

  Note

  A single scan chain can be assigned to its connection only once, and all scan chains need to have a connection assigned. Otherwise, a validation error is reported.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

This example shows the specification wrapper with the compactor connections specified:

```
DftSpecification(CoreA,rtl) {
  EDT {
    Controller(c2) {
      scan_chain_count : 16 ;
      input_channel_count : 2 ;
      output_channel_count : 2 ;
      longest_chain_range : 100, 110 ;
      Compactor {
        type: xpress ;
        CompactorConnection(1) {
          scan_chain_range_list: 1:2, 4:10 ;
        }
        CompactorConnection(2) {
          scan_chain_range_list: 3, 11:16 ;
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Clocking

A wrapper that specifies the information used by the EDT controller clock.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      Clocking {
        type           : edge | level ;
        lockup_cells   : on | off ;
        reset_signal   : asynchronous | off | ijtag_reset ;
        reset_polarity : auto | active_low | active_high ;
      }
    }
  }
}
```

Arguments

- type : edge | level ;

  An optional property that determines what type of architecture logic EDT uses. The default is edge. This property is equivalent to the "-CLocking" argument in the set_edt_options command. The following are the possible values:

  - **Edge** — Specifies DFF-based logic. You can use DFF-based logic for designs that are LSSD-based, mux-DFF-based, or that incorporate a mixture of these two architectures.
  - **Level** — Specifies latch-based logic. This option is for pure LSSD designs only.

  Restriction

  An error message is issued if you specify the type as level when the EDT controller is connected to an SSH.

- lockup_cells : on | off ;

  An optional property that determines whether EDT logic uses lockup cells. The default is on. This property is equivalent to the "-LOCKup_cells" argument in the set_edt_options command.

  Caution

  It is recommended to leave this property set to *on*. Turning it off can lead to DRC violations during pattern generation and the inability to generate valid patterns. For more information, see "Understanding Lockup Cells" in the *Tessent TestKompress User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following are the possible values:

- **ON** — Includes lockup cells in the EDT logic as needed between the decompressor and the scan chains, between the scan chains and the compactors (if pipelining is used), and between scan chains when they are concatenated for bypass mode.
- **OFf** — Disables lockup cells in the EDT logic.

- reset_signal : asynchronous | off | ijtag_reset ;

  An optional property that specifies whether the EDT logic is to include an asynchronous reset signal, edt_reset. In most cases, you do not need this signal, but you can use it if design requirements dictate that all sequential elements in the design are resettable. This is equivalent to the "-RESet_signal" argument in the set_edt_options command. The following are the possible values:

  - **OFf** — Disables the asynchronous reset. This is the default.
  - **Asynchronous** — Includes an asynchronous reset signal for all sequential elements in the EDT logic and a new top-level pin named "edt_reset" to control it. You can rename this pin or share it with a functional pin in the design core.

    When you specify reset as asynchronous, it is described as a DataInPort in ICL, and you can use the edt_reset core instance parameter to assert the reset signal.
  - **ijtag_reset** — For designs with IJTAG, you can use this setting to connect the edt_reset signal to ijtag_reset, thus avoiding the need for a separate top-level reset port. This applies to both EDT and hybrid EDT/LBIST instruments. This option is not supported for type-3 LPCTs, because type-3 LPCT controllers are not IJTAG compatible.

    When you connect edt_reset to ijtag_reset, the tool infers the connection as follows:

    - EDT IP — If the controller has a valid ijtag_host_interface, connect edt_reset to the ijtag_reset specified by that interface. When ijtag_host_interface is "none" (even when the design has an IJTAG network), the existing Connections/ edt_reset property specifies the connection. The edt_reset signal uses the active low polarity from the ijtag_host_interface unless the reset_polarity value specifies active_high.
    - Hybrid EDT/LBIST IP — The top-level EDT and LBIST modules do not have edt_reset pins; instead, they connect internally to the ijtag_reset pin already present.

    When you specify reset as ijtag_reset, it is connected to ijtag_reset, so the reset is automatic, and you cannot influence it outside of calls to iReset.

- reset_polarity : auto | active_low | active_high ;

  An optional property that specifies the polarity for the asynchronous reset signal or ijtag_reset. A value of "auto" (default) resolves to active_high for an asynchronous reset_signal and to active_low for an ijtag_reset reset signal.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can specify active_high for an ijtag_reset reset signal only if the ijtag_host_interface value is “none.” If you specify an ijtag_host_interface, the tool prefers that interface over a custom connection. Specifying active_high in this case causes the tool to report an error.

Specifying an asynchronous active_low EDT reset with LBIST is not supported. Doing so causes the tool to report an error.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

HighCompressionConfiguration

This wrapper is used to specify high-compression properties for the EDT controller.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      HighCompressionConfiguration {
        present              : on | off ;
        input_channel_count  : int  ;
        output_channel_count : int  ;
      }
    }
  }
}
```

Description

Use this wrapper to specify the presence of the high-compression configuration for the EDT controller and the input and output channel counts for the high-compression configuration. For more information, see "Dual Compression Configurations" in the *Tessent TestKompress User's Manual*.

Arguments

- present : on | off ;

  An optional property that, when set to on, specifies that the high-compression EDT configuration is available for the EDT controller. When this property is set to on, the -parameter switch of the add_core_instances command enables a high_compression or low_compression value for the edt_configuration core instance parameter. The default is off.

- input_channel_count : *int* ;

  A required property for a present high-compression configuration that specifies the input channel count for the high compression configuration of the EDT controller.

- output_channel_count : *int* ;

  A required property for a present high-compression configuration that specifies the output channel count for the high compression configuration of the EDT controller.

Examples

This example shows the DftSpecification of an EDT controller. It uses a HighCompressionConfiguration wrapper, highlighted in green. The specified settings result in the generation of a second EDT configuration with one input channel and one output channel. This is in addition to the low compression configuration defined in the Controller wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(CoreA,rtl) {
  EDT {
    Controller(c1) +{
      scan_chain_count : 16 ;
      input_channel_count : 2 ;
      output_channel_count : 2 ;
      longest_chain_range : 100, 110 ;
      BypassChains {
        present: on ;
        single_bypass_chain: on ;
      }
      HighCompressionConfiguration {
        present               : on ;
        input_channel_count   : 1  ;
        output_channel_count  : 1  ;
      }
      ShiftPowerOptions {
        present               : on ;
      }
      Connections +{
        StaticExternalControls {
          edt_bypass:                  top_edt_bypass ;
          edt_single_bypass_chain:     top_edt_single_bypass_chain ;
          edt_configuration:           top_edt_configuration ;
          edt_low_power_shift_enable:  top_edt_low_power_shift_enable ;
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ShiftPowerOptions

This wrapper adds and specifies properties for the EDT shift power options.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      ShiftPowerOptions {
        present                              : on | off ;
        full_control                         : on | off ;
        min_switching_threshold_percentage : int  ; // default: 15
      }
    }
  }
}
```

Arguments

- present : on | off ;

  An optional property that specifies whether the EDT low-power controller is present. This is equivalent to the set_edt_power_controller command specifying None (off) or Enabled or Disabled (on). The default is off.

- full_control : on | off ;

  An optional property that adds one bit per scan chain to control whether that chain loads a constant value or not. This provides complete control over the scan chain input masking. In addition, when this property is on, any value specified with the min_switching_threshold_percentage property is ignored. This is equivalent to the "-Full_control" option in the set_edt_power_controller command. The default is off.

  Note

  This property does not enable you to explicitly specify the chain or chains to be controlled. The "on" value instructs the tool to insert hardware that enables the ATPG to have more fine-grained control over power thresholds. You must still specify your switching threshold with the set_power_control command, but with "full_control : on", the ATPG may be able to more closely meet the specified thresholds. Note that selecting "-full_control on" increases the size of the low power controller logic.

- min_switching_threshold_percentage : *int* ;

  An optional property that specifies the minimum switching threshold value used to determine power controller parameters, such as input shift register size. Valid integer values are 1 to 50. If an integer greater than 50 is specified, the tool uses 50. This is equivalent to the "-MIN_Switching_threshold_percentage" option in the set_edt_power_controller command. The default value is 15, indicating 15 percent.

  The tool uses this value to calculate the size of the power_controller. The power controller must be large enough to control the number of scan chains required to achieve the minimum

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

switching threshold specified. For more information, see "Low-Power Shift and Switching Thresholds" in the *Tessent TestKompress User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

LogicBistOptions

This wrapper groups the LogicBist-related options set at the EDT controller level.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      LogicBistOptions {
        present              : on | off | auto ;
        misr_input_ratio     : integer | auto ;
        chain_mask_register_ratio : integer ; // default: 1
        prpg_seed            : hex_value ;      // default: 1
        ShiftPowerOptions {
        }
      }
    }
  }
}
```

Description

You use this wrapper to set LogicBIST-related options for the EDT controller.

Arguments

- present : on | off | auto ;

  An optional property that specifies the generated EDT controller is a hybrid TK/LBIST controller. The default, auto, indicates that when the DftSpecification/LogicBist wrapper is used, this property is set to “on” and hybrid controllers are generated.

- misr_input_ratio : *integer* | auto ;

  An optional property that specifies the ratio of the number of scan chains to the number of MISR inputs. The specified value should be a power of 2. If the value is 1, then all scan chain outputs are connected to the MISR. If the value is greater than or equal to #scan_chains/#channels, then all EDT channel outputs are connected to the MISR. If any value in between is used, then taps are taken from the middle of the EDT compactor XOR tree corresponding to the specified value.

  The default value is auto. Setting this property to auto reverts the input ratio to the default 1 value.

  This property only specifies the number of MISR inputs; the size of the MISR is implicitly determined based on this input. The minimum size of the MISR is at least 24 bits. If this property is not specified, a default MISR of 24 bits is used. When the design has more than 24 output channels, the default MISR size is increased to be greater than or equal to the number of output channels.

  This is equivalent to the “-lbist_misr_input_ratio” argument in the set_edt_options command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- chain_mask_register_ratio : *integer* :

  An optional property that specifies the number of scan chains controlled by a single EDT chain mask register bit. By default, the ratio is 1:1; there are as many bits in the chain mask register as there are number of scan chains.

  You can use this property to minimize the area impact of the chain mask register. For example, specifying 3 for *integer* means a 3:1 ratio with each chain mask register bit controlling 3 scan chains. Specify the ratio on a per EDT block basis.

  The larger the ratio you specify, the more you trade off resolution for area impact. When any scan chain connected to a chain mask register bit is masked, all the other scan chains connected to the bit are also masked automatically.

  This is equivalent to the "-chain_mask_register_ratio" argument in the set_edt_options command.

  Refer to "Scan Chain Masking" in the *Hybrid TK/LBIST User's Manual* for more information.

- prpg_seed : *hex_value* ;

  An optional property that specifies a hexadecimal number used to overwrite the prpg_reference_seed property value specified with the global Edt/LogicBistOptions wrapper. The default is 1.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ShiftPowerOptions

This wrapper groups properties related to Low Power Shift capabilities.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      LogicBistOptions {
        ShiftPowerOptions {
          present           : off | on ;
          default_operation : enabled | disabled ;
          SwitchingThresholdPercentage{
          }
        }
      }
    }
  }
}
```

Description

This wrapper enables you to set low power shift options for the hybrid TK/LBIST controller.

Arguments

- present : off | on | auto ;

  An optional property that enables the low power shift functionality for the hybrid TK/LBIST controller and synthesizes the low power controller. The auto setting resolves to on when the DFT signal lbist_low_power_shift_en exists and its source_node from the port is properly defined; otherwise it resolves to off. The default is auto.

  When “present” is set to on and the lbist_low_power_shift_en does not exist, the signal is automatically inferred on port when the current design level is sub-block or physical block, and inferred on buffer (tied to zero) when the current design level is chip.

  This is equivalent to the “set_lbist_power_controller_options none” command.

- default_operation : enabled | disabled ;

  An optional property that specifies whether low power shift should be enabled or disabled in the default hardware mode.

  This is equivalent to the “set_lbist_power_controller_options {enabled | disabled}” command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SwitchingThresholdPercentage

This wrapper enables you to set the switching threshold percentage for the hybrid TK/LBIST controller.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      LogicBistOptions {
        ShiftPowerOptions {
          SwitchingThresholdPercentage {
            hardware_default : integer ; //default: 15
          }
        }
      }
    }
  }
}
```

Arguments

- hardware_default : *integer* ;

  An optional property that specifies the switching threshold value for the Hybrid TK/LBIST controller hardware mode. Valid values are between 1 and 50. The default is 15.

  This is equivalent to the "set_lbist_power_controller_options -min_switching_threshold_percentage" command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Connections

This wrapper is used to add and specify properties for the EDT controller connections.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      Connections {
        edt_clock       : port_pin_name | DftSignal(edt_clock) |
                          OptionalDftSignal(edt_clock) ;
        edt_slave_clock : port_pin_name ; // default: edt_slave_clock
        edt_update      : port_pin_name | DftSignal(edt_update) |
                          OptionalDftSignal(edt_update) ;
        edt_reset       : port_pin_name ;
        ssh_chain_group : chain_group_id ;
        mode_enables    : port_pin_name |
                          DftSignal(scan_mode_dft_signal), ... ;
        edt_bypass_change_edge_clock : occ_spec ;
        StaticExternalControls {
        }
        EdtChannelsIn(range) {
        }
        EdtChannelsOut(range) {
        }
      }
    }
  }
}
```

Description

This wrapper is used to add and specify properties for the EDT controller connections.

There are cases when you may need to connect multiple EDT channel connections to a single top level port. A method for doing this, along with an example, is described in the documentation for the add_dft_modal_connections command.

If you are using SSN with a ScanHost node, the multiplexing between an internal and an external EDT controller is automated. You simply need to use the mode_enables property to specify one or more DFT signals of type scan_mode to identify all the scan modes using each EDT controller. If the ScanHost node has more than one ChainGroup wrapper, you need to use the ssh_chain_group property to specify which one to use for the EDT controller as in Example 1 of the ScanHost section.

Arguments

- edt_clock : *port_pin_name* | DftSignal(edt_clock) | OptionalDftSignal(edt_clock);

  An optional property used to specify a source node to connect the edt_clock port to.

  The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the set_design_level command,

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is not "chip" and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it is created as an input port during insertion.

When design level is "chip", specified port names must already be connected to a pad cell. The connection is automatically made to the internal side of the input pad buffer. You can specify OptionalDftSignal(edt_clock) or DftSignal(edt_clock) as the value, and it is interpreted to request a connection to the DFT signal edt_clock as it was added using the add_dft_signals command. When using OptionalDftsignal(edt_clock), you do not get an error if the DFT signal edt_clock was not added; a connection to the port called "edt_clock" is made instead and the port is created if not present unless design level is equal to "chip" in which case you get an error.

When the property is unspecified, its value is inherited from the ../Connections/edt_clock property, which itself defaults to OptionalDftSignal(edt_clock) when unspecified.

When the SSN wrapper is present and it contains a ScanHost wrapper, the DFT signal edt_clock gets mapped to the edt_clock output port of the ScanHost module. If you keep the default value of this property, which is OptionalDftsignal(edt_clock), the edt_clock port of the EDT controller will be sourced by the edt_clock port of the ScanHost module automatically.

- edt_slave_clock : *port_pin_name* ;

  An optional property that specifies the EDT slave clock. When you generate level-sensitive clocking (latch-based EDT IP), the EDT IP contains a master clock and a slave clock to connect. See "TestKompress Compression Logic" in the *Tessent TestKompress User's Manual*. The default is edt_slave_clock.

- edt_update : *port_pin_name* | DftSignal(edt_update) | OptionalDftSignal(edt_update) ;

  An optional property used to specify a source node to connect the edt_update port to.

  The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the set_design_level command, is not "chip" and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it is created as an input port during insertion.

  When design level is "chip", specified port names must already be connected to a pad cell. The connection is automatically made to the internal side of the input pad buffer. You can specify OptionalDftSignal(edt_update) or DftSignal(edt_update) as the value, and it is interpreted to request a connection to the DFT signal edt_update as it was added using the add_dft_signals command. When using OptionalDftsignal(edt_update), you do not get an error if the DFT signal edt_update was not added; a connection to the port called "edt_update" is made instead and the port is created if not present unless design level is equal to "chip" in which case you get an error.

  When the property is unspecified, its value is inherited from the ../Connections/edt_update property, which itself defaults to OptionalDftSignal(edt_update) when unspecified.

  When the SSN wrapper is present and it contains a ScanHost wrapper, the DFT signal edt_update gets mapped to the edt_update output port of the ScanHost module. If you keep the default value of this property which is OptionalDftsignal(edt_update), the edt_update

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

port of the EDT controller will be sourced by the edt_update port of the ScanHost module automatically.

- edt_reset : *port_pin_name* ;

  An optional property that specifies an asynchronous reset signal. The default is edt_reset.

- mode_enables : *port_pin_name* | DftSignal(scan_mode_dft_signal), ...;

  An optional property that specifies one or more mode enables to identify all the scan modes using each EDT controller. If you are using an EDT controller for internal scan modes and a smaller one for the external scan modes, the multiplexing between an internal and an external EDT controller is automated. You can point to a pin or a port objects or to a DFT signal of type scan mode using the DftSignal(scan_mode_dft_signal) symbolic name.

  When you are not using SSN, you must specify the Connections/EdtChannelsOut wrapper to specify a common EDT channel output connection points for both EDT controllers.

- ssh_chain_group : *chain_group_id*;

  An optional property that is used when the DftSpecification includes the SSN wrapper with a ScanHost wrapper. When using SSN, the EDT channels terminates on the ScanHost node. If the ScanHost node has more than one ChainGroup wrapper, you need to use the ssh_chain_group property to specify which one to use for the EDT controller as in Example 1 of the ScanHost section.

- edt_bypass_change_edge_clock : *occ_spec* ;

  An optional property that specifies the OCC to use for the clock driving cells in the EDT bypass module. The <*occ_spec*> is a Tessent OCC specification that can include the following:

  a. OCC instance pathnames

  b. OCC module names

  c. OCC core names used in the TCD

  d. OCC controller IDs generated in the same pass

  If you do not specify this property when the EDT has bypass retiming cells and only one OCC is available, that OCC drives the retiming cells. The tool resolves ambiguity (for example, if a module and a top-level instance have the same name) by using the order in the preceding list. Avoid ambiguity by fully specifying instance pathnames. You can specify the OCC generated in the same pass only by using its ID as Occ(<*id*>).

  Refer to the Examples section to see a sample usage of this property.

Examples

The following shows a specification with example OCC and the associated edt_bypass_change_edge_clock properties.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Occ {
  ijtag_host_interface : Sib(occ);
  include_clocks_in_icl_model : yes;
  Controller(clk_1) {
    clock_intercept_node : my_clk;
  }
}

Edt {
  ...
  Controller(c1_int) {
    ...
    Interface {
      edt_bypass_change_edge_clock : my_edt_bypass_change_edge_clock;
    }
    ...
    Connections {
      edt_bypass_change_edge_clock : coreC_occ_insertion0_tessent_occ_clk_2_inst;
    }
  }
  Controller(c2_int) {
    ...
    Interface {
      edt_bypass_change_edge_clock : my_edt_bypass_change_edge_clock_2;
    }
    ...
    Connections {
      edt_bypass_change_edge_clock : Occ(clk_1);
                                // specify connection to the OCC from the same pass
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

StaticExternalControls

This wrapper is used to add and specify properties for the EDT static external controls.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      Connections {
        StaticExternalControls {
          edt_bypass                 : port_pin_name | 0 | 1 ;
          edt_single_bypass_chain    : port_pin_name | 0 | 1 ;
          edt_configuration          : port_pin_name | 0 | 1 ;
          edt_low_power_shift_enable : port_pin_name | 0 | 1 ;
        }
      }
    }
  }
}
```

Description

This wrapper is used to add and specify properties for the EDT static external controls.

Arguments

- edt_bypass : *port_pin_name* | 0 | 1 ;

  An optional property that specifies the connection to the edt_bypass EDT static control pin. The edt_bypass pin connects to the specified port_pin_name, or is tied to high or low using the 1 or 0 symbol. Setting this property overrides the EDT/Connections/StaticExternalControls/edt_bypass specification. The default is 0.

- edt_single_bypass_chain : *port_pin_name* | 0 | 1 ;

  An optional property that specifies the connection to the edt_single_bypass_chain static EDT control pin. The edt_single_bypass_chain pin connects to the specified port_pin_name, or is tied to high or low using the 1 or 0 symbol. Setting this property overrides the EDT/Connections/StaticExternalControls/edt_single_bypass_chain specification. The default is 0.

- edt_configuration : *port_pin_name* | 0 | 1 ;

  An optional property that specifies the connection to the edt_configuration static EDT control pin. The edt_configuration pin connects to the specified port_pin_name, or is tied to high or low using the 1 or 0 symbol. Setting this property overrides the EDT/Connections/StaticExternalControls/edt_configuration specification. The default is 0.

- edt_low_power_shift_enable : *port_pin_name* | 0 | 1 ;

  An optional property that specifies the connection to the edt_low_power_shift_enable static EDT control pin. The edt_low_power_shift_enable pin connects to the specified port_pin_name, or is tied to high or low using the 1 or 0 symbol. Setting this property

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

overrides the EDT/Connections/StaticExternalControls/edt_low_power_shift_enable specification. The default is 1.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

EdtChannelsIn

This wrapper is used to add and specify properties for the EDT channel in ports.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      Connections {
        EdtChannelsIn(range) {
          port_pin_name      : port_pin_name ;
          pipeline_clock     : port_pin_name | DftSignal(edt_clock) |
                             | DftSignal(shift_capture_clock) ;
                        // default: Inherited from Connections/edt_clock
          insert_lockup_cell : auto | off;
          lockup_cell_type   : latch | dff;
          PipelineStage {
          }
        }
      }
    }
  }
}
```

Description

The port_pin_name and the pipeline_clock properties are not used when you are using SSN with a ScanHost wrapper present. You will get an error in that case if you try to specify those properties. The source of the EDT channel input will be the to_scan_in ports of the ScanHost node. You need to specify the ssh_chain_group property if the ScanHost node was build with more than one chain group as in "Example 1" on page 4523 of the ScanHost section.

The process_dft_specification command for EDT creates bus ports for EDT channel connections on the block level as specified in the configuration data. If the bus ports exist already, they are used for EDT channel connections. If they do not exist, they are created on the block level by process_dft_specification.

You specify the bus names with the port_pin_name properties with this configuration data wrapper. The ports must have sufficient bits to make all of the specified EDT channel input connections. EDT process_dft_specification cannot enhance existing ports with additional bits.

You can specify the bus ports to be created with any bit range as long as it is a valid Verilog range. They can start with 0 or any other number. Any range is acceptable as long as the EDT channel range width matches the specified connection width. The order you specify the bus ports has an impact on the connection order made to the EDT IP.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The *(range)* component of the EdtChannelsIn wrapper accepts a range of positive integer values using a single value or a colon-separated pair of values. For example:

```
EdtChannelsIn(4:1)  – first 4 EDT input channels
EdtChannelsIn(1:4)  – first 4 EDT input channels with inverted connection
                      order
EdtChannelsIn(1)    – first EDT input channel
EdtChannelsIn(1:1)  – first EDT input channel
```

When you specify at least one input connection, you must specify all values and ranges.

Note

The EDT process_dft_specification does not expand existing bus ports. An error is reported if you use more bits than are on the existing bus.

Arguments

- port_pin_name : *port_pin_name* ;

  A required property that specifies signals that are connected to the edt_channels_in port of the EDT controller. The value can be any of these:

  - bus
  - list of scalar ports/pins
  - list of mixed bus and port/pin objects
  - indexed identifier

- pipeline_clock : *port_pin_name* | DftSignal(edt_clock) | DftSignal(shift_capture_clock) ;

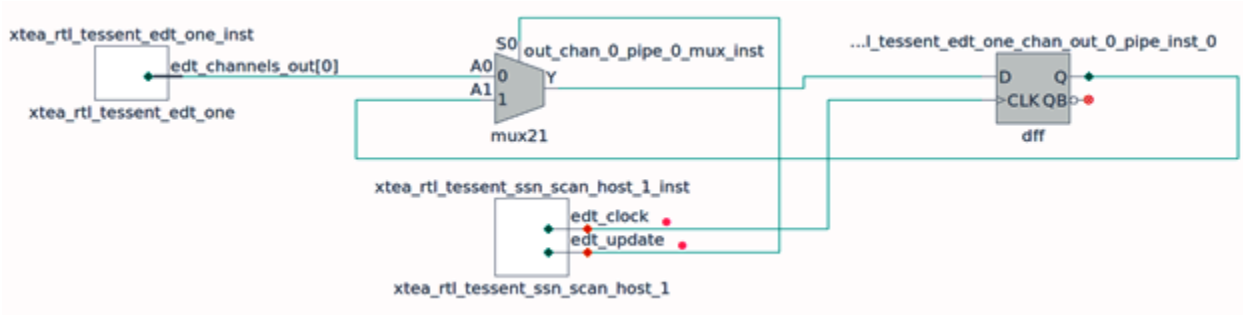
  An optional property that specifies the pipeline clock. The default is inherited from the edt_clock property in the Connections wrapper, which defaults to OptionalDftSignal(edt_clock). See "Pipeline Stage Insertion" and "Lockup Cells on the Input Side of the EDT Controller" in the *Tessent TestKompress User's Manual* for more information.

  If you specify DftSignal(edt_clock) or DftSignal(shift_capture_clock), the tool inserts "hold-mux" logic on the pipelines for the specified clock (analogous to a hold scan cell). If the value is DftSignal(shift_capture_clock), the tool inserts hold logic. For DftSignal (edt_clock), the tool uses a different hold logic that enables the EDT bypass mode to also work correctly. During EDT bypass, you must remember to pulse the edt_clock when you perform bypass. The hold-mux logic for the edt_clock uses the edt_update signal to ensure that cells hold their value during the load_unload phase, because the pipelines are included

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

with the scan chains. See the following figure for an example of hold-mux logic modified for using the edt_clock to drive the pipelines:

Figure 13-3. Hold-Mux Logic for edt_clock



One reason to use edt_clock rather than shift_capture_clock for EDT is to minimize transitions between clocks. For example, you may have EDT input channel pipelines that use the shift_capture_clock followed by the compactor, which uses edt_clock; and then the decompressor, which uses edt_clock; followed by output pipes using the shift_capture_clock again.

Note

The EdtChannelsIn/PipelineStage/pipeline_clock value, if specified, overrides this value.

- insert_lockup_cell : auto | off ;

  An optional property that specifies whether the tool inserts lockup cells. By default, the tool inserts lockup cells as needed when the source and destination clocks overlap as follows:

  - Both clocks have identical waveform timing within a tester cycle; clocks are on at the same time and their edges are aligned.
  - The active edge of the destination clock occurs later in the cycle than the active edge of the source clock.

  When clocks do not overlap, the timing sequence protects the data, so no lockup cells are needed.

  When this property is “off”, the tool does not insert lockup cells.

Caution

You should only turn off lockup cells in the following circumstances:

- Between two pipeline stages that are driven by two different clocks when those clocks are known to be synchronous during shift.
- After the last channel input pipeline, when the clock driving the last pipeline is synchronous with the EDT clock driving the decompressor *and* is also synchronous with the clock driving the first scan cells in bypass mode (if present in EDT), and the first cells are driven by the LE of their clocks.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

When this property is set to auto, the tool automatically omits the lockup cell after the last channel output pipeline stage when the design level is “chip”. The lockup cell is unnecessary in this case, and omitting it improves timing.

- lockup_cell_type : latch | dff ;

  An optional property that specifies whether to use latch or DFF lockup cells. By default, the tool uses latches for lockup cells.

  When you specify DFF lockup cells, the tool first attempts to use negedge DFFs. If these are not available, it then attempts to use posedge DFFs and clock inverters. If no clock inverter is available, the tool uses regular inverters and reports a warning that negedge DFFs and clock inverters are not available.

Examples

Example 1

In this example, the tool connects the first four EDT input channels to the *top_bus[3:0]* port object. The first EDT channel connects to *top_bus[0]* and the fourth channel connects to *top_bus[3]*.

```
EdtChannelsIn(4:1) {
  port_pin_name : top_bus[3:0];
}
```

Example 2

This example demonstrates using a mixed list of scalar ports and buses.

```
EdtChannelsIn(8:5) {
  port_pin_name : top8, top7, in[1:0];
}
```

The tool makes connections as follows:

```
edt_ctrl_inst/edt_channels_in[7] -> top8
edt_ctrl_inst/edt_channels_in[6] -> top7
edt_ctrl_inst/edt_channels_in[5] -> in[1]
edt_ctrl_inst/edt_channels_in[4] -> in[0]
```

Example 3

This example uses an indexed bus identifier.

```
EdtChannelsIn(4:1) {
  port_pin_name : in[%d];
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The tool automatically infers the bus bit values for the connection as follows:

```
edt_ctrl_inst/edt_channels_in[3] -> in[3]
edt_ctrl_inst/edt_channels_in[2] -> in[2]
edt_ctrl_inst/edt_channels_in[1] -> in[1]
edt_ctrl_inst/edt_channels_in[0] -> in[0]
```

Example 4

This example shows how to create two EDT controllers. While it shows the creation of both input and output bus ports, notice the effect of EdtChannelsIn.

```
DftSpecification(CoreB,rtl) {
    EDT {
      Controller(1) {
        scan_chain_count: 100;
        longest_chain_range: 100,110;
        input_channel_count: 4;
        output_channel_count: 4;
        leaf_instance_name: edt_ctrl_inst1;
        Connections {
           EdtChannelsIn(4:1) {
              port_pin_name: new_input_bus[3:0];
           }
           EdtChannelsOut(4:1) {
              port_pin_name: new_output_bus[3:0];
           }
        }
      }
      Controller(2) {
        scan_chain_count: 100;
        longest_chain_range: 100,110;
        input_channel_count: 4;
        output_channel_count: 4;
        leaf_instance_name: edt_ctrl_inst2;
        Connections {
           EdtChannelsIn(4:1) {
              port_pin_name: new_input_bus[15:12];
           }
           EdtChannelsOut(4:1) {
              port_pin_name: new_output_bus[15:12];
           }
        }
      }
    }
  }
```

With this configuration data in place, two new EDT controllers are created when you run the process_dft_specification command, edt_ctrl_inst1 and edt_ctrl_inst2. The input EDT channels

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

in each controller are connected to the created bus ports named “new_input_bus” by the EdtChannelsIn wrapper. This shows the connections made for both controllers:

```
edt_ctrl_inst1/edt_channels_in[3] -> new_input_bus[3]
edt_ctrl_inst1/edt_channels_in[2] -> new_input_bus[2]
edt_ctrl_inst1/edt_channels_in[1] -> new_input_bus[1]
edt_ctrl_inst1/edt_channels_in[0] -> new_input_bus[0]
edt_ctrl_inst2/edt_channels_in[3] -> new_input_bus[15]
edt_ctrl_inst2/edt_channels_in[2] -> new_input_bus[14]
edt_ctrl_inst2/edt_channels_in[1] -> new_input_bus[13]
edt_ctrl_inst2/edt_channels_in[0] -> new_input_bus[12]
```

The new bus ports have a range of 16 bits (15:0) each because the MSB specified is 15 and the LSB is 0 for both of them. The tool always creates every bus port from its MSB to LSB bits as specified in the configuration data.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

PipelineStage

This wrapper is used to add and specify properties for the EDT channel in pipeline stage.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      Connections {
        EdtChannelsIn(id) {
          PipelineStage {
            parent_instance    : instance_name ;
            leaf_instance_name : instance_name ;
            pipeline_clock     : port_pin_name ;
                               // default: Inherited from EdtChannelsIn value
            insert_lockup_cell : auto | off;
                               // default: Inherited from EdtChannelsIn value
            lockup_cell_type   : latch | dff;
                               // default: Inherited from EdtChannelsIn value
          }
        }
      }
    }
  }
}
```

Description

This wrapper is used to add and specify properties for the EDT channel in pipeline stage. See "Pipeline Stage Insertion" and "Lockup Cells on the Input Side of the EDT Controller" in the *Tessent TestKompress User's Manual* for more information.

Note

For SSN, additional muxing logic is added to EDT pipeline stages to avoid capture mismatches. Affected pipeline stages are also clocked by the shift_capture_clock output of the corresponding SSH instance instead of the edt_clock. These hardware modifications are only included when using the following configuration:

- The EDT controller is connected to an SSH.
- The EDT input or output channels require one or more pipeline stages.
- The EDT controller has a bypass mode included.

Arguments

- parent_instance : *instance_name* ;

  An optional property that specifies the parent instance. This is the name of the instance where the pipeline IP is generated. If it is not specified, the IP is placed in the current design scope.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- leaf_instance_name : *instance_name* ;

  An optional property that specifies the instance name of the pipeline IP. If not specified, the leaf instance name is auto generated with the following format:

  ```
  <top_module_name>_<dft_spec_id>_tessent_<instrument_name>_
  <edt_ctrl_id>_<channel_type>_<channel_bit>_pipe_inst_<pipe_index>
  ```

- pipeline_clock : *port_pin_name* ;

  An optional property that specifies the pipeline clock port pin name. This property overrides the value of the pipeline_clock specified in the EdtChannelsIn wrapper. If not specified, it inherits the value from the EdtChannelsIn wrapper. See "Pipeline Stage Insertion" and "Lockup Cells on the Input Side of the EDT Controller" in the *Tessent TestKompress User's Manual* for more information.

  If you specify DftSignal(edt_clock) or DftSignal(shift_capture_clock), the tool inserts hold-mux logic on the pipelines for the specified clock. The hold-mux logic for the edt_clock uses the edt_update signal to ensure that cells hold their value during the load_unload phase, because the pipelines are included with the scan chains.

- insert_lockup_cell : auto | off ;

  An optional property that specifies whether the tool inserts lockup cells. By default, the tool inserts lockup cells as needed when the source and destination clocks overlap as follows:

  - Both clocks have identical waveform timing within a tester cycle; clocks are on at the same time and their edges are aligned.
  - The active edge of the destination clock occurs later in the cycle than the active edge of the source clock.

  When clocks do not overlap, the timing sequence protects the data, so no lockup cells are needed.

  When this property is "off", the tool does not insert lockup cells.

  Caution

  You should only turn off lockup cells in the following circumstances:

  - Between two pipeline stages that are driven by two different clocks when those clocks are known to be synchronous during shift.
  - After the last channel input pipeline, when the clock driving the last pipeline is synchronous with the EDT clock driving the decompressor *and* is also synchronous with the clock driving the first scan cells in bypass mode (if present in EDT), and the first cells are driven by the LE of their clocks.

  Note

  When this property is set to auto, the tool automatically omits the lockup cell after the last channel output pipeline stage when the design level is "chip". The lockup cell is unnecessary in this case, and omitting it improves timing.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- lockup_cell_type : latch | dff ;

  An optional property that specifies whether to use latch or DFF lockup cells. By default, the tool uses latches for lockup cells.

  When you specify DFF lockup cells, the tool first attempts to use negedge DFFs. If these are not available, it then attempts to use posedge DFFs and clock inverters. If no clock inverter is available, the tool uses regular inverters and reports a warning that negedge DFFs and clock inverters are not available.

Examples

The following example uses the PipelineStage wrapper to define pipeline stages for the EDT input and output channels.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(CoreB,rtl) {
    EDT {
      Controller(1) {
        scan_chain_count: 100;
        longest_chain_range: 100,110;
        input_channel_count: 4;
        output_channel_count: 4;
        leaf_instance_name: edt_ctrl_inst1;
        Connections {
          EdtChannelsIn(4:1) {
            port_pin_name: new_input_bus[3:0];
            PipelineStage {
              parent_instance : block1;
              leaf_instance_name : pipe1 ;
              pipeline_clock : edt_clock;
            }
            PipelineStage {
              parent_instance : block1;
              leaf_instance_name : pipe2 ;
              pipeline_clock : edt_clock;
            }
          }
          EdtChannelsOut(4:1) {
            port_pin_name: new_output_bus[3:0];
            PipelineStage {
              parent_instance : block1;
              leaf_instance_name : pipe3 ;
              pipeline_clock : edt_clock;
            }
            PipelineStage {
              parent_instance : block1;
              leaf_instance_name : pipe4 ;
              pipeline_clock : edt_clock;
            }
          }
        }
      }
      Controller(2) {
        scan_chain_count: 100;
        longest_chain_range: 100,110;
        input_channel_count: 4;
        output_channel_count: 4;
        leaf_instance_name: edt_ctrl_inst2;
        Connections {
           EdtChannelsIn(4:1) {
              port_pin_name: new_input_bus[15:12];
              PipelineStage {
                parent_instance : block1;
                leaf_instance_name : pipe5 ;
                pipeline_clock : edt_clock;
              }
           }
           EdtChannelsOut(4:1) {
              port_pin_name: new_output_bus[15:12];
           }
        }
      }
    } }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

EdtChannelsOut

This wrapper is used to add and specify properties for the EDT channel out ports.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      Connections {
        EdtChannelsOut(range) {
          port_pin_name  : port_pin_name ;
          pipeline_clock : port_pin_name ; // default: edt_clock
          insert_lockup_cell : auto | off;
          lockup_cell_type   : latch | dff;
          PipelineStage {
          }
        }
      }
    }
  }
}
```

Description

The port_pin_name and the pipeline_clock properties are not used when you are using SSN with a ScanHost wrapper present. You will get an error in that case if you try to specify those properties. The destination of the EDT channel output will be the from_scan_out ports of the ScanHost node. You need to specify the ssh_chain_group property if the ScanHost node was build with more than one chain group as in "Example 1" on page 4523 of the ScanHost section.

The process_dft_specification command for EDT creates bus ports for EDT channel connections on the block level as specified in the configuration data. If the bus ports exist already, they are used for EDT channel connections. If they do not exist, they are created on the block level by process_dft_specification.

You specify the bus names with the port_pin_name properties with this configuration data wrapper. The ports must have sufficient bits to make all of the specified EDT channel output connections. EDT process_dft_specification cannot enhance existing ports with additional bits.

You can specify the bus ports to be created with any bit range as long as it is a valid Verilog range. They can start with 0 or any other number. Any range is acceptable as long as the EDT channel range width matches the specified connection width. The order you specify the bus ports has an impact on the connection order made to the EDT IP.

The *(range)* component of the EdtChannelsOut wrapper accepts a range of positive integer values using a single value or a colon-separated pair of values. For example:

```
EdtChannelsOut(8:7) — two EDT output channels
```

When you specify at least one output connection, you must specify all values and ranges.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The EDT process_dft_specification does not expand existing bus ports. An error is reported if you use more bits than are on the existing bus.

Arguments

- port_pin_name : *port_pin_name* ;

  A required property that specifies the signals connected to the edt_channels_out port of the EDT controller. If you are using an EDT controller for the internal scan modes and a smaller one for the external scan modes, you can specify the same destinations for each controller as long as you have specified the mode_enables property in both Controller wrappers.

- pipeline_clock : *port_pin_name* ;

  An optional property that specifies the pipeline clock. If not specified, the default is edt_clock. See "Pipeline Stage Insertion" and "Lockup Cells on the Output Side of the EDT Controller" in the *Tessent TestKompress User's Manual* for more information.

- insert_lockup_cell : auto | off ;

  An optional property that specifies whether the tool inserts lockup cells. By default, the tool inserts lockup cells as needed when the source and destination clocks overlap as follows:

  - Both clocks have identical waveform timing within a tester cycle; clocks are on at the same time and their edges are aligned.
  - The active edge of the destination clock occurs later in the cycle than the active edge of the source clock.

  When clocks do not overlap, the timing sequence protects the data, so no lockup cells are needed.

  When this property is "off", the tool does not insert lockup cells.

  Caution

  You should only turn off lockup cells in the following circumstances:

  - Between two pipeline stages that are driven by two different clocks when those clocks are known to be synchronous during shift.
  - After the last channel input pipeline, when the clock driving the last pipeline is synchronous with the EDT clock driving the decompressor *and* is also synchronous with the clock driving the first scan cells in bypass mode (if present in EDT), and the first cells are driven by the LE of their clocks.

  Note

  When this property is set to auto, the tool automatically omits the lockup cell after the last channel output pipeline stage when the design level is "chip". The lockup cell is unnecessary in this case, and omitting it improves timing.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- lockup_cell_type : latch | dff ;

  An optional property that specifies whether to use latch or DFF lockup cells. By default, the tool uses latches for lockup cells.

  When you specify DFF lockup cells, the tool first attempts to use negedge DFFs. If these are not available, it then attempts to use posedge DFFs and clock inverters. If no clock inverter is available, the tool uses regular inverters and reports a warning that negedge DFFs and clock inverters are not available.

Examples

Example 1

In this example, the tool connects the first four EDT output channels to the out* scalar ports.

```
EdtChannelsOut(1:4) {
  port_pin_name : out4, out3, out2, out1;
}
```

The EDT channel index order is reversed. Consequently, the connections are made as follows:

```
edt_ctrl_inst/edt_channels_out[3] -> out1
edt_ctrl_inst/edt_channels_out[2] -> out2
edt_ctrl_inst/edt_channels_out[1] -> out3
edt_ctrl_inst/edt_channels_out[0] -> out4
```

Example 2

This example shows how to create two EDT Controllers with new bus ports.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(CoreB,rtl) {
    EDT {
      Controller(1) {
        scan_chain_count: 100;
        longest_chain_range: 100,110;
        input_channel_count: 4;
        output_channel_count: 4;
        leaf_instance_name: edt_ctrl_inst1;
        Connections {
           EdtChannelsIn(4:1) {
              port_pin_name: new_input_bus[3:0];
           }
           EdtChannelsOut(4:1) {
              port_pin_name: new_output_bus[3:0];
           }
        }
      }
      Controller(2) {
        scan_chain_count: 100;
        longest_chain_range: 100,110;
        input_channel_count: 4;
        output_channel_count: 4;
        leaf_instance_name: edt_ctrl_inst2;
        Connections {
           EdtChannelsIn(4:1) {
              port_pin_name: new_input_bus[15:12];
           }
           EdtChannelsOut(4:1) {
              port_pin_name: new_output_bus[15:12];
           }
        }
      }
    }
  }
```

With this configuration data in place, two new EDT controllers are created when you run the process_dft_specification command, edt_ctrl_inst1 and edt_ctrl_inst2. The output EDT channels in each controller are connected to the created bus ports named “new_output_bus” by the EdtChannelsOut wrapper. This shows the connections made for both controllers:

```
edt_ctrl_inst1/edt_channels_out[3] -> new_output_bus[15]
edt_ctrl_inst1/edt_channels_out[2] -> new_output_bus[14]
edt_ctrl_inst1/edt_channels_out[2] -> new_output_bus[13]
edt_ctrl_inst1/edt_channels_out[0] -> new_output_bus[12]
edt_ctrl_inst2/edt_channels_out[3] -> new_output_bus[15]
edt_ctrl_inst2/edt_channels_out[2] -> new_output_bus[14]
edt_ctrl_inst2/edt_channels_out[1] -> new_output_bus[13]
edt_ctrl_inst2/edt_channels_out[0] -> new_output_bus[12]
```

The new bus ports have a range of 16 bits (15:0) each because the MSB specified is 15 and the LSB is 0 for both of them. The tool always creates every bus port from its MSB to LSB bits as specified in the config data.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

PipelineStage

This wrapper is used to add and specify properties for the EDT channel out pipeline stage.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      Connections {
        EdtChannelsOut(id) {
          PipelineStage {
            parent_instance    : instance_name ;
            leaf_instance_name : instance_name ;
            pipeline_clock     : port_pin_name ;
                               default: Inherited from EdtChannelsOut value
            insert_lockup_cell : auto | off;
                               default: Inherited from EdtChannelsOut value
            lockup_cell_type   : latch | dff;
                               default: Inherited from EdtChannelsOut value
           }
        }
      }
    }
  }
}
```

Description

This wrapper is used to add and specify properties for the EDT channel out pipeline stage. See "Pipeline Stage Insertion" and "Lockup Cells on the Output Side of the EDT Controller" in the *Tessent TestKompress User's Manual* for more information.

Note

For SSN, additional muxing logic is added to EDT pipeline stages to avoid capture mismatches. Affected pipeline stages are also clocked by the shift_capture_clock output of the corresponding SSH instance instead of the edt_clock. These hardware modifications are only included when using the following configuration:

- The EDT controller is connected to an SSH.
- The EDT input or output channels require one or more pipeline stages.
- The EDT controller has a bypass mode included.

Caution

If the EDT compactor change edge has been changed from its default using the change_edge_at_compactor_output property, and an output pipeline stage has been inserted and is not clocked by the LE edt_clock, there is a risk of violating the hold time requirement on the pipeline stage. The clock to the output pipeline stage must be balanced to the EDT clock to avoid violating the hold time requirement of the pipeline stage.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- parent_instance : *instance_name* ;

  An optional property that specifies the parent instance. This is the name of the instance where the pipeline IP is generated. If it is not specified, the IP is placed in the current design scope.

- leaf_instance_name : *instance_name* ;

  An optional property that specifies the instance name of the pipeline IP. If not specified, the leaf instance name is auto generated with the following format:

  ```
  <top_module_name>_<dft_spec_id>_tessent_<instrument_name>_
  <edt_ctrl_id>_<channel_type>_<channel_bit>_pipe_inst_<pipe_index>
  ```

- pipeline_clock : *port_pin_name* ;

  An optional property that specifies the pipeline clock port pin name. This property overrides the value of the pipeline_clock specified in the EdtChannelsOut wrapper. If not specified, it inherits the value from the EdtChannelsOut wrapper. See "Pipeline Stage Insertion" and "Lockup Cells on the Output Side of the EDT Controller" in the *Tessent TestKompress User's Manual* for more information.

- insert_lockup_cell : auto | off ;

  An optional property that specifies whether the tool inserts lockup cells. By default, the tool inserts lockup cells as needed when the source and destination clocks overlap as follows:

  - Both clocks have identical waveform timing within a tester cycle; clocks are on at the same time and their edges are aligned.
  - The active edge of the destination clock occurs later in the cycle than the active edge of the source clock.

  When clocks do not overlap, the timing sequence protects the data, so no lockup cells are needed.

  When this property is "off", the tool does not insert lockup cells.

  Caution

  You should only turn off lockup cells in the following circumstances:

  - Between two pipeline stages that are driven by two different clocks when those clocks are known to be synchronous during shift.
  - After the last channel input pipeline, when the clock driving the last pipeline is synchronous with the EDT clock driving the decompressor *and* is also synchronous with the clock driving the first scan cells in bypass mode (if present in EDT), and the first cells are driven by the LE of their clocks.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

When this property is set to auto, the tool automatically omits the lockup cell after the last channel output pipeline stage when the design level is “chip”. The lockup cell is unnecessary in this case, and omitting it improves timing.

- lockup_cell_type : latch | dff ;

  An optional property that specifies whether to use latch or DFF lockup cells. By default, the tool uses latches for lockup cells.

  When you specify DFF lockup cells, the tool first attempts to use negedge DFFs. If these are not available, it then attempts to use posedge DFFs and clock inverters. If no clock inverter is available, the tool uses regular inverters and reports a warning that negedge DFFs and clock inverters are not available.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example uses the PipelineStage wrapper to define pipeline stages for the EDT input and output channels:

```
DftSpecification(CoreB,rtl) {
    EDT {
      Controller(1) {
        scan_chain_count: 100;
        longest_chain_range: 100,110;
        input_channel_count: 4;
        output_channel_count: 4;
        leaf_instance_name: edt_ctrl_inst1;
        Connections {
          EdtChannelsIn(4:1) {
            port_pin_name: new_input_bus[3:0];
            PipelineStage {
              parent_instance : block1;
              leaf_instance_name : pipe1 ;
              pipeline_clock : edt_clock;
            }
            PipelineStage {
              parent_instance : block1;
              leaf_instance_name : pipe2 ;
              pipeline_clock : edt_clock;
            }
          }
          EdtChannelsOut(4:1) {
            port_pin_name: new_output_bus[3:0];
            PipelineStage {
              parent_instance : block1;
              leaf_instance_name : pipe3 ;
              pipeline_clock : edt_clock;
            }
            PipelineStage {
              parent_instance : block1;
              leaf_instance_name : pipe4 ;
              pipeline_clock : edt_clock;
            }
          }
        }
      }
      Controller(2) {
        scan_chain_count: 100;
        longest_chain_range: 100,110;
        input_channel_count: 4;
        output_channel_count: 4;
        leaf_instance_name: edt_ctrl_inst2;
        Connections {
           EdtChannelsIn(4:1) {
              port_pin_name: new_input_bus[15:12];
              PipelineStage {
                parent_instance : block1;
                leaf_instance_name : pipe5 ;
                pipeline_clock : edt_clock;
              }
           }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
                EdtChannelsOut(4:1) {
                   port_pin_name: new_output_bus[15:12];
             }  }
          }
       }
    }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Decompressor

This wrapper adds and specifies properties for the EDT decompressor options.

Usage

```
DftSpecification(module_name,id) {
  EDT {
    Controller(id) {
      Decompressor {
        segments                : int ;  // default: auto
        max_chains_per_segment  : int ;  // default: 150
      }
    }
  }
}
```

Arguments

- segments : *int* ;

  An optional property that specifies the number of segments to split the decompressor into. If you provide a value for this property, the tool determines the number of chains per segment based on the total number of chains and the specified number of segments. Lower numbers of chains per segment can help reduce routing congestion. The default value, auto, indicates that the tool uses the max_chains_per_segment value to determine the number of segments. If you specify a value for both the max_chains_per_segment property and this property, the tool issues a warning and uses this value. This property is equivalent to the "-SEGmented_decompressor" option in the set_edt_options command.

- max_chains_per_segment : *int* ;

  An optional property that specifies the maximum number of chains per segment when the tool segments the decompressor. By default, the tool segments the decompressor into equal parts with a maximum of 150 chains per segment. Lower numbers of chains per segment can help reduce routing congestion. If you specify a value for both the segments property and this property, the tool issues a warning and ignores this value. This property is equivalent to the "-MAX_Chains_per_decompressor_segment" option in the set_edt_options command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following specification produces two decompressor segments with 470 scan chains each:

```
DftSpecification(design, rtl) {
  Edt {
    ijtag_host_interface: Sib(edt);
    Controller(c1) {
      longest_chain_range: 2, 50;
      scan_chain_count: 940;
      input_channel_count: 4;
      output_channel_count: 4;
      Decompressor {
        segments: 2;
      }
    }
  }
}
```

If you did not specify the segments property, this would instead use the default maximum value of 150 scan chains per segment and produce seven segments.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

IjtagNetwork

A wrapper that specifies the IJTAG network to be built and, optionally, inserted into the design.

Usage

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    ImplementationOptions {
    }
    DataInPorts {
    }
    DataOutPorts {
    }
    HostScanInterface (id) {            // repeatable
      Interface {                       // *DefSpec
      }
      DesignInstance(instance_name) {   // repeatable
      }
      SecondaryHostScanInterface(id){   // repeatable
      }
      ScanMux(id) {                     // repeatable
      }
      Sib(id) {                         // repeatable
      }
      Tdr(id) {                         // repeatable
      }
      Tap(id) {                         // repeatable
      }
      TwoPinSerialPort(id) {            // repeatable
      }
      STap(id) {                        // repeatable
      }
      STapPipeline(id) {                // repeatable
      }
    }
  }
}
```

Description

This wrapper can be specified once for each DftSpecification wrapper. As shown in the syntax above, the IjtagNetwork specification consists of one IjtagNetwork/DataInPorts wrapper, one IjtagNetwork/DataOutPorts wrapper, one ImplementationOptions wrapper, and one or more HostScanInterface wrappers.

A HostScanInterface wrapper defines a scan interface below which a network of IJTAG nodes are serially accessible. The HostScanInterface/Interface wrapper specifies the end points of the HostScanInterface. By default, the wrapper defines a set of ports that may or may not already exist on the current design, but you can also use this wrapper to specify how to connect to a host interface already present in the design. For more details and example usage, see the description of the HostScanInterface/Interface wrapper. If the HostScanInterface/Interface wrapper only consists of ports on the current design, run the add_icl_scan_interfaces and set_icl_scan_interface_ports commands during insertion. Then, if you perform ICL extraction

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

once you are done with insertion, the tool names the ScanInterface in the extracted ICL module using the id of the HostScanInterface you specified in the DftSpecification.

The DesignInstance, ScanMux, Sib, Tap, Tdr, and TwoPinSerialPort wrappers define nodes serially connected below the HostScanInterface. You can specify them in any order, and they are all repeatable. However, their relative order defines the scan path connectivity. The scan_out of the first element declared in the HostScanInterface wrapper sources the scan_out port of the HostScanInterface. The scan_in port of the last element declared in the HostScanInterface wrapper is sourced by the scan_in port of the HostScanInterface.

The SIB, ScanMux, TAP and TwoPinSerialPort nodes become host interfaces to the client interfaces of the nodes declared inside them. See the description of the Sib, ScanMux, Tap, and TwoPinSerialPort wrappers for more details. The example below shows the hierarchical nature of the IJTAG network.

- If you want to insert BoundaryScan and the TAP controller, and connect the TAP to internal pins, follow the instructions in Example 2.
- If you want to insert an IJTAG network that is compatible to a tiling physical flow where no login nor complex routing is allowed at the top level, follow the instructions in Example 3.

Scan Testability of the IJTAG Network

Elements of the IJTAG network can be used to configure the circuit in its various scan modes. The example shown in Figure 13-4 illustrates the use of a TDR to generate the various scan mode signals. In this example, the portion of the network that gives access to this TDR must remain active during scan test and must remain unaffected by the scan test modes like any other test_setup hardware you may use. If the property keep_active_during_scan_test is set on a node, the host access path to the node is traced and every node in the access path is marked with the keep_active_during_scan_test attribute. The ICL modules are generated with this attribute and collected into the merged ICL file created by ICL extract. You can later use this information to declare those modules as non-scan during scan insertion, and force them to be kept in the graybox during graybox generation.

When the id of a Tdr wrapper is named “scan_modes”, the keep_active_during_scan_test property defaults to on for that TDR. This, in turn, forces all nodes needed to access it from the HostScanInterface to have keep_active_during_scan_test set to on.

Once you have extracted the final ICL using ICL extract, executing the command “get_icl_instances -filter keep_active_during_scan_test” generates a collection of all ICL instances having keep_active_during_scan_test set to on. You can then use the tessent_design_instance attribute on the ICL instances to find the corresponding design instances and pass this list to your scan insertion and/or graybox generation step. For a complete list of attributes available on the ICL instance object type, see the icl_instance data model.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

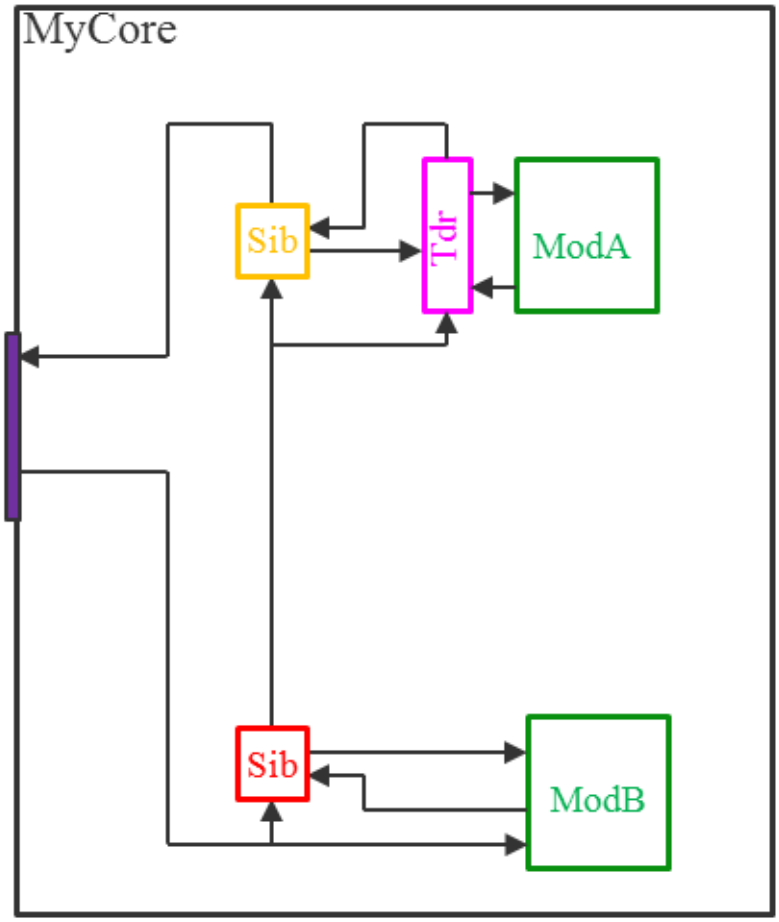
None.

Examples

Example 1

The following example shows the DftSpecification of a simple IJTAG network having two SIBs connected in series. The first SIB hosts a design instance, and the second SIB hosts a TDR, which in turn sources and observes the data signals of a second design instance. The example assumes the design already includes those two design instances and that an ICL file describing their ports exists for each of them. Figure 13-4 shows the schematic for this example.

Figure 13-4. IJTAG Network Example Schematic



The instance “u1/bottom_green” is an instance of module ModB and has a single ScanInterface described in its ICL file; its ScanInterface is connected to the host interface of the red SIB.

The instance “u2/top_green” is an instance of module ModA and has four DataInPorts and two DataOutPorts described in its ICL file. The DataOutPorts of the pink TDR is connected to the DataInPorts of ModA and vice versa. See the description of the DesignInstance wrapper for

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

information about how the ICL files are automatically searched and matched to the design instance during the set_current_design invocation.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Sib(orange) {
        Tdr(pink) {
          DataOutPorts {
            connection(0)   : u2/top_green/en ;
            connection(3:1) : u2/top_green/mode[2:0] ;
          }
          DataInPorts {
            connection(0) : u2/top_green/go ;
            connection(1) : u2/top_green/done ;
          }
        }
      }
      Sib(red) {
        DesignInstance (u1/bottom_green) {}
      }
    }
  }
}
```

Example 2

The following example inserts the TAP and the Boundary Scan chain in one pass using a DftSpecification wrapper. This example shows the steps to follow when you want to connect the TAP controller to internal pins that are not directly on the pads associated with the TAP ports. Typically, you want to do this if you are using a compliance enable pin, and the logic to enable the TAP is already present in the design. If the gating logic is not already present, use the -active_high_compliance_enables or -active_low_compliance_enables switches of the create_dft_specification command.

The following example sets the function attribute on the top-level ports to identify the top-level TAP port in Step 1. This example connects the TAP to internal pins, which are enabled when the top-level port “myCE1” is asserted high. Step 1b shows how your compliance enable port is described, so the tool reports it as such in the BSDL file. This example assumes you already have this logic in the design, and that you want to connect the TAP to it. The recommended flow is to not insert this logic manually, but let the tool auto-create and insert it by using the -active_high_compliance_enables or -active_low_compliance_enables switches of the create_dft_specification command. When you do that, the flow is fully automated. When you create your own compliance enable logic (as shown in Figure 13-5 on page 3936), you need to either describe that module in ICL or assert the compliance enable ports before running ICL extraction. The end of this example provides an explanation of when to use which method.

In Step 2, you call “set spec [create_dft_specification]” to create the DftSpecification that includes a HostScanInterface wrapper with a TAP wrapper inside it. Step 3 augments the IjtagNetwork wrapper with a second HostScanInterface, where you use the Interface properties to specify the connections to internal pins. Step 4 shows the use of the move_config_element

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

command to move the TAP wrapper that exists in the original HostScanInterface to the new one. Finally, Step 5 reports the final DftSpecification. The tool uses the HostScanInterface(tap) to identify the top-level TAP ports present in the BSDL file, and the second HostScanInterface(tap_internal) specifies where to connect the client scan interface of the TAP controller.

```
# step 1: set the 'function' attribute on the top level TAP ports
set_attribute_value -name function -value trst jtag_trst
set_attribute_value -name function -value tdi  jtag_tdi
set_attribute_value -name function -value tms  jtag_tms
set_attribute_value -name function -value tck  jtag_tck
set_attribute_value -name function -value tdo  jtag_tdo

# Step 1b: declare your compliance enable port so it is documented in
#          the BSDL file
set_boundary_scan_port_options myCE1 -cell_options compliance_enable1

# Step 2: create an initial DFT specification
set_system_mode analysis
set spec [create_dft_specification]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# Step 3: add new HostScanInterface with the internal hookup points
read_config_data -in_wrapper ${spec}/IjtagNetwork -from_string {
   HostScanInterface(tap_internal) {
      Interface {
        tck :             jtag_and0/o ;
        trst :            jtag_and3/o ;
        tms :             jtag_and1/o ;
        tdi :             jtag_and2/o ;
        tdo :             jtag_buf5/a ;
        tdo_en :          jtag_buf4/a ;
        tdo_en_polarity : active_high ;
      }
   }
}
# step 4: move the Tap to the new HostScanInterface
move_config_element ${spec}/IjtagNetwork/HostScanInterface(tap)/Tap(main) \
       -in_wrapper ${spec}/IjtagNetwork/HostScanInterface(tap_internal)

# step 5: report the final DftSpecification.
report_config_data $spec
The resulting DFT specification looks like:
DftSpecification(top,rtl) {
  IjtagNetwork {
    HostScanInterface(tap) {
      Interface {
        tck : jtag_tck ;
        trst : jtag_trst ;
        tms : jtag_tms ;
        tdi : jtag_tdi ;
        tdo : jtag_tdo ;
      }
    }
    HostScanInterface(tap_internal) {
      Interface {
        tck : jtag_and0/o ;
        trst : jtag_and3/o ;
        tms : jtag_and1/o ;
        tdi : jtag_and2/o ;
        tdo : jtag_buf5/a ;
        tdo_en : jtag_buf4/a ;
        tdo_en_polarity : active_high ;
      }
      Tap(main) {
        HostIjtag(1) {
        }
        HostBscan {
        }
      }
    }
  }
  BoundaryScan {
    ijtag_host_interface : Tap(main)/HostBscan ;
    pin_order_file : ../data/top.pinorder ;
    BoundaryScanCellOptions {
      myCE1 : compliance_enable1 ;
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

This example explains how to connect the newly created TAP to internal pins, which typically happens when you have compliance enable logic already inserted into the design. This is allowed, but you must deal with this logic before running the extract_icl command.

However, the flow is fully automated when you let the tool insert the compliance enable logic using the -active_high_compliance_enables or -active_low_compliance_enables switches of the create_dft_specification command.

Figure 13-5 on page 3936 shows the typical logic used to implement compliance enabling when you want to switch between an existing TAP and the Tessent Shell TAP. You may also be using this logic to switch between arbitrary functional logic and the Tessent Shell TAP. If there is no existing TAP or if the existing TAP does not have an ICL view that you want to be represented in the extract ICL file, you need to use the add_input_constraints command to assert the compliance enable ports to their active value before running the extract_icl command. If you control the enable with internal logic, you must use a test_setup proc file to describe the vectors needed to activate it. If you have an ICL model for the existing TAP and you want it described

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

in the extracted ICL file, you must model the multiplexing module with an ICL file similar to this:

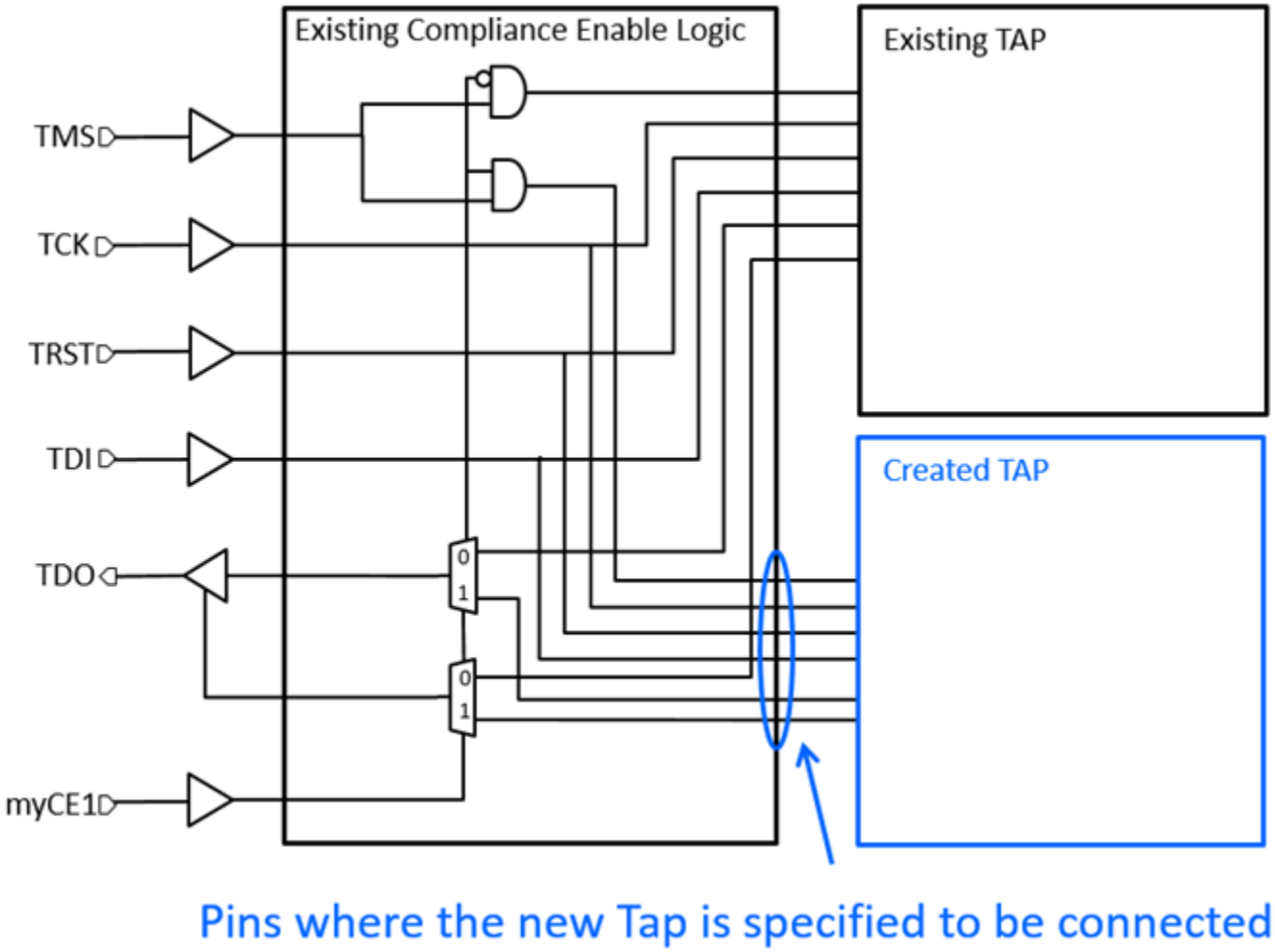
```
Module compliance_enable {
   // from the pads
   TCKPort     tck ;
   ScanInPort  tdi ;
   ScanOutPort tdo {
      Source ComplianceOutputSelect;
      Attribute forced_high_output_port_list = "tdo_en" ;
   }
   DataOutPort tdo_en {
      Attribute associated_scan_port_list = "tdo";
      Attribute connection_rule_option = "allowed_no_destination";
      Attribute function_modifier = "tdo_enable_active_high";
   }
   TMSPort     tms ;
   TRSTPort    trst {
      Attribute connection_rule_option = "allowed_tied_high" ;
   }
   DataInPort ce ;
   DataInPort tdo_en_other {
      Attribute associated_scan_port_list = "tdo_other";
      Attribute connection_rule_option = "allowed_no_source";
      Attribute function_modifier = "tdo_enable_active_low";
   }
   DataInPort tdo_en_ts {
      Attribute associated_scan_port_list = "tdo_ts";
      Attribute connection_rule_option = "allowed_no_source";
      Attribute function_modifier = "tdo_enable_active_low";
   }
   ScanInterface tap_pins {
      Port tdi ;
      Port tdo ;
      Port tms ;
      Port tck ;
      Port trst ;
   }
   // to the existing TAP
   ToTCKPort   tck_other ;
   ToTMSPort   tms_other ;
   ToTRSTPort  trst_other ;
   ScanInPort  tdo_other ;
   ScanOutPort tdi_other {
      Source tdi ;
   }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
    ScanInterface tap_pins_other {
       Port tdi_other ;
       Port tdo_other ;
       Port tms_other ;
       Port tck_other ;
       Port trst_other ;
    }
    // to the Tessent Shell generated TAP
    ToTCKPort   tck_ts ;
    ToTMSPort   tms_ts ;
    ToTRSTPort  trst_ts ;
    ScanInPort  tdo_ts ;
    ScanOutPort tdi_ts {
       Source tdi ;
    }
    ScanInterface tap_pins_ts {
       Port tdi_ts ;
       Port tdo_ts ;
       Port tms_ts ;
       Port tck_ts ;
       Port trst_ts ;
    }
    // the mux for the tdo
    ScanMux ComplianceOutputSelect SelectedBy ce {
      1'b1 : tdo_ts ;
      1'b0 : tdo_other ;
    }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-5. Existing Compliance Enable Modules



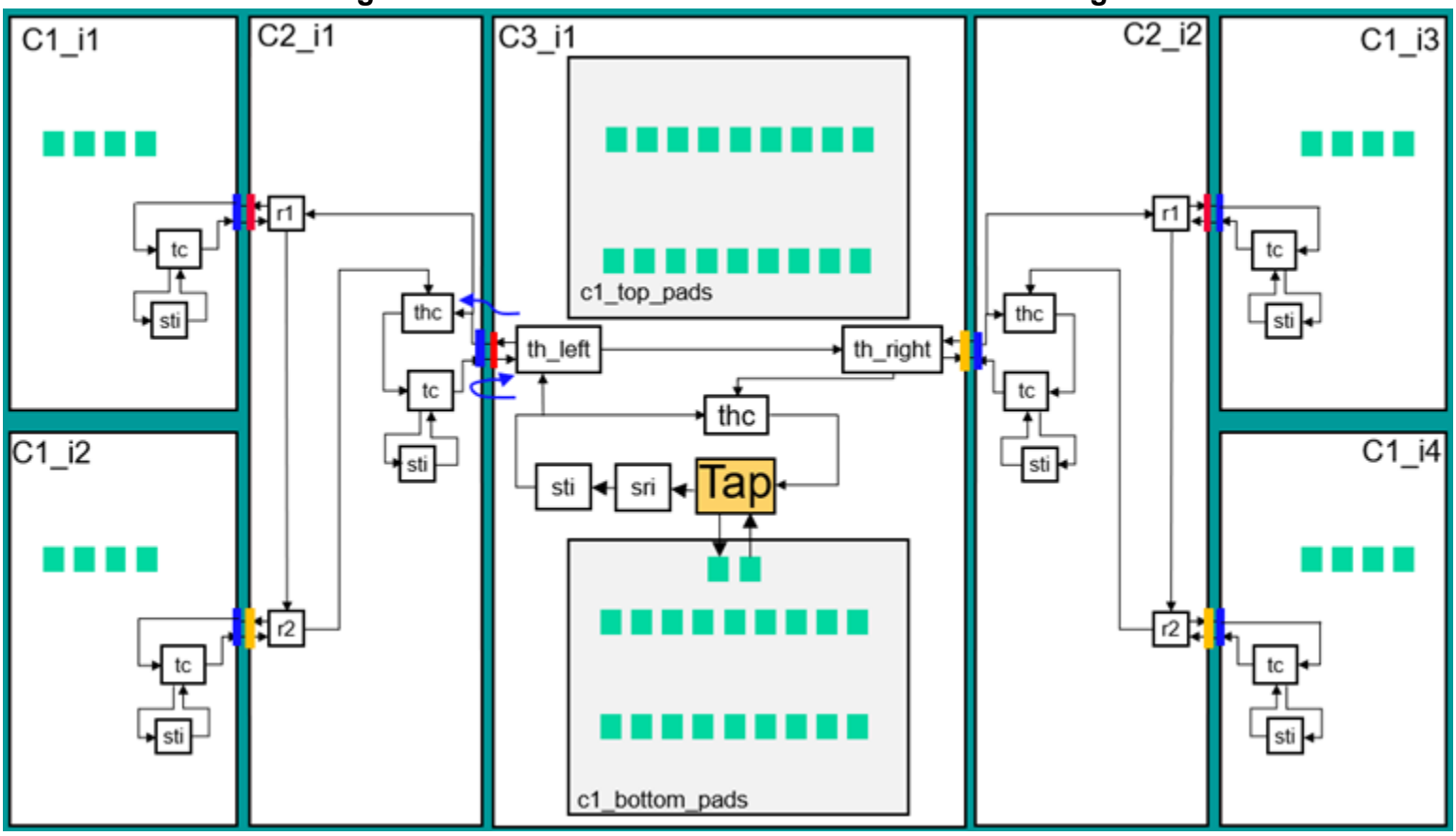
Example 3

This example shows how to build the IJTAG network when using a tiling physical flow, where physical regions abut to each other with no top-level logic or complex routing. This flow also uses a localized TCK clock tree in each physical region, and the TCK communication between the blocks use source-synchronous timing.

The example focuses on the IJTAG part of the reference design presented in Example 4 of the OCC wrapper. That example explains the functional clocking, with the localized functional clock trees and the forward and loop timing interfaces. Optimally, the IJTAG network clocking is similar to the functional mode, where the TCK clock trees are localized to each tile and the leaf node of one tree sources the TCK clock of the adjacent tiles. Figure 13-6 shows a schematic representation of the IJTAG network used in the tiled design used in Example 4 in the OCC wrapper reference section.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-6. IJTAG Network Inserted Into a Tiled Design



The example inserts the TAP controller into the C3 central block, which contains the TAP ports. The TCK clock trees are localized to each physical block. The outward direction uses a forward-timing interface, having a negedge-to-posedge path, and the return path uses a posedge-to-posedge loop timing interface. The path taken by the IJTAG network in Figure 13-6 follows the symmetry of the tiles, and routes the IJTAG scan path in a way to avoid having several alternative input and output paths within each tile.

Tip

In this example, you may want to route the IJTAG scan chain from C1_I1 down into C1_I2. But this does not make sense, as both tiles are instances of the same module. If the scan_in entered the tile from the top of C1_I2, then it must also enter from the same place in the other instances. Just as you designed the functional logic paths, entering the C1 tiles from one side keeps the symmetry intended by the design. Below are sections of the three dofiles used to insert the IJTAG network into the C1, C2, and C3 tiles.

The first insertion pass in this design is to insert the boundary scan chain. You can choose to insert boundary scan, IJTAG, memory BIST, and Memory BISR in one insertion pass. In this example, boundary scan insertion is separate from the other three components to show that it is possible. The TAP controller and the Sib(sri) in C3 were previously inserted into the RTL as

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

part of the boundary scan phase, as shown in Example 1 of the EmbeddedBoundaryScan/HostBScanInterface wrapper reference section.

This example includes dofile excerpts to create the IjtagNetwork section of the DFT specification. Typically, you insert memory BIST and memory BISR in the same insertion pass but this example omits the switches specific to memory BISR from these excerpts for simplification. Refer to Example 3 of the MemoryBISR/SecondaryHostChainInterface wrapper for the switches and commands specific to memory BISR insertion within a tiling flow.

C1

Below is the dofile excerpt to insert IJTAG network into the C1 tile. The only difference from the normal flow is the use of the "**-design_type tile**" switch in the set_dft_specification_requirements command. The reported IjtagNetwork wrapper below shows how this instructs the create_dft_specification command to add a special SIB at the root of the network. This Sib is given the "tc" ID and serves as a host to the entire IJTAG network within the tile. The **to_scan_in_feedthrough** property is set to **pipeline**, so that the ijtag_si pin gets locally pipelined before going to the rest of the nodes. The **so_retiming** property is set to **off**, so that the loop timing interface from the TCK clock to the ijtag_so output has a full clock period to propagate. Adding a retiming element in a loop timing interface forces the loop timing interface to operate in half a clock period. The retiming element serves no useful function in a loop timing interface as the TCK within the tile is always delayed with respect to the TCK within the tile that is sourcing the TCK and receiving the scan out data back. The Sib(sti) inserted within the Sib(tc) is the normal Sib that is always added to host the scan testable side of the IJTAG network.

```
read_design c1 -design_id rtl1
set_current_design c1

set_dft_specification_requirements -memory_test on -design_type tile
check_design_rules

set spec [create_dft_specification]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
report_config_data ${spec}/IjtagNetwork
IjtagNetwork {
  HostScanInterface(ijtag) {
    Sib(tc) {
      Attributes {
        tessent_dft_function : tile_client_sib;
      }
      to_scan_in_feedthrough : pipeline;
      so_retiming : off;
      Sib(sti) {
        Attributes {
          tessent_dft_function : scan_tested_instrument_host;
        }
        Sib(mbist) {
        }
      }
    }
  }
}
```

C2

Below is the dofile excerpt to insert the IJTAG network into the C2 tile. In this case, the C2 tile not only needs to service the local IJTAG network, but it also needs to pass it along to the C1 tiles, as shown in Figure 13-6. This is achieved with the use of "**-tile_ijtag_host**" switch on the create_dft_specification command. This example requests two host ports, r1 and r2, to be created to service the row 1 and row 2 of the C1 tiles that connects to the C2 tile. A special sib with ID **thc** (tile host collector) is instantiated in series with the Sib(tc) that is inferred because of the "**-design_type tile**" switch on the set_dft_specification_requirements command. Within the Sib(thc), two special Sibs are inserted with the **to_scan_in_feedthrough** switch specified as **pipeline**. The SecondaryHostScanInterface wrapper is then inserted within those Sibs to make connections to the top-level ports. Those DFT ports are typically pre-created into each tile and pre-connected into the chip-level netlist, as explained in the process_top_module_connections command and TopModuleConnections wrapper descriptions. Therefore, you must set the **allow_port_creation_on_current_design** property to **off** to ensure that no ports are accidentally created that would not have been pre-connected in the chip-level netlist.

```
read_design c2 -design_id rtl1set_current_design c2
set_dft_specification_requirements -memory_test on -design_type tile

check_design_rules

set spec [create_dft_specification -tile_ijtag_host_list {r1 r2}]
set_config_value allow_port_creation_on_current_design -in $spec off
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
report_config_data ${spec}/IjtagNetwork
IjtagNetwork {
  HostScanInterface(ijtag) {
    Sib(tc) {
      Attributes {
        tessent_dft_function : tile_client_sib;
      }
      to_scan_in_feedthrough : pipeline;
      so_retiming : off;
      Sib(sti) {
        Attributes {
          tessent_dft_function : scan_tested_instrument_host;
        }
        Sib(mbist) {
        }
      }
    }
    Sib(thc) {
      Attributes {
        tessent_dft_function : tile_host_collector;
      }
      Sib(th_r2) {
        to_scan_in_feedthrough : pipeline;
        SecondaryHostScanInterface(r2) {
        }
      }
      Sib(th_r1) {
        to_scan_in_feedthrough : pipeline;
        SecondaryHostScanInterface(r1) {
        }
      }
    }
  }
}
```

C3

Below is the dofile excerpt to insert the IJTAG network within the C3 tile, as shown in Figure 13-6. The "**-tile_ijtag_host**" switch on the create_dft_specification command is used to create the two IJTAG host ports on either side of the C3 tile. The TAP controller was inserted in the previous insertion pass when the boundary scan chains were connected to the TAP controllers, as shown in Example 1 of the EmbeddedBoundaryScan/HostBScanInterface section. The Sib(sri) was inserted along with the TAP during that phase because the boundary scan segments are used as dedicated wrapper cells during logic test. The create_dft_specification command detects the presence of the Sib(sri) node and automatically creates an IjtagNetwork specification that places the two new SIBs in series with it. The **allow_port_creation_on_current_design** property is set to **off** to make sure no new DFT ports are created on the C3 tile during the bottom-up insertion step. Those were pre-created into each tile and pre-connected into the chip-level netlist, as explained in the process_top_module_connections command and TopModuleConnections wrapper descriptions.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
read_design c3 -design_id rtl1
read_design c3_top_pads -design_id rtl1 -view ijtag_graybox
read_design c3_bottom_pads -design_id rtl1 -view ijtag_graybox
set_current_design c3

set_dft_specification_requirements -memory_test on \
                                   -design_type tile
check_design_rules

set spec [create_dft_specification -tile_ijtag_host_list {left right}]
set_config_value allow_port_creation_on_current_design -in $spec off

report_config_data ${spec}/IjtagNetwork
IjtagNetwork {
  HostScanInterface(sti) {
    Interface {
      design_instance : c3_rtl1_tessent_sib_sri_inst;
      scan_interface : client;
    }
    Sib(thc) {
      Attributes {
        tessent_dft_function : tile_host_collector;
      }
      Sib(th_right) {
        to_scan_in_feedthrough : pipeline;
        SecondaryHostScanInterface(right) {
        }
      }
      Sib(th_left) {
        to_scan_in_feedthrough : pipeline;
        SecondaryHostScanInterface(left) {
        }
      }
    }
    Sib(sti) {
      Attributes {
        tessent_dft_function : scan_tested_instrument_host;
      }
      Sib(mbisr) {
      }
    }
  }
}
```

Related Topics

DesignInstance

Sib

ScanMux

Tdr

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

IjtagNetwork/ImplementationOptions

A wrapper that specifies implementation options when building the IJTAG network.

Usage

```
IjtagNetwork {
  ImplementationOptions {
      scan_path_retiming : latch | flop ;
  }
}
```

Description

A wrapper that specifies implementation options when building the IJTAG network. The options consist of the use of latches or flip-flops for scan path retiming.

Arguments

- scan_path_retiming : latch | flop ;

  A property that specifies if the scan path retiming element is to be implemented using a latch or a flop.

  This property has significant impact on the TAP controller. When the scan_path_retiming property is set or left to default to “latch”, a retiming latch is used on the tdo output of the TAP controller to make sure that the scan_out data of any scan register is launched from the negative edge of TCK. It is likely that the IJTAG elements feeding the scan_in port of its HostIjtag scan interfaces already have a retiming element on their scan_out port. The retiming latch inserted in series before reaching TDO has no effect and simply acts as a redundant retiming latch. If you set the property to “flop”, it is no longer possible to systematically place the retiming element on all TDO paths because it would end up in series with the one that is likely already present in IJTAG elements feeding the scan_in port of its HostIjtag scan interfaces. Cascading two retiming latches does not affect the resulting timing but cascading two retiming flops does change the timing. When using flops for scan_path_retiming, you must ensure that all scan_out ports feeding the scan_in port of the HostIjtag scan interfaces already have a retiming element. This is always true when using elements created with the process_dft_specification command. Only when a third party IJTAG node is directly connected to the scan_in port of a HostIjtag scan interface of the TAP, can you fail this requirement.

  Note

  Make sure third party nodes include a retiming element on their scan_out when you connect them directly to the scan_in port of the HostIjtag scan interface’s Tessent Tap, and you have specified the scan_path_retiming property to “flop”, otherwise your elements are not 1149.1 compliant and are not usable within a board environment.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

IjtagNetwork/DataInPorts

A wrapper that specifies data-in ports that may or may not exist as input ports on the current design.

Usage

```
IjtagNetwork {
  DataInPorts {
    count              : integer ;           // default: auto
    port_naming        : port_naming, ... ; // default: ijtag_data_in[%d]
    multiplexing       : on | off | auto ;
    connection(range) : pin_name ;          // repeatable
  }
}
```

Description

A wrapper that specifies data-in ports that may or may not exist as input ports on the current design.

These data-in ports can be used to source data-in ports on child design instances, or to source the select port of a ScanMux as shown in the example for this wrapper.

Arguments

- count : *integer* ;

  A property to request a specific number of data-in ports. If the property is set, or defaults to auto, the count is determined to be large enough to satisfy the *range* value of all connection properties. If no connection property is specified, the count is determined to be large enough to satisfy the port_naming values if they are in a non-indexed format (%d).

- port_naming : *port_naming, ... ;*

  A property that specifies the naming of the data-in ports. The port_naming value has two distinct formats: a single string using an index symbol “%d”, or a list of port names with explicit sizes.

  When using the indexed format, you specify base_name[%d] when you want to use a bus and you specify base_name%<integer>d when you want to create a set of scalar ports. In both cases, the range is <count>-1 to 0. Using <integer> in the scalar naming ensures the integer is padded with 0s to be at least <integer> bits wide. For example, Din%2d creates Din00, Din01, Din02, and so on. Din%d and Din%1d mean the same thing. The default is “ijtag_data_in[%d]”.

  When using a list of port names, you control the exact naming and width of each data-in port. For example, if you specify “locked”, you get exactly one data-in port called “locked” as illustrated in “Example 2” on page 4004 of the Tdr/DataInPorts section.

- multiplexing : on | off | auto ;

  A property that specifies if a multiplexer is to be added when making the connection to the specified *pin_name* of the connection properties. When set to auto, a multiplexer is added

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

unless the pin is unconnected, connected to a constant or connected to a net with no fanin. As soon as one connection needs multiplexing, an extra data-in port is added to control the select of the multiplexers. The multiplexers are inserted in the direct parent instance of the specified *pin_name* of the connection properties. See "DataOutPort Connection Multiplexing" on page 4008 in the Tdr/DataOutPorts section for an illustration of this.

- connection(*range*) : *pin_name* ;

  A repeatable property that specifies the connections to make between the bits of the data-in ports and the existing pins inside the design. Because data-in ports are sources, the same bit can be connected to multiple destinations by simply repeating the connection property with the same range indexes.

  If the pin name refers to a bused pin and you are connecting to a full bus, you can omit the bus range if the process_dft_specification -no_insertion option is not used. If you are using the process_dft_specification -no_insertion switch, you must specify the full range of the destination pin because the range cannot be extracted from the design.

Examples

The following example illustrates defining five data-in ports. The ports are named my_data_in04, my_data_in03, my_data_in02, my_data_in01, and my_data_in00. The port my_data_in00 is connected to the "en" pin of the u2/top_green instance. The ports my_data_in03, my_data_in02, and my_data_in01 are connected to the mode[2:0] pins of the same instance. If the count property was not specified, it would have defaulted to four, corresponding to the largest specified connection range index value of three. In this example, a count of five is specified to request an extra data-in port to be used by the select port of the ScanMux(red) used in the HostScanInterface(purple).

Notice that if this core was further integrated into a parent design and a Tdr was used to source the five data-in ports, the property output_timing(4) with a value of "ijtag_scan_selection" would need to be specified because bit 4 is used to reconfigure the scan network. For further explanation, see "Update Stage Timing" on page 3995.

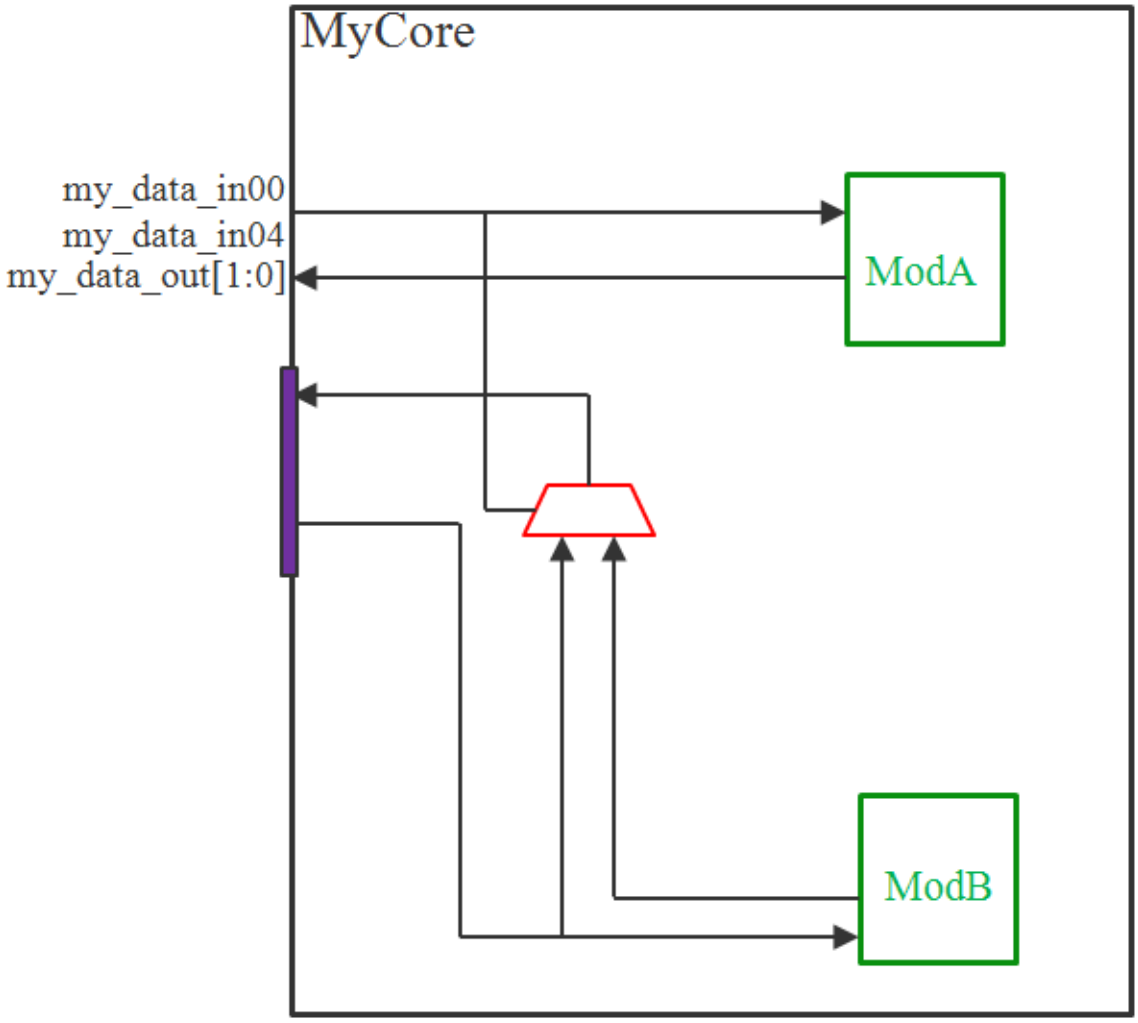
The tool is currently unable to determine if the data-in port on a design instance is used to reconfigure the network within the specified design instance but this automation is planned for the future.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    DataInPorts {
      Count            : 5 ;
      port_naming      : my_data_in%2d ;
      connection(0)    : u2/top_green/en ;
      connection(3:1)  : u2/top_green/mode[2:0];
    }
    DataOutPorts {
      port_naming : my_data_out[%d] ;
      connection(0) : u2/top_green/go ;
      connection(1) : u2/top_green/done ;
    }
    HostScanInterface(purple) {
      ScanMux(red) {
        select : DataIn(4) ;
        Input(1) {
          DesignInstance (u1/bottom_green) {}
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-7. DataInPorts Example Schematic



Related Topics

IjtagNetwork/DataOutPorts

Tdr/DataOutPorts

Tdr/DataInPorts

Tdr/DecodedSignal

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

IjtagNetwork/DataOutPorts

A wrapper that specifies data-out ports that may or may not exist as output ports on the current design.

Usage

```
IjtagNetwork {
  DataOutPorts {
    count              : integer ;           // default: auto
    port_naming        : port_naming,... ;   // default: ijtag_data_out[%d]
    Connection(range)  : pin_name ;          // repeatable
  }
}
```

Description

A wrapper that specifies data-out ports that may or may not exist as output ports on the current design.

These data-out ports can be used to observe data-out ports on child design instances.

Arguments

- count : *integer* ;

  A property to request a specific number of data-out ports. If the property is set, or defaults to auto, the count is determined to be large enough to satisfy the *range* value of all connection properties. If no connection property is specified, the count is determined to be large enough to satisfy the *port_naming* values if they are in a non-indexed format (%d).

- port_naming : *port_naming*, ... ;

  A property that specifies the naming of the data-out ports. The *port_naming* value as two distinct formats. It can be a single string using an index symbol or a list of port names with explicit sizes.

  When using the indexed format, you specify base_name[%d] when you want to use a bus and you specify base_name%<integer>d when you want to create a set of scalar ports. In both cases, the range is <count>-1 to 0. Using <integer> in the scalar naming ensures the integer is padded with 0s to be at least <integer> bits wide. For example, Dout%2d creates Dout00, Dout01, Dout02, and so on. Dout%d and Dout%1d mean the same thing. The default is “ijtag_data_out[%d]”.

  When using a list of port names, you control the exact naming and width of all data-out ports. For example, if you specify “mux_sel,mult[2:0],div[2:0]”, you get exactly seven data-out ports where the most significant bit is called mux_sel and the least significant bit is called div[0] as illustrated in “DataOutPort Connection Multiplexing” on page 4008 of the Tdr/DataOutPorts section.

- connection(*range*) : *pin_name* ;

  A repeatable property that specifies the connections to make between the bits of the data-out port and the existing pins inside the design. Because data-out ports are sinks, the same bit

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

cannot be connected to multiple sources. It is, therefore, illegal to repeat the connection property with overlapping range indexes.

If the pin name refers to a bused pin and you are connecting to a full bus, you can omit the bus range if the process_dft_specification -no_insertion option is not used. If you are using the process_dft_specification -no_insertion switch, you must specify the full range of the destination pin because the range cannot be extracted from the design.

Examples

The following example illustrates the creation of two data-out ports. The count property is not specified so it defaults to two corresponding to the largest connection range index used. The port naming is specified to define a port named “my_data_out[1:0]”. Each bit is connected to a unique source.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    DataOutPorts {
      port_naming : my_data_out[%d] ;
      connection(0) : u2/top_green/go ;
      connection(1) : u2/top_green/done ;
    }
  }
}
```

Related Topics

IjtagNetwork/DataInPorts

Tdr/DataOutPorts

Tdr/DataInPorts

Tdr/DecodedSignal

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

HostScanInterface/Interface

A wrapper used to customize the interface of the HostScanInterface.

Usage

```
IjtagNetwork {
  HostScanInterface(id) {
    Interface {
      design_instance : instance_name ;
      scan_interface  : id ;
      tck             : port_pin_name ;  // default: ijtag_tck
      reset           : port_pin_name ;  // default: ijtag_reset *DefSpec
      select          : port_pin_name ;  // default: ijtag_sel *DefSpec
      shift_en        : port_pin_name ;  // default: ijtag_se *DefSpec
      capture_en      : port_pin_name ;  // default: ijtag_ce *DefSpec
      update_en       : port_pin_name ;  // default: ijtag_ue *DefSpec
      scan_in         : port_pin_name ;  // default: ijtag_si *DefSpec
      scan_out        : port_pin_name ;  // default: ijtag_so *DefSpec
      reset_polarity  : active_high | active_low | auto ;
      trst            : port_pin_name ;  // default: trst *DefSpec
      tms             : port_pin_name ;  // default: tms *DefSpec
      tdi             : port_pin_name ;  // default: tdi *DefSpec
      tdi_local       : port_pin_name ;  // default: ""
      tdo             : port_pin_name ;  // default: tdo *DefSpec
      tdo_en          : port_or_pin_name ; // default: tdo_en
      tdo_en_polarity : active_high | active_low | auto ;
      SacrificialPads {
        parent_instance    : instance_name;
        leaf_instance_name : instance_name;
        enable             : port_pin_name; // default: enable_probe
        tck                : port_pin_name; // default: tck_probe
        trst               : port_pin_name; // default: trst_probe
        tms                : port_pin_name; // default: tms_probe
        tdi                : port_pin_name; // default: tdi_probe
        tdo                : port_pin_name; // default: tdo_probe
        tdo_en             : port_pin_name; // default: tdo_en_probe
        tdo_en_polarity    : active_high |active_low | auto;
      }
      spio            : port_name     ; // default: spio     *DefSpec
      spclk           : port_pin_name ; // default: spclk    *DefSpec
      daisy_chain_with_existing_client : on | off | auto ;
      ComplianceEnable {
        active_high_enables : port_pin_name, ... ;
        active_low_enables  : port_pin_name, ... ;
        parent_instance     : instance_name ;
        leaf_instance_name  : leaf_instance_name ;
      }
    }
  }
}
```

Description

A wrapper used to customize the interface of the HostScanInterface.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

By default, the HostScanInterface connects directly to ports on the current design and follows the default naming shown in the syntax summary above. Those ports may or may not already exist in the current design. If they are not already present, they are created during insertion if the design level is set to "physical_block" or "sub_block". If the design level is set to “chip”, ports need to already exist and be connected to pad cells. For more information, see the set_design_level and read_cell_library commands.

You can also specify internal pins to connect the HostScanInterface to an existing host interface already present in the design. The figure shows how the Interface wrapper is used to create top-level ports, and to hook up to an existing host port. Example 2 shows how to hook up a SIB to a TAP inserted by ETAssemble in the LV Flow.

To insert a TAP controller on internal pins and perform boundary scan insertion, refer to Example 2 of the IjtagNetwork wrapper section which shows the required specification manipulations.

When connecting to an internal pin, you must specify the reset_polarity of the reset pin; otherwise, you get an ICL extract violation later on if the assumed value is not consistent with the actual reset polarity. If specified as auto, the tool counts the number of children with an active high reset polarity and an active low reset polarity and takes the polarity of the larger count as the reset_polarity.

Parameters

- design_instance : *instance_name* ;

  This property defines the HostScanInterface on a ScanInterface defined in an ICL module associated with an instance in the design. The specified instance_name must be the name of an instance that already exists in the design, and it must be the instantiation of a module having a corresponding ICL module.

  ICL modules are automatically searched and matched to the design modules when the set_current_design command is executed. The search paths for the ICL files are specified using the set_design_sources -format icl command. The module name matching rules are specified using the set_module_matching_options command.

  A design_instance cannot be specified when process_dft_specification is run using the -no_insertion option.

- scan_interface : *id* ;

  This property specifies which ScanInterface to define the HostScanInterface on when the design_instance property points to an instance associated with an ICL module having more than one ScanInterface. If the associated ICL module only has one ScanInterface, the scan_interface property is not needed.

  If the referenced ScanInterface is a client ScanInterface, the host ScanInterface is defined in such a way that the inserted IJTAG nodes end up behind and in series with the referenced client ScanInterface. If the referenced ScanInterface is a host ScanInterface, the inserted

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

IJTAG nodes connect to it. This varying behavior based on the fact that the ScanInterface is of type host or client is illustrated in Example 1.

- tck : *port_pin_name* ;

  This property defines the port or the pin name to source the IJTAG TCKPort function. The default is “ijtag_tck”. See create_dft_specification description to learn how it can be automatically set.

- reset : *port_pin_name* ;

  This property defines the port or the pin name to source the IJTAG ResetPort function. The default is “ijtag_reset”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- select : *port_pin_name* ;

  This property defines the port or the pin name that is to source the IJTAG SelectPort function. The default is “ijtag_sel”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- shift_en : *port_pin_name* ;

  This property defines the port or the pin name that is to source the IJTAG ShiftEnPort function. The default is “ijtag_se”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- capture_en : *port_pin_name* ;

  This property defines the port or the pin name that is to source the IJTAG CaptureEnPort function. The default is “ijtag_ce”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- update_en : *port_pin_name* ;

  This property defines the port or the pin name that is to source the IJTAG UpdateEnPort function. The default is “ijtag_ue”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- scan_in : *port_pin_name* ;

  This property defines the port or the pin name that is to source the IJTAG ScanInPort function. The default is “ijtag_si”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- scan_out : *port_pin_name* ;

  This property defines the port or the pin name to receive the IJTAG ScanOutPort function. The default is “ijtag_so”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- reset_polarity : active_high | active_low | <u>auto</u> ;

  This property defines the polarity of the ResetPort. When set to auto, the reset_polarity is determine by counting the number of children with an active high or an active low reset port and taking the polarity of the larger count. When the reset property points to an existing

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

reset pin, it is required to specify the reset polarity consistent with the source being used. Failing to specify the proper polarity results in a subsequent ICL extraction DRC violation. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- trst : *port_pin_name* ;

  This property defines the port or the pin name that is to source the IJTAG TRSTPort function. The default is “trst”. This property is only used when the HostScanInterface is for a TAP scan interface. See create_dft_specification description to learn how it can be automatically set.

  When performing boundary scan insertion, the specified port needs to be a top level port. A 5-pin TAP interface is created for this specification. An internal pin can also be specified, which needs to be a power-on reset pin. A 4-pin TAP interface is created for this specification.

  Note

  When performing boundary scan insertion, if you need to specify an internal pin for tdi, tms, tck, or tdo that connects to a top level port, two insertion passes are needed. The TAP needs to be inserted in the first pass and the boundary scan insertion is performed in a subsequent pass.

- tms: *port_pin_name* ;

  This property defines the port or the pin name to receive the IJTAG TMSPort function. The default is “tms”. This property is only used when the HostScanInterface is for a TAP scan interface. See create_dft_specification description to learn how it can be automatically set.

- tdi : *port_pin_name* ;

  This property defines the port or the pin name that is to source the IJTAG ScanInPort function. The default is “tdi”. This property is only used when the HostScanInterface is for a TAP scan interface. See create_dft_specification description to learn how it can be automatically set.

- tdi_local : *port_pin_name* ;

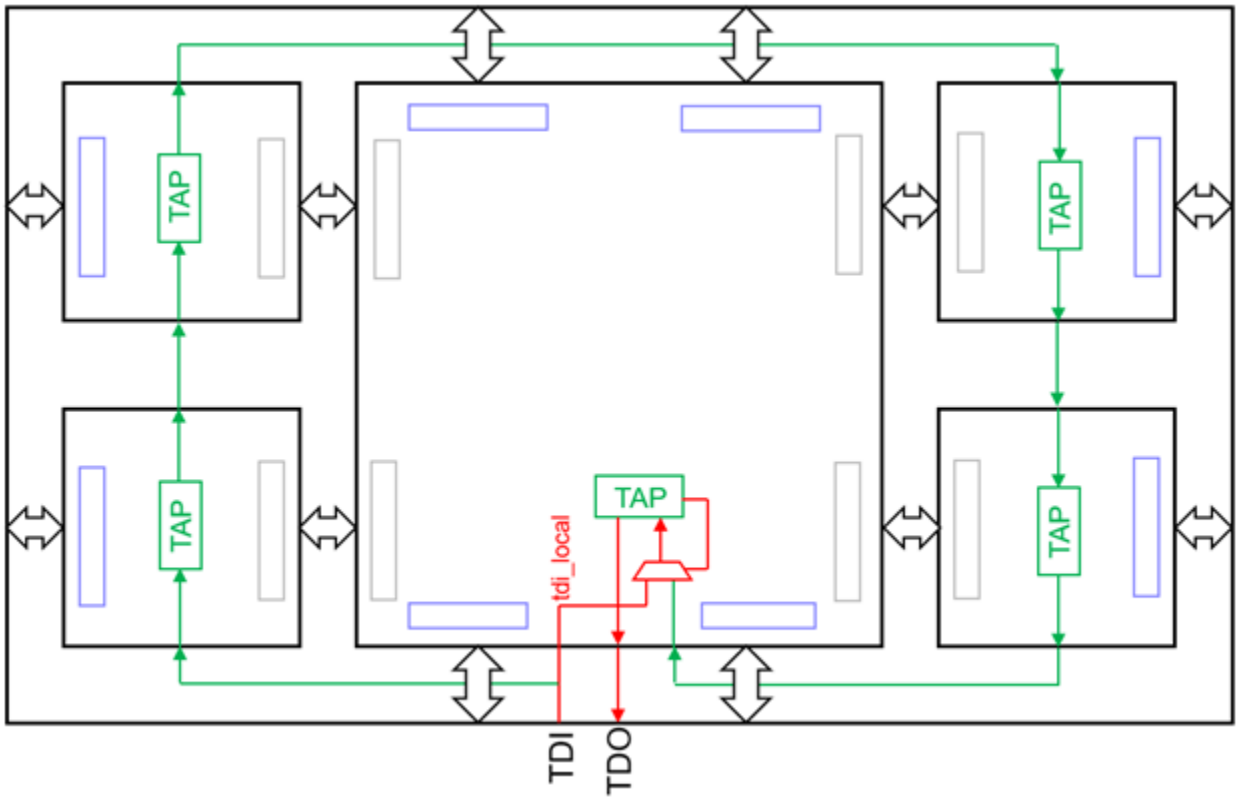
  This property defines a second TDI port for 2.5D designs, used when the TAP shifts through the BYPASS or DEVID register in the die closest to TDO in the package. Without a tdi_local port and tdi multiplexer, a 2.5D package is not IEEE 1149.1 compliant and the tool cannot generate a BSDL because the standard requires the package to have a one-bit BYPASS and a 32-bit DEVID register.

  The tdi_local property defaults to null. If you specify a value, the tool creates a multiplexer that selects between the normal TDI port and the specified TDI local port. Figure 13-8 shows that if the TAP shifts through the BYPASS or DEVID register, it selects the local TDI, which results in only a single one-bit bypass register and a single 32-bit device ID register being present between the TDI and TDO pin of the package. Specifying this property also creates two ports on the TAP controller to control the tdi local mux, called bypass_devid_select and ir_select by default. You can use the Tap/Interface/TdiMux wrapper to change the naming of these ports on the TAP controller and on the TdiMux

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

module. You can also use the Tap/Interface/TdiMux/parent_instance property to control where the tdi local mux is instantiated.

Figure 13-8. TDI Local Mux Shown in Die Closest to TDO in a 2.5D Package



- tdo : *port_pin_name* ;

  This property defines the port or the pin name to receive the IJTAG ScanOutPort function. The default is “tdo”. This property is only used when the HostScanInterface is for a TAP scan interface. See create_dft_specification description to learn how it can be automatically set.

- tdo_en : *port_or_pin_name* ;

  This property defines the port or the pin name to receive the tdo_en function. The default is “tdo_en”. This property is only used when the HostScanInterface is for a TAP scan interface and the tdo port is not driven by an output pad buffer in which case no tdo_en port is created as the tdo_en signal is connected to the enable pin of the output pad buffer.

- tdo_en_polarity : active_high | active_low | auto ;

  This property defines the polarity of the port or the pin object that receives the tdo_en function. This property is only used when the HostScanInterface is for a TAP scan interface. If the tdo port is driven by an output pad buffer, the tool infers tdo_en_polarity from the polarity of the enable on the TAP. It generates an error if the tdo_en_polarity property is specified to active_high and the enable pin of the output buffer is active_low and vice versa. No error is ever generated if specified to auto. If the tdo port is not driven by an output pad

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

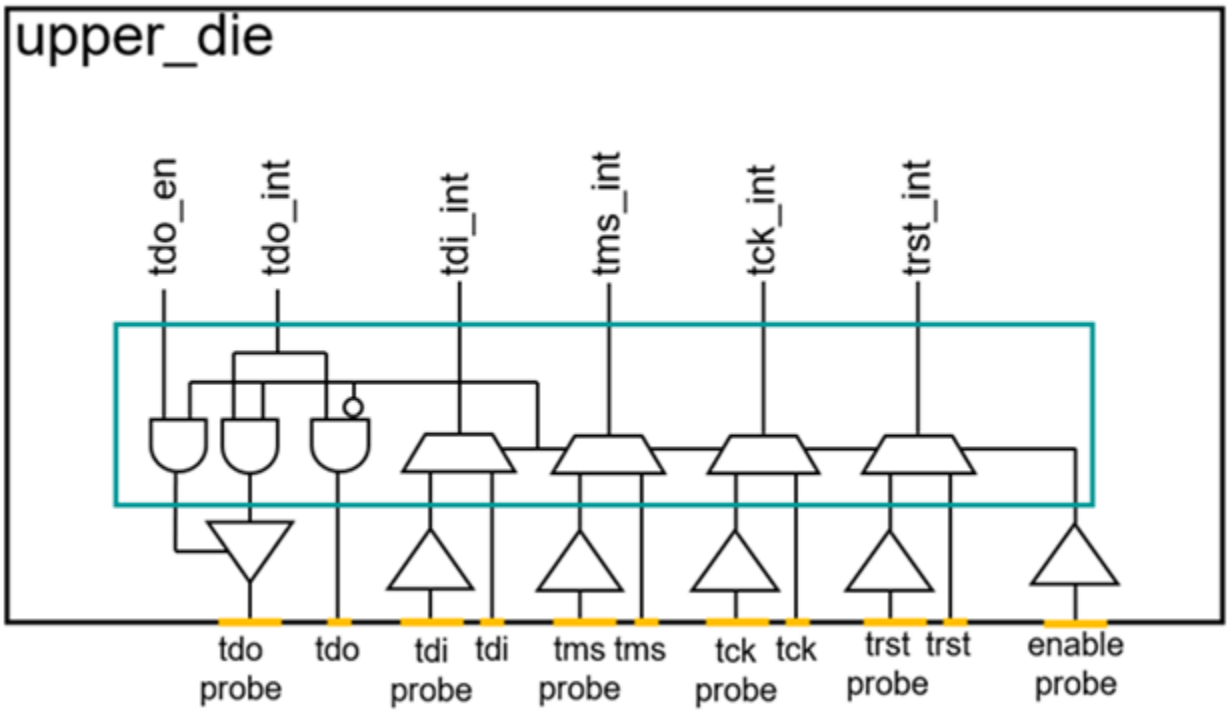
buffer, the tool creates a tdo_en port with the specified active polarity. The tool uses active_low polarity if the property is specified or defaults to auto.

- SacrificialPads

  An optional wrapper that adds multiplexers and sacrificial pads for the TAP pins used during probe test of a die going into a 3D stack. The presence of the wrapper triggers the insertion of a module containing the logic shown in Figure 13-9. The pad buffers must pre-exist in the design and already connect to the sacrificial pads. The presence of the SacrificialPads wrapper is mutually exclusive with the following features:

  - Using the tdi_local property.
  - Using within a HostScanInterface that is not of type TAP.
  - Setting or defaulting the daisy_chain_with_existing_client to on.
  - Using the ComplianceEnable/active_high_enables or ComplianceEnable/ active_low_enables properties.

Figure 13-9. Sacrificial Pad Multiplexing Module



- SacrificialPads/parent_instance : *instance_name* ;

  An optional property that specifies the parent instance in which to instantiate the sacrificial pad muxing module shown in Figure 13-9. By default, the tool instantiates the module in the current design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- SacrificialPads/leaf_instance_name : *instance_name* ;

  An optional property that specifies the leaf instance name of the sacrificial pad muxing module. The default name is *<design_name>*_*<design_id>*_tessent_sacrificial_pad_mux_*<host_scan_int_id>*_inst. The *<host_scan_int_id>* value is the id of the HostScanInterface wrapper that contains the Interface/SacrificialPads wrapper.

- SacrificialPads/<function> : *port_name*;

  An optional set of properties that specify the port names used during probe test for the associated function. The port name defaults to <function>_probe and the port must already be equipped with a pad buffer.

- SacrificialPads/tdo_en_polarity : active_high | active_low | auto ;

  An optional property that specifies the active polarity of the enable pin of the TDO pad buffer. When insertion is on, the tool automatically extracts polarity from the identified pad buffer. Use this property when you want to create the module in isolation and manually insert it later in a design flow.

- spio : *port_name*;

  This property defines the port name that is to be the TwoPinSerialInOutPort function. This property is only used when the HostScanInterface is for a TwoPinSerialPort interface. The input and output connections for the TwoPinSerialPort controller will be extracted from the pad description in your cell library.

- spclk : *port_pin_name*;

  This property defines the port or pin name that is to source the TwoPinSerialClockPort function. This property is only used when the HostScanInterface is for a TwoPinSerialPort interface.

- daisy_chain_with_existing_client : on | off | auto ;

  A string property affecting how the connection to the HostScanInterface is done. When set to off, the source of the scan_out pin is first disconnected before hooking up the scan-out port of the network to it.

  When set to on, it is assumed a valid scan-out source is already connected to the specified scan-out port. The source of the scan-out is used as the scan-in of the newly inserted IJTAG network.

  When set or left to default to auto, the scan-out port of the HostScanInterface is checked and if it seem to already have a source (its has_functional_source attribute is true), the On value is assumed otherwise the Off value is assumed.

- ComplianceEnable

  An optional wrapper that creates and instantiates a Tessent compliance enable module inside the IJTAG network. By default, it is instantiated next to the node of the first DesignInstance or Tap wrapper of the current HostScanInterface. The Tessent compliance enable module decodes all compliance enable signals, muxes the TDO signal, and gates the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TMS signal going to the nodes in this HostScanInterface. You would typically use the active_high_enables or active_low_enables property if you currently have a TAP controller connected to the control ports and want to use a Tessent TAP for boundary scan tests. Unlike using a ScanMux for this purpose, the existing TAP does not need to be described in ICL. The expectation of Tessent Shell and the IEEE 1687 standard is that all other TAPs sharing the control ports have their TMS signals gated low when all compliance enable signals are active. If this is not the case, it is suggested that you use a ScanMux configuration instead.

- ComplianceEnable/active_high_enables : *port_pin_list*, ... ;

  This property defines active high ports or pins that must be enabled to select the scan path of the HostScanInterface. Use this property only when the HostScanInterface is for a TAP scan interface and is not defined using the design_instance and scan_interface properties. See the ComplianceEnable wrapper description for more details.

- ComplianceEnable/active_low_enables : *port_pin_list*, ... ;

  This property defines active low ports or pins that must be enabled to select the scan path of this HostScanInterface. Use this property only when the HostScanInterface is for a TAP scan interface and is not defined using the design_instance and scan_interface properties. See the ComplianceEnable wrapper description for more details.

- ComplianceEnable/parent_instance : *instance_name* ;

  An optional property that specifies the parent instance where Tessent Shell instantiates the compliance enable module. By default, the tool instantiates the compliance enable module next to the node of the first DesignInstance or Tap wrapper of the current HostScanInterface. The parent module must not be a hard module instance (“is_hard_module” attribute = true).

- ComplianceEnable/leaf_instance_name : *leaf_instance_name* ;

  An optional property that specifies the leaf instance name of the compliance enable module. The property defaults to *design_name_dft_spec_id*_tessent_compliance_enable_tap_inst when unspecified. The *dft_spec_id* portion is defined by the DftSpecification *id* string.

Examples

Example 1

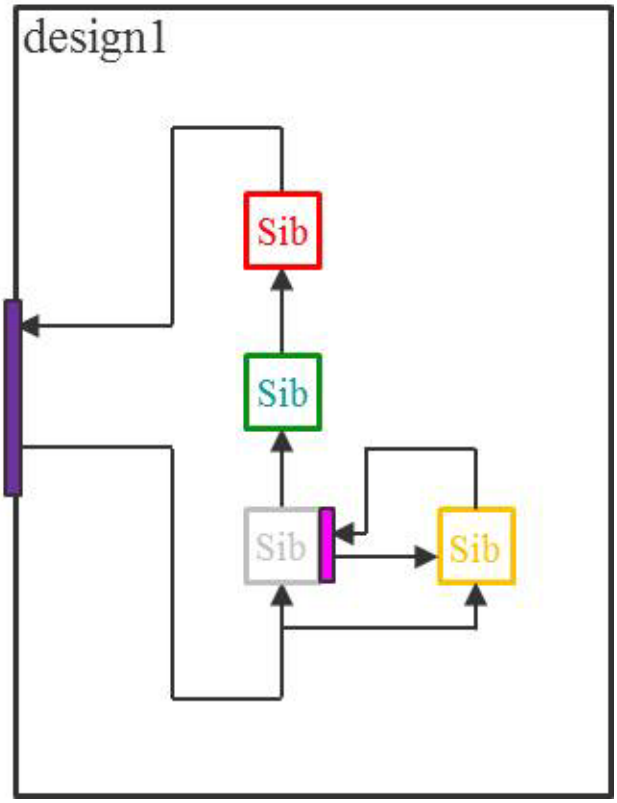
The following example has three HostScanInterface wrappers. The first wrapper called purple is used to insert a SIB called red in series with whatever is already connected to the purple HostScanInterface (the gray SIB in Figure 13-10). The second wrapper, called pink, is used to connect a SIB called orange to the Host ScanInterface of an existing instance in the design which, in this example, is a SIB inserted in an initial insertion pass. The third wrapper, called gray, is used to connect a SIB called green in series with the Client ScanInterface of an existing instance in the design which, in this example, is a SIB inserted in an initial insertion pass.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(design1,pass2) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Interface {
        scan_in    : si ;
        scan_out   : so ;
        capture_en : ce ;
        shift_en   : se ;
        update_en  : ue ;
        reset      : rstn ;
        select     : sel ;
        daisy_chain_with_existing_client : on ;
      }
      Sib(red) {
      }
    }
    HostScanInterface(pink) {
      Interface {
        scan_in    : si ;
        scan_out   : tessent_pass1_sib_first_inst/ijtag_from_so ;
        capture_en : ce ;
        shift_en   : se ;
        update_en  : ue ;
        reset      : rstn ;
        select     : tessent_pass1_sib_first_inst/ijtag_to_sel ;
      }
      Sib(orange) {
      }
    }
    HostScanInterface (gray){
      Interface {
        design_instance  : tessent_pass1_sib_first_inst ;
        scan_interface   : client ;
      }
      Sib(green) {
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-10. HostScanInterface/Interface Example 1 Schematic



Example 2

This example shows how to connect a SIB to the TAP controller inserted by ETAssemble as part of one of the LV flows. The first DftSpecification wrapper uses the design_instance syntax. The second DftSpecification wrapper uses explicit function connections but results in the same implementation as the first one. Although the second method is more verbose, it works even when the connections are not all made to a single instance having an associated ICL module, which is a requirement for the first method.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(my_design,post_etassemble) {
  IjtagNetwork {
    HostScanInterface(ijtag){
      Interface {
        design_instance : LVISION_JTAP_INST ;
        scan_interface  : Bist4 ;
      }
      Sib(1) {
      }
    }
  }
}
DftSpecification(my_design,post_etassemble) {
  IjtagNetwork {
    HostScanInterface(ijtag){
      Interface {
        scan_in        : LVISION_JTAP_INST/tdi ;
        scan_out       : LVISION_JTAP_INST/fromBist4 ;
        select         : LVISION_JTAP_INST/bistEn4 ;
        capture_en     : LVISION_JTAP_INST/captureDR2Edge ;
        shift_en       : LVISION_JTAP_INST/shiftDR2Edge ;
        update_en      : LVISION_JTAP_INST/updateDREnable ;
        reset          : LVISION_JTAP_INST/testLogicResetInv ;
        reset_polarity : active_low ;
        tck            : LVISION_JTAP_INST/tck ;
        }
        Sib(1) {
        }
      }
    }
  }
```

Example 3

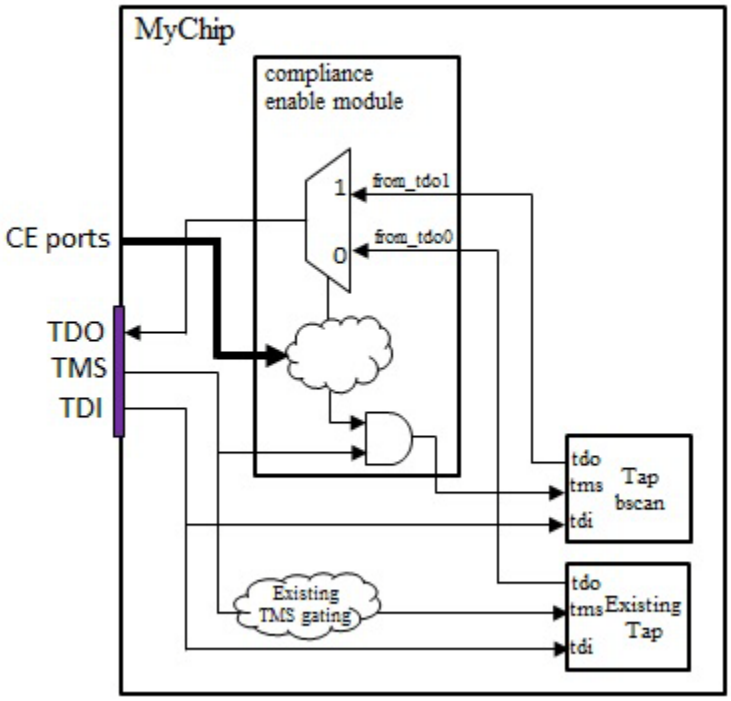
This example shows how to insert a TAP with boundary scan that shares ports with an existing TAP controller through compliance enable ports. Notice that the existing TAP has TMS gating low when the compliance enable values are not selecting it. If you are inserting boundary scan, the compliance enable ports are required to be documented as shown in the BoundaryScanCellOptions.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(MyChip,bscan_tap) {
  IjtagNetwork {
    HostScanInterface(tap) {
      Interface {
        tck  : TCK  ;
        trst : TRST ;
        tms  : TMS  ;
        tdi  : TDI  ;
        tdo  : TDO  ;
        ComplianceEnable {
          active_high_enables : CE0, CE1 ;
          active_low_enables : CE2, CE3 ;
        }
      }
      Tap(bscan) {
        HostIjtag(1) {
        }
        HostBscan {
        }
      }
    }
  }
  BoundaryScan {
    ijtag_host_interface : Tap(bscan)/HostBscan;
    BoundaryScanCellOptions {
      CE0 : compliance_enable1 ;
      CE1 : compliance_enable1 ;
      CE2 : compliance_enable0 ;
      CE3 : compliance_enable0 ;
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-11. HostScanInterface/Interface Example 3 Schematic



Related Topics

ScanMux/Interface

Tdr/Interface

Sib/Interface

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DesignInstance

A wrapper that specifies an instance in the design that has a ScanInterface to connect to.

Usage

```
IjtagNetwork  {
  HostScanInterface(id) {
    DesignInstance(instance_name) {
      scan_interface : scan_interface_name ;
    }
  }
}
```

Description

A wrapper that specifies an instance in the design that has a ScanInterface to connect to.

The *instance_name* string is a design instance name relative to the current design. When the process_dft_specification command is executed without the -no_insertion option, the tool verifies that the specified *instance_name* exists within the design. The tool also verifies that the associated design module has a matched ICL module with at least one ScanInterface in it. If there is only one scan interface in the ICL module, you do not need to specify the scan_interface property, but if there is more than one ScanInterface, you must use the scan_interface property to identify which one to connect to.

DesignInstance nodes extract the KEEP_ACTIVE_DURING_SCAN_TEST attribute value from the ICL module definition. When unspecified, the attribute defaults to "true". In the rare case that a DesignInstance only contains ScanTestedInstruments, you can use the set_attribute_value command to set the attribute to “false” on the ICL module as follows:

```
set_attribute_value [get_icl_module <module_name>] -name \

 KEEP_ACTIVE_DURING_SCAN_TEST -value false
```

but if there is more than one ScanInterface, you must use the scan_interface property to identify which one to connect to.

ICL modules are automatically searched and matched to the design modules when the set_current_design command is executed. The search paths for the ICL files are specified using the set_design_sources -format icl command. The module name matching rules are specified using the set_module_matching_options command.

Arguments

- *instance_name*

  A string that refers to a design instance relative to the current design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- scan_interface : *scan_interface_name ;*

  A property that is used only when the referenced design instance is an instance of a module having more than one ScanInterface declared in its ICL module. You must repeat the DesignInstance wrapper for each *scan_interface_name* found in the ICL module, if you want to connect all of them to the network.

  If you have a design instance with multiple scan interfaces and they share a common scan-in port, you cannot connect them to a SIB because this would required independent scan-in ports. Instead, you must connect them all to inputs of a scan mux, so that you end up with one scan-in port and one scan-out port. This is illustrated in the example that follows.

Examples

The following example illustrates connecting the ScanInterface of a DesignInstance to a SIB. It also shows how a design instance that has three scan interfaces sharing a common scan-in port is first connected to two scan multiplexers before being connected to a SIB.

The select of the multiplexers are sourced by the DataOutPorts of the orange TDR which automatically sets its output_timing setting to “ijtag_scan_selection”. For a description and impact of this setting, see the Tdr wrapper description and specifically “Update Stage Timing” on page 3995.

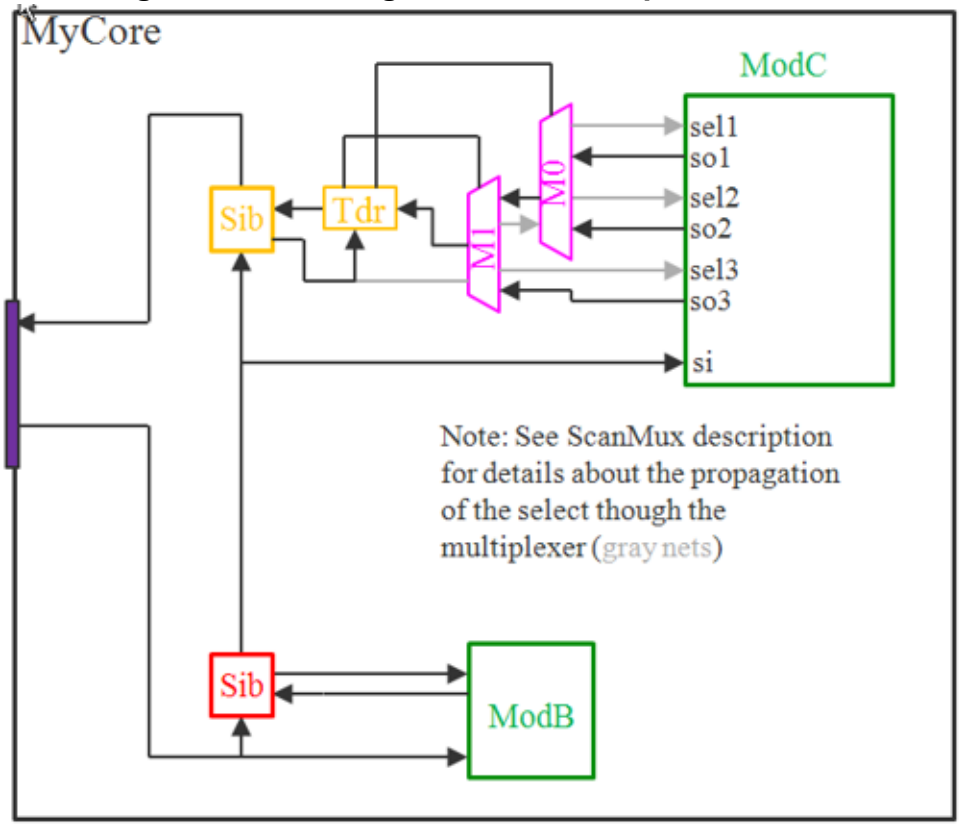
This example also shows the use of ScanMux in the network. It provides a hint about the backward propagation of the select signal though the selected path of the multiplexer. For a complete description of the ScanMux and the backward propagation of the select signals, refer to the ScanMux wrapper description section.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Sib(orange) {
        Tdr(orange) {
          DataOutPorts {
            Count : 2 ;
          }
        }
        ScanMux(M1) {
          Select : tdr(orange)/DataOut(1) ;
          Input(0) {
            ScanMux(M0) {
              Select : tdr(orange)/DataOut(0) ;
              Input(0) {
                DesignInstance (u3/top_green) {
                  scan_interface : I1 ;
                }
              }
              Input(1) {
                DesignInstance (u3/top_green) {
                  scan_interface : I2 ;
                }
              }
            }
          }
          Input(1) {
            DesignInstance (u3/top_green) {
              scan_interface : I3 ;
            }
          }
        }
      }
      Sib(red) {
        DesignInstance (u1/bottom_green) {}
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-12. DesignInstance Example Schematic



Related Topics

HostScanInterface/Interface

Sib

ScanMux

Tdr

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ScanMux

A wrapper that specifies the creation and, optionally, the insertion of a scan multiplexer into the design.

Usage

```
IjtagNetwork {
  HostScanInterface(id) {
    ScanMux(id) {
      Interface {
      }
      parent_instance    : instance_name ;
      leaf_instance_name : leaf_instance_name ;
      select             : source ;
      Attributes {
        attribute_name   : attribute_value ;
      }
      Input (integer) {                    // legal: 0 1
        keep_active_during_scan_test : on | off | auto ;
        Sib(id) {                          // repeatable
        }
        Tap(id) {                          // repeatable
        }
        Tdr(id) {                          // repeatable
        }
        ScanMux(id) {                      // repeatable
        }
        DesignInstance(instance_name) {    // repeatable
        }
        SecondaryHostScanInterface(id){    // repeatable
        }
      }
    }
  }
}
```

Description

A wrapper that specifies the creation and, optionally, the insertion of a scan multiplexer into the design. The Input(0) and Input(1) wrappers are used to defined the nodes that feed into that input of the multiplexer. If you omit an Input wrapper, the multiplexer simply bypasses the elements connected to the other input when the unspecified input is selected. See the example that follows for an illustration of this feature.

The actual schematic of the multiplexer is shown in Figure 13-13 and Figure 13-14. As you can see, the scan multiplexer is actually more than a simple multiplexer. When used in a DR scan path, it also has logic to supply a select line (output_en1 and output_en0) for each set of clients connected to the two scan inputs (input1 and input0). When used in a TAP scan path, it also has logic to supply a to_tms (to_tms1 and to_tms0) for each set of tap clients connected to the two scan inputs (input1 and input0).The output port is the multiplexed scan-out data of the clients to source the scan-in of the host or of the next node in series with the ScanMux.

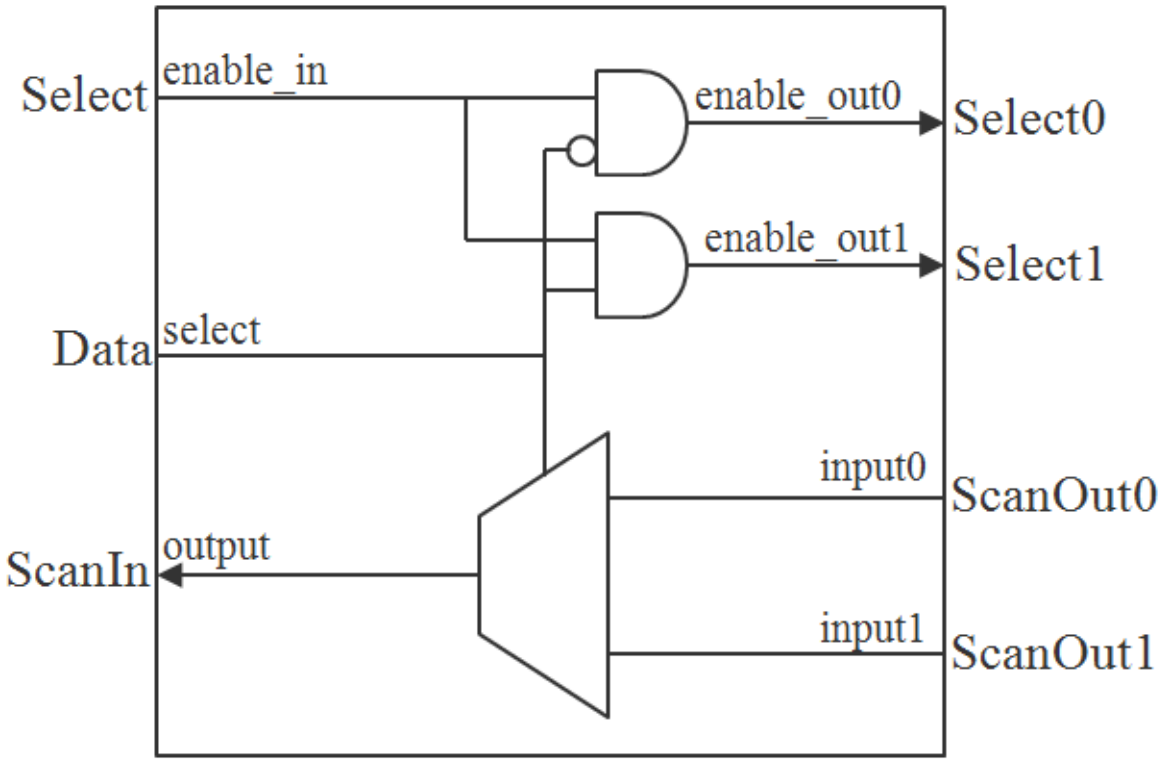
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The select input is sourced by a data signal which is either one of the IjtagNetwork/DataInPorts of the IjtagNetwork, one of the IjtagNetwork/DataOutPorts of a Tdr or a Tap, or an output port on an existing design instance. The select input is used to control the multiplexer and to create two gated versions of the enable_in or tms inputs. The gated versions are used as the select or tms source of the clients connected to each scan input. The enable_in port is sourced by the host of the ScanMux. If the host is a Sib, the enable_in is sourced by the to_select port. If the host is another ScanMux, the enable_in port is sourced by one of the "enable_out#" ports. If the host is the HostScanInterface/Interface, the enable_in is sourced by the select port. The tms port is sourced by the host of the ScanMux. If the host is a ScanMux, the tms port is sourced by one of the "to_tms#" ports. If the host is the HostScanInterface, the tms is sourced by the tms port.

When the reuse_modules_when_possible property is set or inferred to be on in the DftSpecification wrapper, the ScanMux module names are hardcoded to *design_name_design_id*_tessent_scanmux_# where # is an integer incrementing from 1. A unique module is created for each combination of port names in the interface wrapper, attributes in the Attributes wrapper, and the value of the keep_active_during_scan_test property.

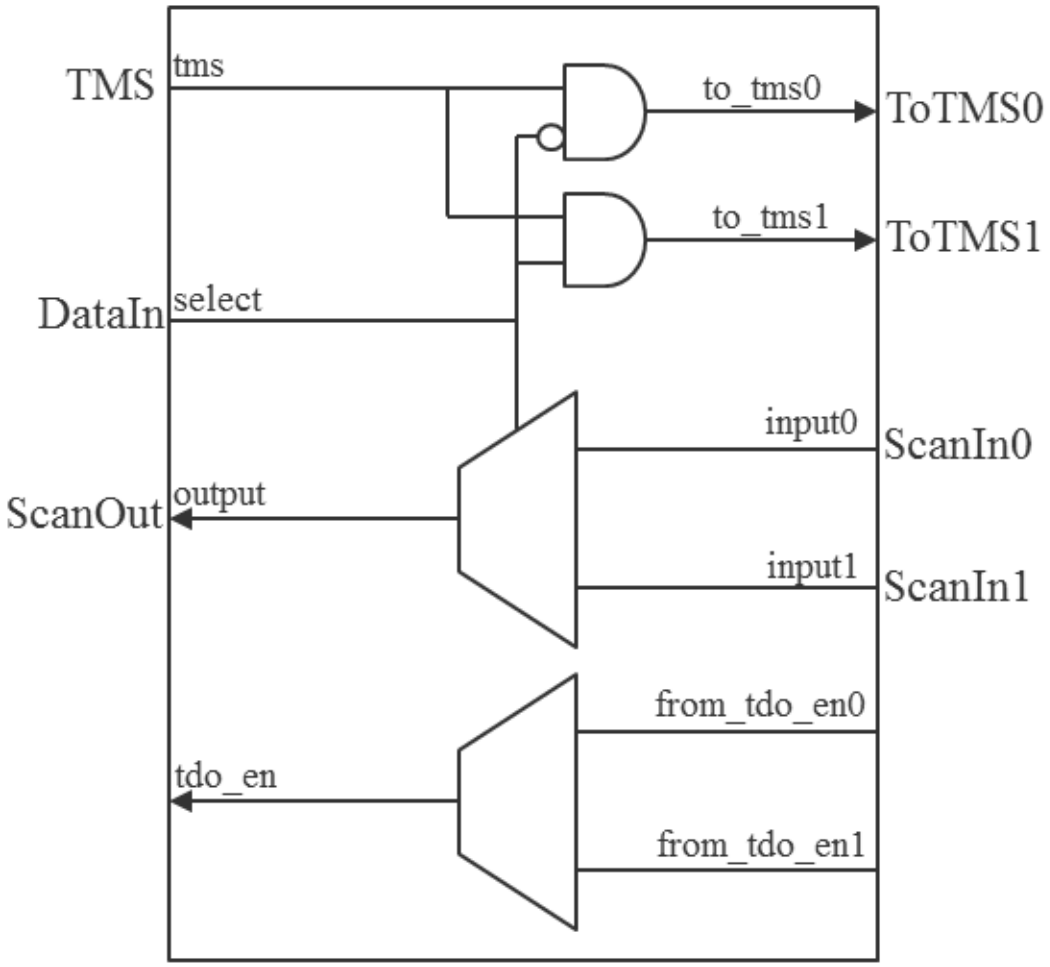
When the reuse_modules_when_possible property is set or inferred to be off in the DftSpecification wrapper, the ScanMux module names are hardcoded to *design_name_design_id*_tessent_scanmux_*id*.

Figure 13-13. ScanMux Internal Schematic When Used on a DR Scan Path



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-14. ScanMux Internal Schematic When Used on a TAP Scan Path



Arguments

- *id*

  A unique id that contains only letters, numbers, and underscores. Every ScanMux wrapper must have a different id inside a common IjtagNetwork wrapper.

- parent_instance : *instance_name* ;

  A string property that specifies the parent instance in which to instantiate the scan mux module. If this property is not unspecified, the scan mux is instantiated in the root module. You can use this property to push a scan multiplexer down near the client interfaces it interacts with.

  If the specified instance_name is an instance of a repeated module, a warning is displayed during validation and a uniquify_instances command is automatically run as part of the insertion process to uniquify the instance_name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

To perform IJTAG network insertion inside a module that is used more than once without uniquifying it, you can set the current design to that module, insert the IJTAG network in it, and refer to it with a DesignInstance wrapper at the next level up. This bottom-up insertion process can be done by going back and forth between setup and insertion mode within a single Tessent Shell invocation. You simply change the current design using the set_current_design command in setup mode.

- leaf_instance_name : *leaf_instance_name* ;

  A string property that specifies the leaf instance name of the scan multiplexer. The property defaults to *design_name_dft_spec_id*_tessent_scanmux_*scanmux_id*_inst when unspecified. The *dft_spec_id* string is the id specified in the DftSpecification wrapper, and the *scanmux_id* is the id of the ScanMux wrapper. Tessent Shell checks for conflicts before instantiating the scan multiplexer and, if needed, applies the uniquification suffix specified by the set_insertion_options command.

- select : *source* ;

  A string property that specifies the source of the select. The source must match one of the formats described in this table.

Format	Description
DataIn(integer)	Specifies that the source of the select is one of the IjtagNetwork/DataInPorts of the IjtagNetwork wrapper. The integer value must be between count -1 and 0 where count is a property that specifies the number of DataInPorts.
Tdr(id)/DataOut(integer)	Specifies that the source of the select is one of the IjtagNetwork/DataOutPorts of a Tdr. The integer value must be between count -1 and 0 where count is a property that specifies the number of DataOutPorts on the Tdr(id).
Tdr(id)/DecodedSignal(name)	Specifies that the source of the select is one of the Tdr/DecodedSignal of a Tdr. The name value must be the name of a decoded signal on the Tdr(id).
Tap(id)/DataOut(integer)	Specifies that the source of the select is one of the IjtagNetwork/DataOutPorts of a Tap. The integer value must be between count -1 and 0 where count is a property that specifies the number of DataOutPorts on the Tap(id).
DesignInstance(inst_name)/port_name	Specifies that the source of the select is a port on the module with the instance name inst_name. During ICL extraction, this pin must be or must trace to a DataOutPort or a ToSelect port of an ICL module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Attributes/attribute_name : *attribute_value* ;

  A data wrapper that defines arbitrary attributes in the ScanMux ICL module. This mechanism is useful for finding special attributed modules in the concatenated ICL, at a later time. The attribute_name can be any identifier starting with a letter and followed by any number of letters, numbers, and underscores. The attribute_value can be any string or integer. If you want to define a personal attribute and later be able to introspect its value using get_attribute_value_list, you must have registered the attribute using the "register_attribute -object_type icl_module" command. For example, assuming you have registered the attribute att1 on the icl_module object type, and you have added the attribute att1 with a value of "v1" in some of your ICL modules using the Attributes wrapper, you can later quickly find those modules using the "get_icl_modules -filter att1==v1" command.

- Input(0|1)/keep_active_during_scan_test : on | off | auto ;

  A property that specifies that all clients connected to the ScanMux input are to be kept active during scan test modes. You should rarely need to use this property because when it is auto (default), it is automatically set to on if a Tdr with keep_active_during_scan_test set to on traces to it. See section "Scan Testability of the IJTAG Network" on page 3928 for information about when to use this property.

Examples

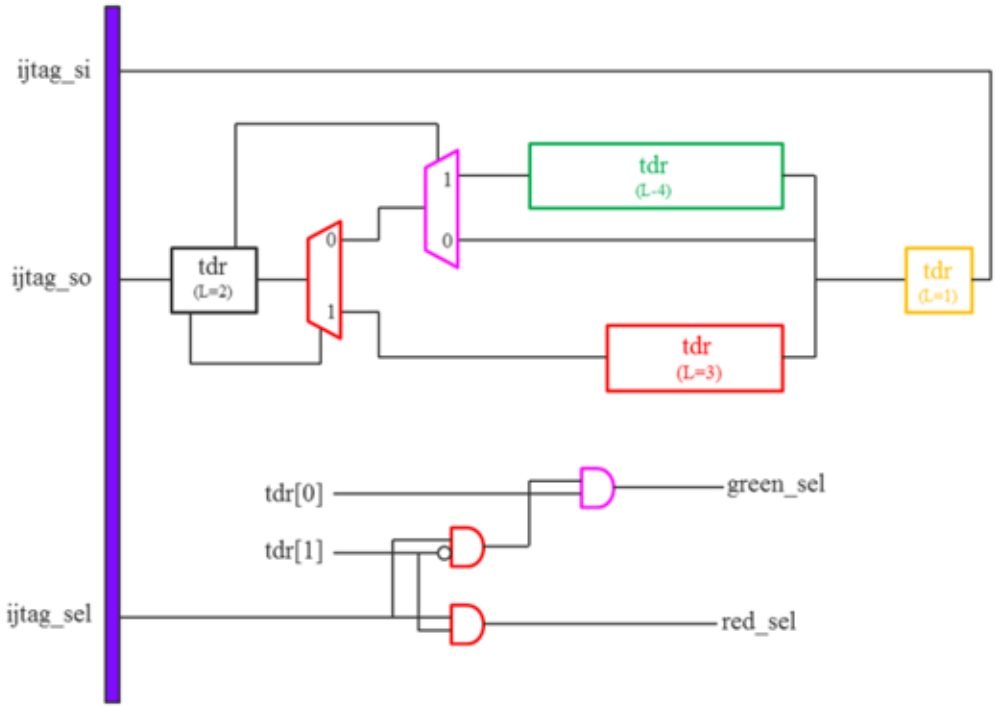
The following example illustrates the use of scan multiplexers in the DR scan paths of the IJTAG network. See Figure 13-28 for an example use in the TAP scan path. The red mux has client nodes specified in both of its Input wrappers which make it a selection multiplexer. The pink mux only has client nodes in its "1" Input wrapper making it a bypass multiplexer. The clients connected to the "1" input are bypassed when the select is low. For both multiplexers, a data-out bit on the black TDR is used as the source of the select signal.

The AND gates shown at the bottom of Figure 13-15 illustrate the distributed gating mechanism used to generate the select signals for the multiplexed TDRs. These AND gates correspond to the AND gates shown in Figure 13-13.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Tdr(black) {
        DataOutPorts {
          count : 2 ;
        }
      }
      ScanMux(red) {
        select : tdr(black)/DataOut(1) ;
        Input(1) {
          Tdr(red) {
            length : 3 ;
          }
        }
        Input(0) {
          ScanMux(pink) {
            select : tdr(black)/DataOut(0) ;
            Input(1) {
              Tdr(green) {
                length : 4 ;
              }
            }
          }
        }
      }
      Tdr(orange) {
      }
    }
  }
}
```

Figure 13-15. ScanMux Example Schematic



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

DesignInstance

Sib

Tdr

HostScanInterface/Interface

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ScanMux/Interface

A wrapper that specifies the name of the ports when creating the ScanMux module.

Usage

```
IjtagNetwork {
  HostScanInterface(id) {
    ScanMux(id) {
      Interface {
        select       : port_name ; // default: mux_select
        input0       : port_name ; // default: mux_in0
        input1       : port_name ; // default: mux_in1
        output       : port_name ; // default: mux_out
        enable_in    : port_name ; // default: enable_in
        enable_out0  : port_name ; // default: enable_out0
        enable_out1  : port_name ; // default: enable_out1
        tms          : port_name ; // default: tms
        to_tms0      : port_name ; // default: to_tms0
        to_tms1      : port_name ; // default: to_tms1
        tdo_en       : port_name ; // default: tdo_en
        from_tdo_en0 : port_name ; // default: from_tdo_en0
        from_tdo_en1 : port_name ; // default: from_tdo_en1
      }
    }
  }
}
```

Description

A wrapper that specifies the name of the ports when creating the ScanMux module.

All port names can be specified by the user; however, they can only be simple scalar port names.

Arguments

- select : *port_name* ;

  This property defines the name of the port labelled “select” in Figure 13-13 and Figure 13-14. The default is “mux_select”.

- input0 : *port_name* ;

  This property defines the name of the port labelled “input0” in Figure 13-13 and Figure 13-14. The default is “mux_in0”.

- input1 : *port_name* ;

  This property defines the name of the port labelled “input1” in Figure 13-13 and Figure 13-14. The default is “mux_in1”.

- output : *port_name* ;

  This property defines the name of the port labelled “output” in Figure 13-13 and Figure 13-14. The default is “mux_out”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- enable_in : *port_name* ;

  This property defines the name of the port labelled “enable_in” in Figure 13-13. The default is “enable_in”.

- enable_out0 : *port_name* ;

  This property defines the name of the port labelled “enable_out0” in Figure 13-13 and Figure 13-14. The default is “enable_out0”.

- enable_out1 : *port_name* ;

  This property defines the name of the port labelled “enable_out1” in Figure 13-13. The default is “enable_out1”.

- tms : *port_name* ;

  This property defines the name of the port labelled “tms” in Figure 13-14. The default is “tms”.

- to_tms0 : *port_name* ;

  This property defines the name of the port labelled “to_tms0” in Figure 13-14. The default is “to_tms0”.

- to_tms1 : *port_name* ;

  This property defines the name of the port labelled “to_tms1” in Figure 13-14. The default is “to_tms1”.

- tdo_en : *port_name* ;

  This property defines the name of the port labelled “tdo_en” in Figure 13-14. The default is “tdo_en”.

- from_tdo_en0 : *port_name* ;

  This property defines the name of the port labelled “from_tdo_en0” in Figure 13-14. The default is “from_tdo_en0”.

- from_tdo_en1 : *port_name* ;

  This property defines the name of the port labelled “from_tdo_en1” in Figure 13-14. The default is “from_tdo_en1”.

Examples

The following example illustrates the use of this wrapper to specify custom port names for the scan multiplexer used in a DR Scan Path.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
IjtagNetwork {
   HostScanInterface(id) {
      ScanMux(id) {
         Interface {
            select      : sm_select ;
            input0      : sm_si0 ;
            input1      : sm_si1 ;
            output      : sm_so ;
            enable_in   : sm_en ;
            enable_out0 : sm_en0 ;
            enable_out1 : sm_en1 ;
         }
      }
   }
}
```

Related Topics

HostScanInterface/Interface

Tdr/Interface

Sib/Interface

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SecondaryHostScanInterface

A wrapper that specifies the connection of an entire scan interface to the boundary of the current design.

Usage

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(id) {
      SecondaryHostScanInterface(id) {
        to_scan_in    : port_pin_name ; //def: %s_ijtag_to_si    *DefSpec
        from_scan_out : port_pin_name ; //def: %s_ijtag_from_so  *DefSpec
        to_select     : port_pin_name ; //def: %s_ijtag_to_sel   *DefSpec
        to_shift_en   : port_pin_name ; //def: %s_ijtag_to_se    *DefSpec
        to_capture_en : port_pin_name ; //def: %s_ijtag_to_ce    *DefSpec
        to_update_en  : port_pin_name ; //def: %s_ijtag_to_ue    *DefSpec
        to_reset      : port_pin_name ; //def: %s_ijtag_to_reset *DefSpec
        to_reset_polarity  : active_low | active_high ;          *DefSpec

        to_tck        : port_pin_name ;  //def: %s_ijtag_to_tck  *DefSpec

        to_tdi        : port_pin_name ;  //def: %s_to_tdi        *DefSpec
        from_tdo      : port_pin_name ;  //def: %s_from_tdo      *DefSpec
        from_tdo_en   : port_pin_name ;  //def: %s_from_tdo_en   *DefSpec
        to_tms        : port_pin_name ;  //def: %s_to_tms        *DefSpec
        to_trst       : port_pin_name ;  //def: %s_to_trst       *DefSpec
      }
    }
  }
}
```

Description

The SecondaryHostScanInterface is the virtualization of more IJTAG nodes outside of the current design. Use this wrapper to bring the connections of a host port up to the boundary.

To run simulations through a SecondaryHostScanInterface, you must add loopback connections between the ScanOutPort and the ScanInPort. See the add_loadboard_loopback_pairs command for pattern sets and the LoadBoardInfo wrapper for PatternsSpecification.

Arguments

- to_scan_in

  This property defines the port or the pin name to receive the IJTAG ScanOutPort function. Use it for IJTAG type interfaces only. The default name is “%s_ijtag_to_si”, where %s is replaced by the ID of the wrapper. It is the data going to the nodes connected to the secondary host scan interface.

- from_scan_out

  This property defines the port or the pin name to receive the IJTAG ScanInPort function. Use it for IJTAG type interfaces only. The default name is “%s_ijtag_from_so”, where %s

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is replaced by the ID of the wrapper. It is the data coming back from the nodes connected to the secondary host scan interface.

- to_select

  This property defines the port or the pin name to receive the IJTAG ToSelectPort function. Use it for IJTAG type interfaces only. The default name is “%s_ijtag_to_sel”, where %s is replaced by the ID of the wrapper.

- to_shift_en

  This property defines the port or the pin name to receive the IJTAG ToShiftEnPort function. Use it for IJTAG type interfaces only. The default name is “%s_ijtag_to_se”, where %s is replaced by the ID of the wrapper.

- to_capture_en

  This property defines the port or the pin name to receive the IJTAG ToCaptureEnPort function. Use it for IJTAG type interfaces only. The default name is “%s_ijtag_to_ce”, where %s is replaced by the ID of the wrapper.

- to_update_en

  This property defines the port or the pin name to receive the IJTAG ToUpdateEnPort function. Use it for IJTAG type interfaces only. The default name is “%s_ijtag_to_ue”, where %s is replaced by the ID of the wrapper.

- to_reset

  This property defines the port or the pin name to receive the IJTAG ToResetPort function. Use it for IJTAG type interfaces only. The default name is “%s_ijtag_to_reset”, where %s is replaced by the ID of the wrapper.

- to_reset_polarity

  This property defines the polarity of the to_reset property. Use it for IJTAG type interfaces only. Specify active_low or active_high polarity. The default is active_low.

- to_tck

  This property defines the port or the pin name to receive the IJTAG ToTCKPort function. The default name is “%s_ijtag_to_tck”, where %s is replaced by the ID of the wrapper.

- to_tdi

  This property defines the port or the pin name to receive the IJTAG ScanOutPort function. Use it for TAP type interfaces only. The default name is “%s_to_tdi”, where %s is replaced by the ID of the wrapper.

- from_tdo

  This property defines the port or the pin name to receive the IJTAG ScanInPort function. Use it for TAP type interfaces only. The default name is “%s_from_tdo”, where %s is replaced by the ID of the wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- from_tdo_en

  This property defines the port or the pin name to receive the tdo_en function. Use it for TAP type interfaces only. The default name is from “%s_from_tdo_en”, where %s is replaced by the ID of the wrapper.

- to_tms

  This property defines the port or the pin name to receive the IJTAG ToTMSPort function. Use it for TAP type interfaces only. The default name is “%s_to_tms”, where %s is replaced by the ID of the wrapper.

- to_trst

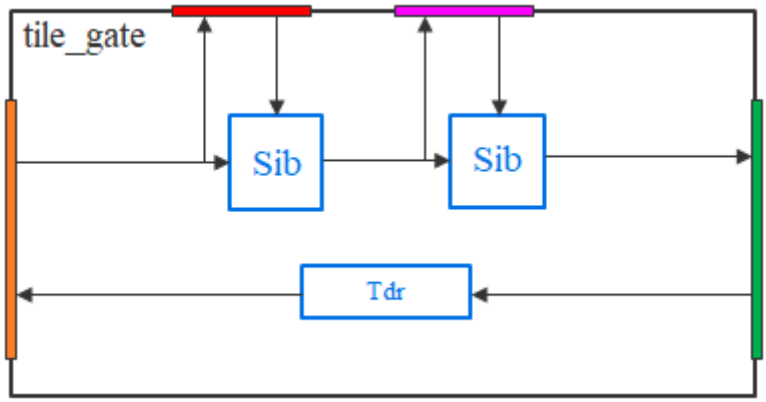
  This property defines the port or the pin name to receive the IJTAG ToTRSTPort function. Use it for TAP type interfaces only. The default name is “%s_to_trst”, where %s is replaced by the ID of the wrapper.

Examples

See Example 3 in IjtagNetwork for a full chip tile example.

The following example shows you how to use the SecondaryHostScanInterface in a tiling context. The main HostScanInterface is on the left (orange), and its scan path goes through both SIBs, out of the SecondaryHostScanInterface into a side network on the right (green), then back into the control TDR. This type of network configuration supports many of these “tile_gate” connections. This example also shows the SecondaryHostScanInterfaces at the top (red and magenta) that bring the complete host port of the SIBs to connect to cores.

Figure 13-16. Example Use of SecondaryHostScanInterface for Tiling



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(tile_gate,rtl) {
   IjtagNetwork {
     HostScanInterface(ijtag_client) {
       Tdr(tile_config) {
         leaf_instance_name: tdr_tile_config;
         DataOutPorts {
           count: 2;
         }
       }
       SecondaryHostScanInterface(ijtag_to_next_client) {
       }
       Sib(2) {
         keep_active_during_scan_test : on;
         SecondaryHostScanInterface(tile_core_gate_2) {
           to_tck        : to_tck2;
           to_shift_en   : ijtag_to_se2;
           to_capture_en : ijtag_to_ce2;
           to_update_en  : ijtag_to_ue2;
           to_select     : ijtag_to_sel2;
           to_reset      : ijtag_to_reset2;
           to_scan_in    : ijtag_to_si2;
           from_scan_out : ijtag_from_so2;
         }
       }
       Sib(1) {
         keep_active_during_scan_test : on;
         SecondaryHostScanInterface(tile_core_gate_1) {
           to_tck        : to_tck1;
           to_shift_en   : ijtag_to_se1;
           to_capture_en : ijtag_to_ce1;
           to_update_en  : ijtag_to_ue1;
           to_select     : ijtag_to_sel1;
           to_reset      : ijtag_to_reset1;
           to_scan_in    : ijtag_to_si1;
           from_scan_out : ijtag_from_so1;
         }
       }
     }
   }
 }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Sib

A wrapper that specifies the creation and, optionally, the insertion of a Segment Insertion Bit (SIB) into the IJTAG network.

Usage

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(id) {
      Sib(id) {
        Interface  {
        }
        Connections {
           clock_sequence : port_pin_constant_name, ... ; // default: 0
        }
        provide_full_host_scan_interface: auto | on | off ;
        to_scan_in_feedthrough          : auto | none | buffer | pipeline;
                                          // *DefSpec
        parent_instance                 : instance_name ;
        leaf_instance_name              : leaf_instance_name ;
        keep_active_during_scan_test    : on | off | auto ;
        capture_value                   : 0 | 1 | self ;
        so_retiming                     : on | off ;
        Attributes {
          attribute_name                : attribute_value ;
        }
        Sib(id) {                         // repeatable
        }
        Tdr(id) {                         // repeatable
        }
        ScanMux(id) {                     // repeatable
        }
        DesignInstance(instance_name) {   // repeatable
        }
        SecondaryHostScanInterface(id){   // repeatable
        }
      }
    }
  }
}
```

Description

A SIB is a special node in IJTAG that contains a single bit shift register. When the bit is zero, the SIB bypasses the client nodes connected to its host port. When the bit is one, the SIB includes the client nodes connected to its host port as part of the active scan path. Figure 13-17 on page 3981 shows the detailed schematic of its implementation.

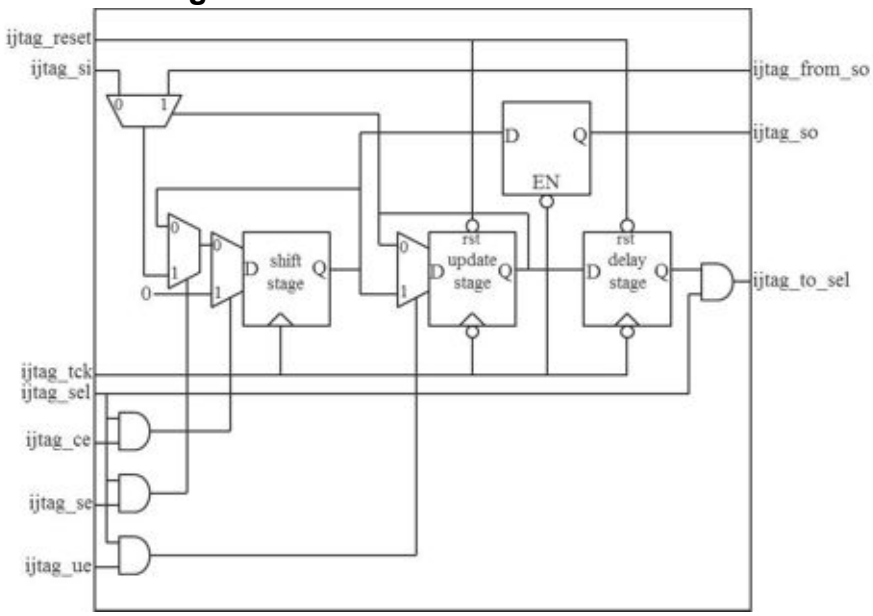
The shift register value is latched into the update stage on the falling edge of TCK when update_en is active. The transition of to_select_en is delayed by one TCK cycle to avoid timing races that exist if TCK is not perfectly balanced between the SIB and its clients. Refer to "Update Stage Timing" on page 3995 for more details about this feature.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When the reuse_modules_when_possible property is set to on or inferred to be on in the DftSpecification wrapper, the SIB module names are hardcoded to *design_name_design_id*_tessent_sib_# where # is an integer incrementing from 1. The tool creates a unique module for each combination of port names in the Interface wrapper, attributes in the Attributes wrapper, and the value of the capture_value and keep_active_during_scan_test properties.

When the reuse_modules_when_possible property is set to off or inferred to be off in the DftSpecification wrapper, the SIB module names are hardcoded to *design_name_design_id*_tessent_sib_*id*.

Figure 13-17. SIB Internal Schematic



As described in the create_dft_specification command description, the instruments that need to remain active during scan are inserted under one SIB, and the ones which are scan tested, such as a memory BIST controller, are inserted under another SIB. This SIB is identified with a tessent_dft_function attribute set to “scan_tested_instrument_host” and is commonly referred to as the Sib(sti). When this attribute is defined as Sib(sti), the SIB is equipped with the added logic shown in Figure 13-18 on page 3983 such that the IJTAG network that is scan tested is fully exercised by the scan test. The Sib(sti) is also equipped with a mini-OCC used to give control to ATPG and LogicBIST as to when the scan tested TCK domain is to receive clock pulses. See Figure 13-18 and Figure 13-20 for schematic views of the scan isolation logic and the mini-OCC. The ltest_to_en, ltest_to_mem_bypass_en, ltest_to_mcp_bounding_en, and ltest_to_se output ports are used during Memory BIST insertion to pre-connect its logic test ports in order to meet logic test design rules (DRCs) and enable memory bypass and RAM sequential testing. The ltest_to_mcp_bounding_en port is useful if you want to run LogicBIST on the memory BIST logic at speed.

The logic test input signals shown on the lower left side of the Sib(sti) in Figure 13-18 are tied off when the Sib(sti) is inserted. You need to drive those inputs and pick up the sub-chain

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

during scan insertion to achieve high fault coverage of the scan testable portion of the IJTAG network. If you use the add_dft_signals command to create the scan_en, ltest_en, memory_bypass_en, ltest_to_mcp_bounding_en, tck_occ_en and shift_capture_clock DFT signals, the corresponding input ports on the Sib(sti) are automatically connected to them as they become available. You then benefit from all the automation provided by the set_static_dft_signal_values and add_core_instances commands to set up the Sib(sti) for logic test and to configure the mini-OCC.

The ltest_capture_en port is left tied on if the mini-OCC for the Sib(sti) uses capture_en as the trigger to generate capture pulses. If the mini-OCC for the Sib(sti) uses shift_en as the trigger to generate capture pulses, the ltest_capture_en port is not created (see Figure 13-20).

By default, the ltest_static_control_clock_mode and ltest_clock_sequence ports are left tied off unless the LogicBist wrapper is used, in which case they are driven by the LogicBIST controller. You can also specify the connection point for ltest_clock_sequence with the clock_sequence property. The tool also connects the ltest_static_control_clock_mode to the lbist_en DFT signal when it becomes available at the current design level.

The mini-OCC enables testing of the scan tested instruments under the Sib(sti) (for example, MBIST hardware) given that this clock domain is different from the clock domains driving the functional logic and that the test clock driving the STI logic during logic test is usually also the trigger clock for the functional OCCs in fast capture mode (at-speed test). If you forget to add the tck_occ_en DFT signal or if you choose to not activate the TCK OCC during logic test, the TCK OCC still injects the shift clock into the TCK domain during the load_unload sequence but it does not pulse the TCK domain during the capture sequence. The IJTAG values feeding the functional domains have known values, but any value observed by the TCK domain is not captured and results in “ATPG Untestable” faults.

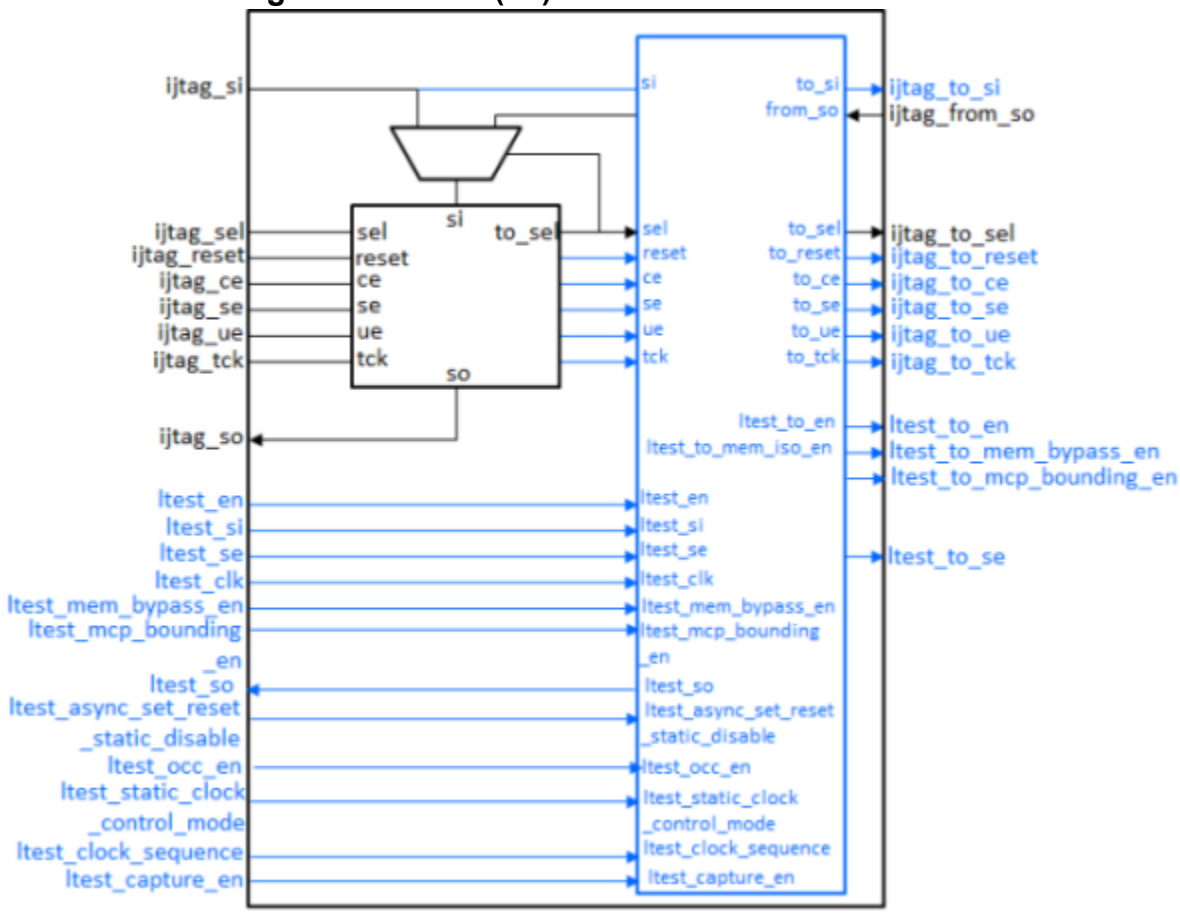
The process_dft_specification command creates a TCD for this mini-OCC and writes the TCD into the TSDB as it does with functional clock OCCs created by the DftSpecification/OCC wrappers. And with the functional clock OCCs, you must use the add_core_instances command in ATPG setup to define this mini-OCC.

Note

You must add the tck_occ_en DFT signal otherwise it is not possible to activate the mini-OCC using the add_core_instances command. If you do not activate the mini-OCC, then the mini-OCC still injects a proper shift clock during the shift cycle but keeps the clock off during the capture cycles. The values going from the STI IJTAG network nodes are known values during logic test, however the STI IJTAG nodes are not able to capture and observe their inputs, which results in numerous AU faults in the STI IJTAG network.

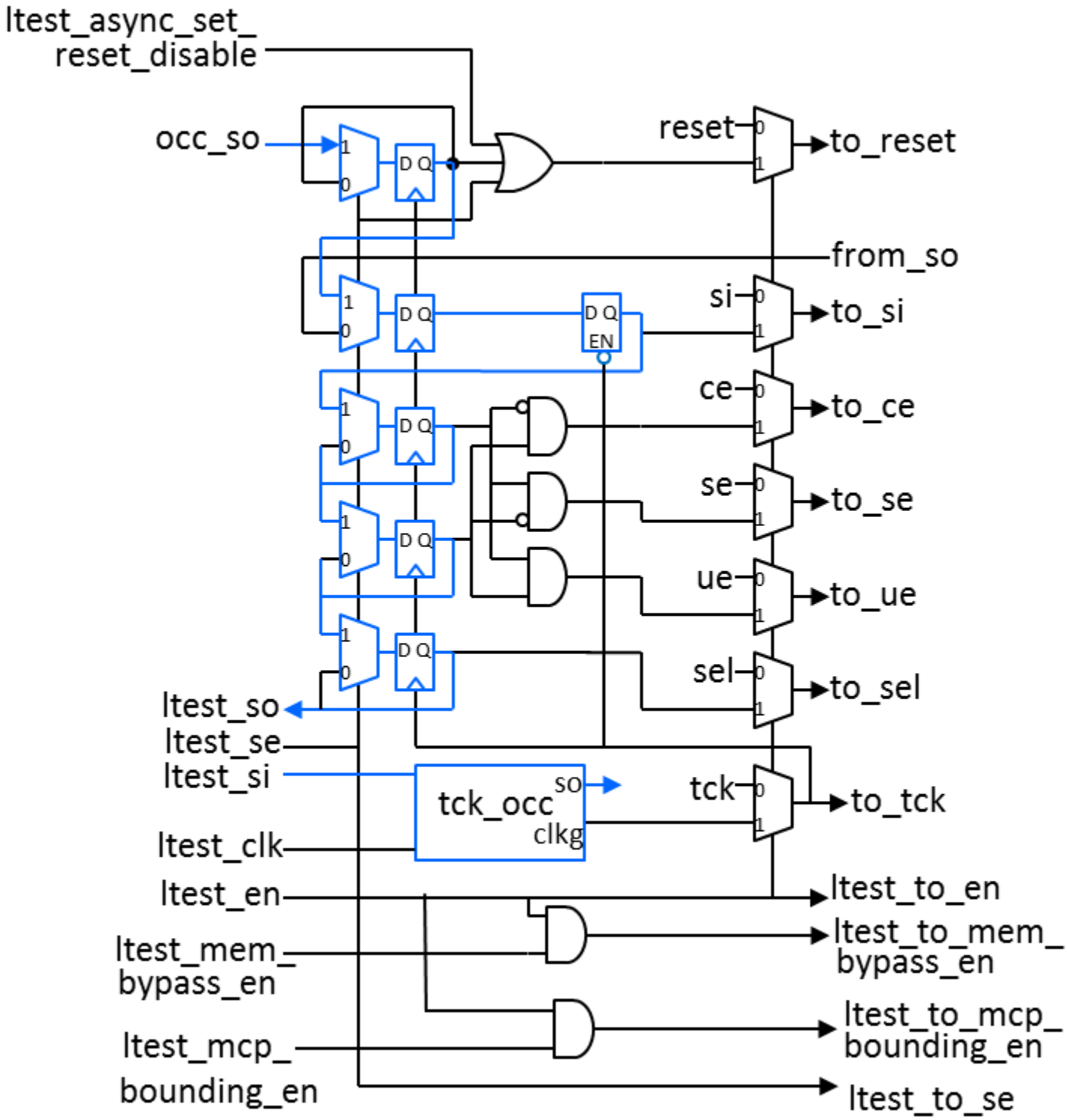
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-18. Sib(sti) With Scan Isolation Chain



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-19. Added Scan Isolation Logic for SIB(sti)



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-20. Mini-OCC for Scan Tested TCK Domain With Capture Triggered on scan_en

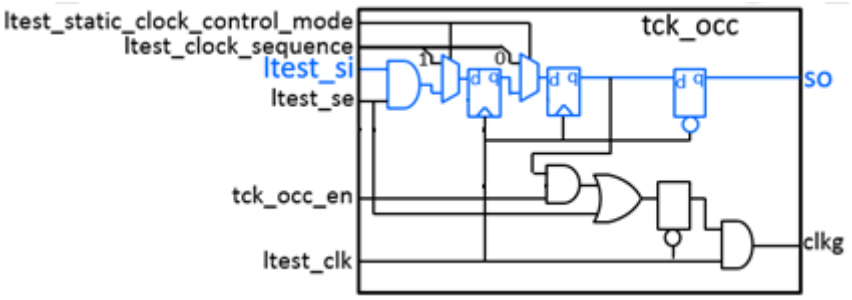
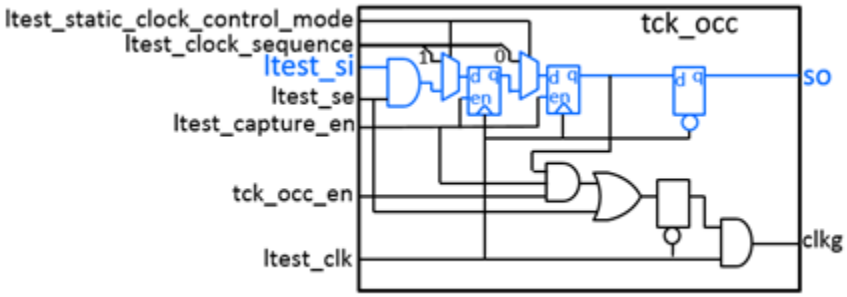


Figure 13-21. Mini-OCC for Scan Tested TCK Domain With Capture Triggered on capture_en



Arguments

- *id*

  A unique id that contains only letters, numbers, and underscores. Every Sib wrapper must have a different id value inside the IjtagNetwork wrapper.

- clock_sequence : *port_pin_constant_name*;

  A string property used for LogicBIST non-ATPG control of the mini-OCC. The default is 0. When this property is non-zero, you should provide a sequence of bit values to be connected.

- provide_full_host_scan_interface : auto | on | off ;

  A property that specifies that the current SIB provides all control, scan data, and clock signals to its clients. The default value is auto. The auto value resolves to off.

  Note

  These two settings are mutually exclusive: “provide_full_host_scan_interface on” and “to_scan_in_feedthrough none”.

- to_scan_in_feedthrough : auto | none | buffer | pipeline;

  A property that specifies if the current SIB intercepts the scan_in of its clients. The default value is auto.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Use auto for Tessent Shell to determine the required value. The auto value resolves to buffer if the provide_full_host_scan_interface property is on. Otherwise, the auto value resolves to none.
- Use none to connect the scan_in of the network hosted by the current SIB to the host scan interface.
- Use buffer to connect the scan_in of the network hosted by the current SIB to a new scan_out port on this SIB. It adds a persistent buffer in the SIB. Use this to manipulate the scan path connection for the sub-network this SIB is hosting.
- Use pipeline to connect the scan_in of the network hosted by the current SIB to a new scan_out port of this SIB. It adds a 1-bit scan register on the host-side of the SIB. Use this to relax the timing of the scan path of the network. You may require pipelining if your IJTAG network runs at a high frequency.

See the "to_scan_in_feedthrough_on_pb_sibs" property in the IjtagNetwork configuration wrapper to automate the setting of this property in some Sib wrappers during create_dft_specification. Refer to "Scan Input Pipelining" in the *Tessent IJTAG User's Manual* for additional information.

- parent_instance : *instance_name* ;

  A string property that specifies the parent instance in which to instantiate the SIB module. If unspecified, the SIB is instantiated in the root module. You can use this property to push a SIB down near the client interfaces it interacts with.

  If the instance_name you specify is an instance of a repeated module, the tool reports a warning during validation and automatically runs a uniquify_instances command as part of the insertion process to uniquify the instance_name.

  To perform IJTAG network insertion inside a module that you use more than once without uniquifying it: set the current design to that module, insert the IJTAG network in it, and refer to it with a DesignInstance wrapper at the next level up. You can do this bottom-up insertion process by going back and forth between setup and insertion mode within a single Tessent Shell invocation. Simply change the current design using the set_current_design command in setup mode.

- leaf_instance_name : *leaf_instance_name* ;

  A string property that specifies the leaf instance name of the SIB. It defaults to *design_name_dft_spec_id*_tessent_sib_*sib_id*_inst. The *dft_spec_id* string is the id you specify in the DftSpecification wrapper, and the *sib_id* is the id of the SIB. Tessent Shell checks for conflicts before instantiating the SIB and, if needed, applies the uniquification suffix specified by the set_insertion_options command.

- keep_active_during_scan_test : on | off | <u>auto</u> ;

  A property that specifies that all nodes serially connected to the SIB are to be kept active during scan test modes. For information about when to use this property, see the section "Scan Testability of the IJTAG Network" on page 3928.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- capture_value : 0 | 1 | self ;

  This property controls what the shift register captures during the capture cycle. By default, it captures a constant 0, which provides a predictable value that can always be compared when it comes out on the scan-out pin. Specifying "self" makes the SIB hold during capture. The multiplexer closest to the shift stage in Figure 13-17 is simply removed when using the "self" value. To diagnose a hold time issue in the scan path, keep the default capture value of 0.

- so_retiming : on | off ;

  A property that specifies whether the scan_out signal is to have a retiming element. The default value is on. The value off removes the scan_out retiming element. Use off only for the last node returning the data outside of a block.

  Note

  Suppose you use "so_retiming : off" for a tile client SIB and then during a second insertion pass, you add a tile host collector SIB. You must add it on the scan-out side of the tile client SIB. The result is a same-edge path between the tile client SIB, the tile host collector SIB, and its first tile host SIB. This can work out but you may have to manually add the retiming SIB to the layout if you encounter problems (meaning an engineering change order flow with your physical design team).

- Attributes/*attribute_name* : *attribute_value* ;

  A data wrapper that defines arbitrary attributes in the SIB ICL module. This mechanism is useful for finding special attributed modules in the concatenated ICL, at a later time. The attribute_name can be any identifier starting with a letter and followed by any number of letters, numbers, and underscores. The attribute_value can be any string or integer. If you want to define a personal attribute and later be able to introspect its value using get_attribute_value_list, you must have registered the attribute using the "register_attribute -object_type icl_module" command. For example, assuming you have registered the att1 attribute on the icl_module object type, and you have added the att1 attribute with a value of "v1" in some of your ICL modules using the Attributes wrapper, you can later quickly find those modules using the "get_icl_modules -filter att1==v1" command.

Examples

The following example illustrates the use of SIBs to provide discrete access to multiple parts of a complex network. There are three SIBs at the top of the network. The first SIB, which is closest to the scan-out and has an id value of "logictest_hw", is used to provide access to a TDR that is used to create the scan mode enable signals. Because the TDR id has a value of "scan_modes", the property keep_active_during_scan_test defaults to on, which in turns propagates to the three SIBs at the top of the network.

The second SIB, with an *id* value of "scannable_instruments", is used to access all instruments that are not kept active during scan test but, instead, are scan tested like the rest of the normal functional logic. This SIB can remain closed during the scan test modes such that the rest of the network remains stable and provides valid and undisturbed test_setup values.

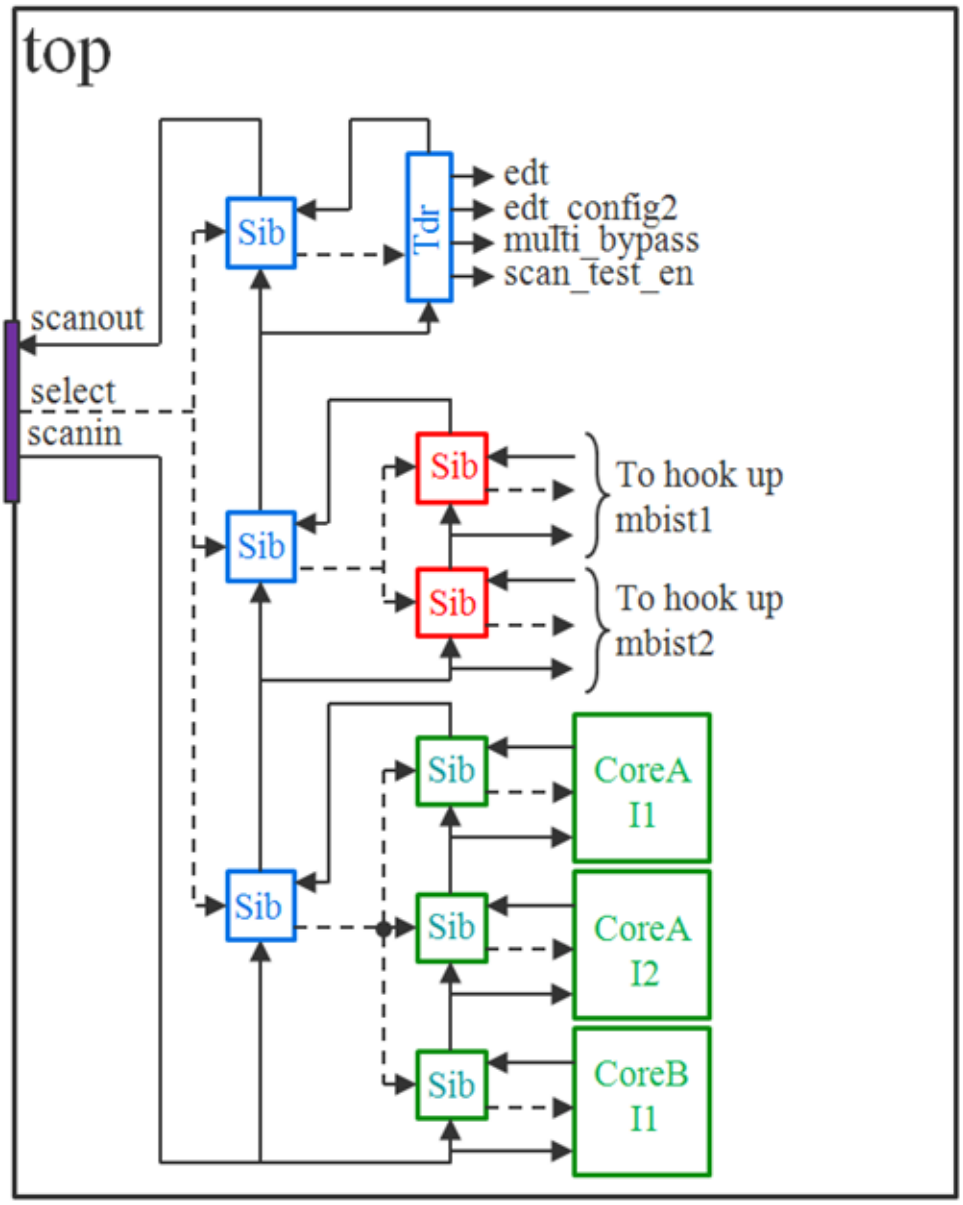
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The third SIB. with an *id* value of “child_cores”, is used to access a chain of SIBs. The chained SIBs independently open the scan path to each child core. The presence of these SIBs guarantees that a defective core or a powered down core can always be bypassed when accessing other cores.

```
DftSpecification(top,setup_hw) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Sib(logictest_hw) {
        Tdr(scan_modes) {
          DecodedSignal(edt) {
            decode_values : 2'b01, 2'b10 ;
          }
          DecodedSignal(edt_config2) {
            decode_values : 2'b10 ;
          }
          DecodedSignal(multi_bypass) {
            decode_values : 2'b11 ;
          }
          DecodedSignal(scan_test_en) {
            decode_values : 2'b01, 2'b10, 2'b11 ;
          }
        }
      }
      // Everything below this Sib is scan tested.
      // This sibs remains off when setting up logictest_hw
      sib(scannable_instruments) {
        sib(mbist1) {
        }
        sib(mbist2) {
        }
      }
      sib(child_cores) {
        sib(c1) {
          DesignInstance(corea_i1) {
          }
        }
        sib(c2) {
          DesignInstance(corea_i2) {
          }
        }
        sib(c3) {
          DesignInstance(coreb_i1) {
          }
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-22. SIB Example Schematic



Related Topics

DesignInstance

ScanMux

IjtagNetwork

Tdr

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Sib/Interface

A wrapper that contains a Client and Host wrapper that together fully specify the client and host ports.

Usage

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(id) {
      Sib(id) {
        Interface {
          Client {
            select              : port_name ; // default: ijtag_sel
            tck                 : port_name ; // default: ijtag_tck
            scan_in             : port_name ; // default: ijtag_si
            scan_out            : port_name ; // default: ijtag_so
            reset               : port_name ; // default: ijtag_reset
            reset_polarity      : active_low | active_high ;
            shift_en            : port_name ; // default: ijtag_se
            capture_en          : port_name ; // default: ijtag_ce
            update_en           : port_name ; // default: ijtag_ue
          }
          Host {
            select              : port_name ; // default: ijtag_to_sel
            tck                 : port_name ; // default: ijtag_to_tck
            scan_in             : port_name ; // default: ijtag_from_so
            scan_out            : port_name ; // default: ijtag_to_si
            reset               : port_name ; // default: ijtag_to_reset
            shift_en            : port_name ; // default: ijtag_to_se
            capture_en          : port_name ; // default: ijtag_to_ce
            update_en           : port_name ; // default: ijtag_to_ue
          }
        }
      }
    }
  }
}
```

Description

The Client wrapper provides properties to fully specify the names of the eight client ports, and also contains a property to define the reset polarity of the reset port. The Host wrapper provides properties to fully specify the names of the eight host ports, and also contains a property to define the reset polarity of the reset port.

Arguments

- Client/select : *port_name* ;

  This property defines the name of the port labelled “ijtag_sel” in Figure 13-17. The default is “ijtag_sel”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Client/tck: *port_name* ;

  This property defines the name of the port labelled “ijtag_tck” in Figure 13-17. The default is “ijtag_tck”.

- Client/scan_in : *port_name* ;

  This property defines the name of the port labelled “ijtag_si” in Figure 13-17. The default is “ijtag_si”.

- Client/scan_out : *port_name* ;

  This property defines the name of the port labelled “ijtag_so” in Figure 13-17. The default is “ijtag_so”.

- Client/reset : *port_name* ;

  This property defines the name of the port labelled “ijtag_reset” in Figure 13-17. The default is “ijtag_reset”.

- Client/reset_polarity : active_low | active_high ;

  This property defines the active polarity of the port labelled “ijtag_reset” in Figure 13-17. The default is “active_low”. Client/reset_polarity also defines the reset polarity for the Host interface of the SIB.

- Client/shift_en : *port_name* ;

  This property defines the name of the port labelled “ijtag_se” in Figure 13-17. The default is “ijtag_se”.

- Client/capture_en : *port_name* ;

  This property defines the name of the port labelled “ijtag_ce” in Figure 13-17. The default is “ijtag_ce”.

- Client/update_en : *port_name* ;

  This property defines the name of the port labelled “ijtag_ue” in Figure 13-17. The default is “ijtag_ue”.

- Host/select : *port_name* ;

  This property defines the name of the port labelled “ijtag_to_sel” in Figure 13-17. The default is “ijtag_to_sel”.

- Host/tck: *port_name* ;

  This property defines the name of the port labelled “tck” in Figure 13-18. The default is “ijtag_to_tck”.

- Host/scan_in : *port_name* ;

  This property defines the name of the port labelled “ijtag_from_so” in Figure 13-17. The default is “ijtag_from_so”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Host/scan_out: *port_name* ;

  This property defines the name of the port labelled “scan_out” in Figure 13-18. The default is “ijtag_to_si”.

- Host/reset: *port_name* ;

  This property defines the name of the port labelled “reset” in Figure 13-18. The default is “ijtag_to_reset”. The Client/reset polarity controls the Host/reset polarity.

  Note

  The host reset signal inherits polarity from the Client/reset_polarity.

- Host/shift_en: *port_name* ;

  This property defines the name of the port labelled “shift_en” in Figure 13-18. The default is “ijtag_to_se”.

- Host/capture_en: *port_name* ;

  This property defines the name of the port labelled “capture_en” in Figure 13-18. The default is “ijtag_to_ce”.

- Host/update_en: *port_name* ;

  This property defines the name of the port labelled “update_en” in Figure 13-18. The default is “ijtag_to_ue”.

Examples

The following example illustrates fully specifying a unique name for each port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(module,id) {
  IjtagNetwork {
    HostScanInterface (id) {
      Sib(id) {
        Interface {
          Client {
            select          : sel ;
            tck             : tck ;
            scan_in         : si ;
            scan_out        : so ;
            reset           : reset ;
            shift_en        : se ;
            capture_en      : ce ;
            update_en       : ue ;
          }
          Host {
            select          : to_sel ;
            tck             : to_tck ;
            scan_in         : from_so ;
            scan_out        : to_so ;
            reset           : to_rst ;
            shift_en        : to_se ;
            capture_en      : to_ce ;
            update_en       : to_ue ;
          }
        }
      }
    }
  }
}
```

Related Topics

HostScanInterface/Interface

Tdr/Interface

ScanMux/Interface

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tdr

A wrapper that specifies the creation and, optionally, the insertion of a TDR inside the IJTAG network.

Usage

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(id) {
      Tdr(id) {
        so_retiming                  : on | off ;
        DataInPorts  {
        }
        DataOutPorts {
        }
        Interface  {
        }
        parent_instance              : instance_name ;
        leaf_instance_name           : leaf_instance_name ;
        keep_active_during_scan_test : on | off | auto ;
        length                       : integer ;      // default: auto
        extra_bits_capture_value     : 0 | 1 | self ;
        reset_value                  : binary ;       // default: auto
        Attributes {
          attribute_name             : attribute_value ;
        }
        DecodedSignal(signal_name) {
        }
      }
    }
  }
}
```

Description

A wrapper that specifies the creation and, optionally, the insertion of a TDR inside the IJTAG network.

Figure 13-24 on page 3999 shows the detailed schematic of a 3-bit TDR with some data-in and data-out ports as well as some decoded output signals. When requesting data-out ports or decoded signals, the default is to always use an update stage between the shift stage and the output port. In the case of decoded signals, the decoding logic is between the shift stage and the update stage. It is possible to request that the update stage be removed in situations where it does not matter that the data bits toggle during the shift cycles. See the description of the output_timing property found in the Tdr/DataOutPorts and Tdr/DecodedSignal sections. On the other hand, if the output data bit is used to enable or disable parts of the IJTAG network, its output timing must be further delayed by an extra cycle to avoid the race condition described in "Update Stage Timing" on page 3995. The TCK clock only needs to be balanced within the TDR module but can tolerate as much as 50 percent of a TCK period between nodes because a negative edge retiming flip-flop is used on the scan-out port of the Tdr and Sib nodes.

The TDR module names are hardcoded to *design_name_design_id*_tessent_tdr_*id*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Update Stage Timing

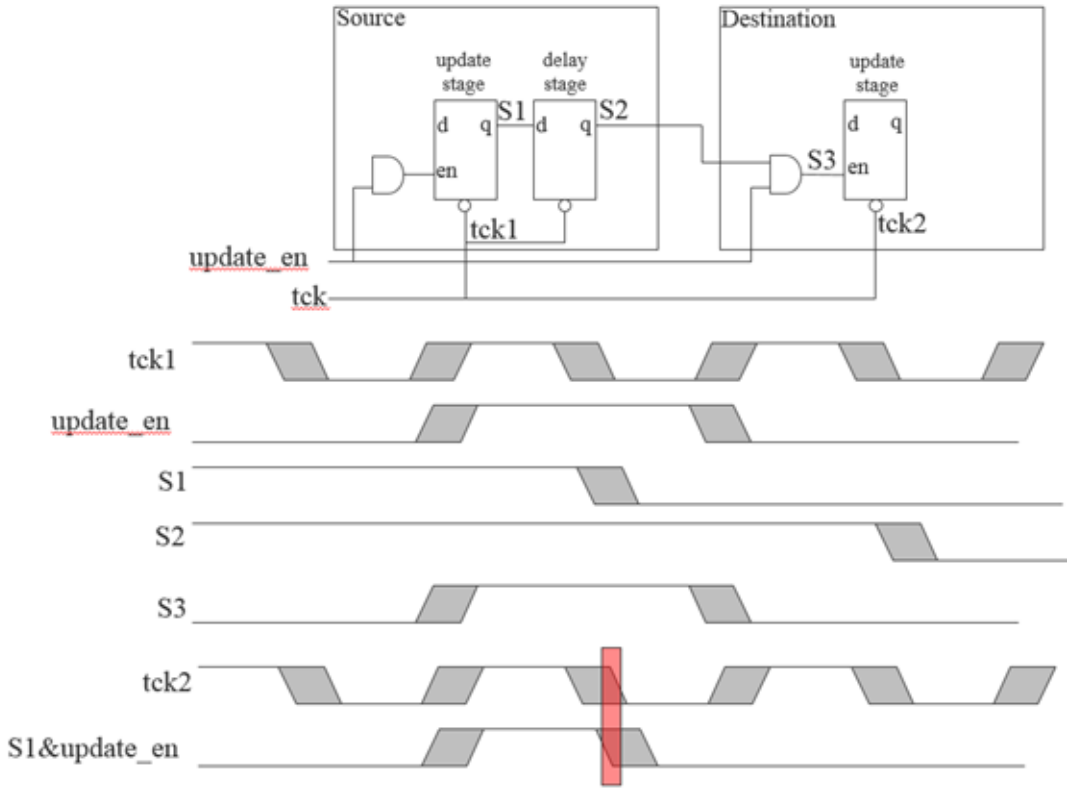
The IJTAG network can be seen as very similar to a 1149.1 or a 1500 network. It has, however, one extra level of freedom where a TDR bit can be used to enable or disable a part of the network while that part of the network is also active. In 1149.1, you never need to worry about races on the select lines because you are in the IR states when changing the instruction and it is only used the next time you go to the DR states. In 1500, the WIR and the selected TDR are accessed in a mutually exclusive manner as controlled by the selectWIR signal.

In IJTAG, you do not have this restriction. If you look at the schematic in Example 13-15 on page 3971, you can see that while you are loading the green TDR, you may also be loading into the black TDR a value that takes out the green TDR for the next scan load and, instead, selects the red TDR. As the black, green and red TDRs all receive the same update_en signal, it is important that the select signal reaching the green TDR is still active by the time the falling edge of TCK reaches the green TDR. It is also important that the select signal reaching the red TDR is still inactive by the time the falling edge of TCK reaches the red TDR.

If these conditions are not met, the green TDR may not perform its last update and the red update stage may be prematurely disturbed. This racing condition is illustrated in Figure 13-23. The source box can be seen as the black TDR while the destination box can be seen as the green TDR in Figure 13-15 on page 3971. If you have skew in the TCK clock distribution where TCK2 is delayed with respect to TCK1, the update stage value labeled S1 could change too soon and block the overlap of S1 and update_en at the destination update flip-flop. Because the enabling can be going in all directions, you either have to precisely balance your TCK clock network all over the network or do as is done in Tessent Shell and add an extra delay stage for those select signals to remove the race condition even in the presence of large TCK clock skew.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-23. Update Stage Timing Diagram



Arguments

- *id*

  A unique id that contains only letters, numbers, and underscores. Every Tdr wrapper must have a different id inside the IjtagNetwork wrapper.

- so_retiming : on | off ;

  A property that specifies whether the scan_out signal is to have a retiming element. The default value is on. The value off removes the scan_out retiming element. Use off only for the last node returning the data outside of a block.

- parent_instance : *instance_name* ;

  A string property that specifies the parent instance in which to instantiate the TDR module. When left unspecified, the TDR is instantiated in the common ancestor of the DataInPorts, DataOutPorts, and DecodedSignal connections. You can use this property to push a TDR down into a different location. You can specify "." as the parent_instance to force the TDR to be placed in the root module independent of the DataInPort, DataOutPort, and DecodedSignal connections. When the TDR has no DataInPorts and DataOutPorts connections, the parent_instance property defaults to the root module.

  If the specified or inferred instance_name is an instance of a repeated module, a warning is displayed during validation and a uniquify_instances command is automatically run as part of the insertion process to uniquify the instance_name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

To perform IjtagNetwork insertion inside a module that is used more than once without uniquifying it, you can set the current design to that module, insert the IJTAG network in it, and refer to it with a DesignInstance wrapper at the next level up. This bottom-up insertion process can be done by going back and forth between setup and insertion mode within a single Tessent Shell invocation. You simply change the current design using the set_current_design command in setup mode.

- leaf_instance_name : *leaf_instance_name* ;

  A string property that specifies the leaf instance name of the TDR. It defaults to *design_name_design_spec_id*_tessent_tdr_*tdr_id*_inst when unspecified. The *dft_spec_id* string is the id specified in the DftSpecification wrapper, and the *tdr_id* string is the id of the TDR. Tessent Shell checks for conflicts before instantiating the TDR and, if needed, applies the uniquification suffix specified by the set_insertion_options command.

- keep_active_during_scan_test : on | off | auto ;

  A property that specifies that all nodes serially connected to the TDR are to be kept active during scan test modes. For information about when to use this property, see section "Scan Testability of the IJTAG Network" on page 3928. This property defaults to on when the id is named "scan_modes".

- length : *integer* ;

  A property to request a specific shift register length. By default, the length is determined to be large enough to satisfy the specified DataInPorts/count, DataOutPorts/count or DecodedSignal/decode_values properties.

- extra_bits_capture_value : 0 | 1 | self ;

  A property that applies to the upper bits of the TDR when the length is larger than the DataInPorts/count value. The specified values control what the extra shift register bits capture during the capture cycle. The value self means that it holds its value.

- reset_value : *binary* ;

  A property that specifies a binary number as wide as the TDR to specify the reset value of the DataOutPorts and the DecodedSignals. The shift register stage is not reset. The ICL description of the TDR automatically sets the DefaultLoadValue of the ScanRegister equal to the specified reset value such that if a TDR bit resets to 1, the DefaultLoadValue for that bit is also set to 1.

- Attributes/*attribute_name* : attribute_value ;

  A data wrapper that defines arbitrary attributes in the TDR ICL module. This mechanism is useful for finding special attributed modules in the concatenated ICL, at a later time. The attribute_name can be any identifier starting with a letter and followed by any number of letters, numbers, and underscores. The attribute_value can be any string or integer. If you want to define a personal attribute and later be able to introspect its value using get_attribute_value_list, you must have registered the attribute using the "register_attribute -object_type icl_module" command. For example, assuming you have registered the att1 attribute on the icl_module object type, and you have added the att1 attribute with a value of

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

"v1" in some of your ICL modules using the Attributes wrapper, you can later quickly find those modules using the "get_icl_modules -filter att1==v1" command.

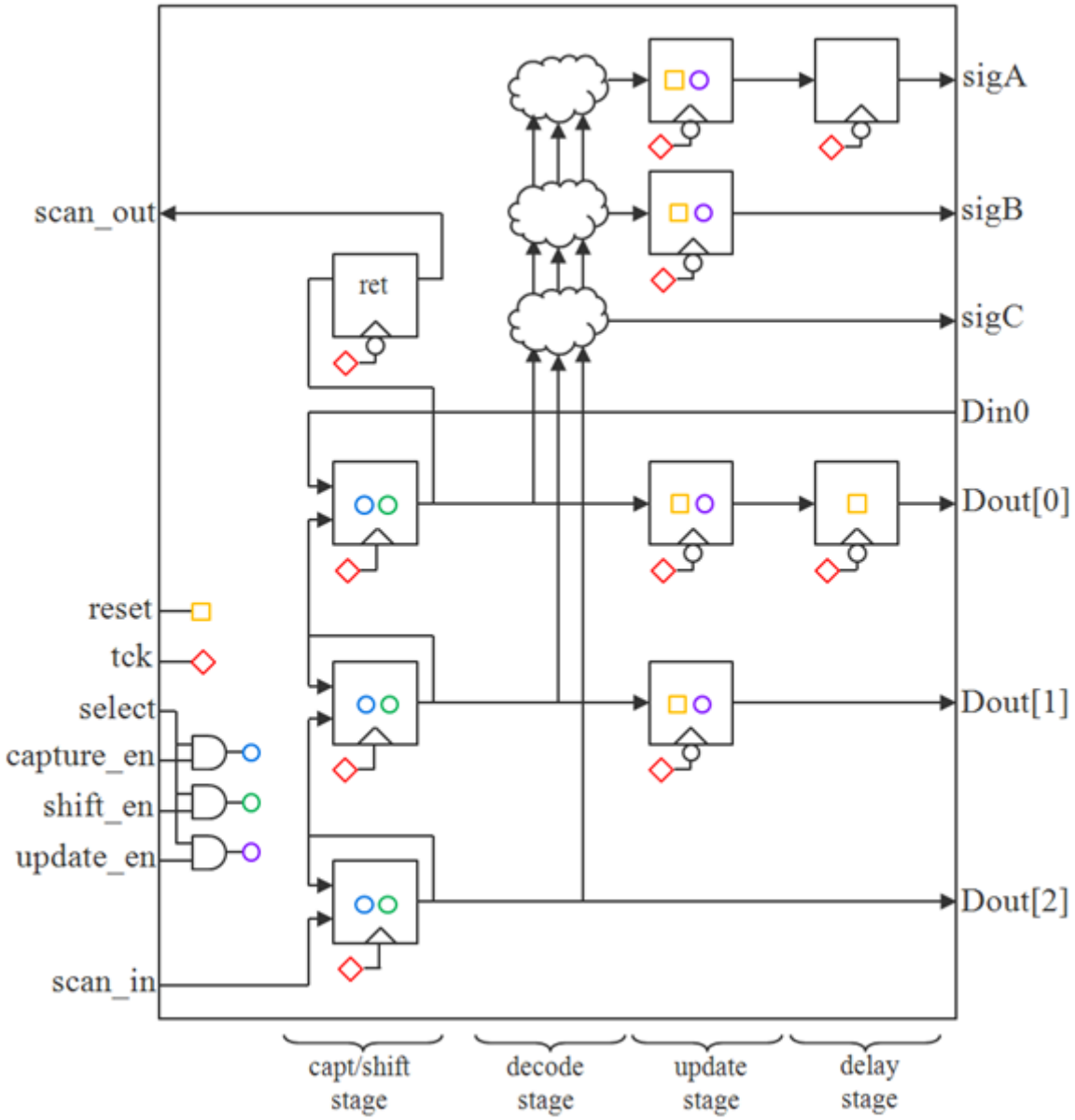
Examples

The following example illustrates the syntax used to create a TDR exactly like the one shown in Figure 13-24. Note how the output_timing property is used to make some of the outputs unlatched and some having the extra delay stage needed for ijtag_scan_selection.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Tdr(blue) {
        length                    : 3 ;
        extra_bits_capture_value : self ;
        DataOutPorts {
          count             : 3 ;
          port_naming       : Dout[%d] ;
          output_timing(0) : ijtag_scan_selection ;
          output_timing(1) : normal ;
          output_timing(2) : unlatched ;
        }
        DataInPorts {
          count             : 1 ;
          port_naming       : Din%d ;
        }
        DecodedSignal(sigA) {
          decode_values     : 3'bx00 ;
          output_timing     : ijtag_scan_selection ;
        }
        DecodedSignal(sigB) {
          decode_values     : 3'bx01 ;
          output_timing     : normal ;
        }
        DecodedSignal(sigC) {
          decode_values     : 3'bx1x ;
          output_timing     : unlatched ;
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-24. Detailed View of TDR Schematic



Related Topics

IjtagNetwork

ScanMux

DesignInstance

Sib

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tdr/Interface

A wrapper that provides properties to fully specify the names of the eight client ports of the TDR.

Usage

```
IjtagNetwork {
  HostScanInterface(id) {
    Tdr(id) {
      Interface {
        select            : port_name ;     // default: ijtag_sel
        tck               : port_name ;     // default: ijtag_tck
        scan_in           : port_name ;     // default: ijtag_si
        scan_out          : port_name ;     // default: ijtag_so
        reset             : port_name ;     // default: ijtag_reset
        reset_polarity    : active_high | active_low ;
        shift_en          : port_name ;    // default: ijtag_se
        capture_en        : port_name ;    // default: ijtag_ce
        update_en         : port_name ;    // default: ijtag_ue
      }
    }
  }
}
```

Description

The wrapper also lets you define the active polarity of the reset port.

Arguments

- select : *port_name* ;

  This property defines the name of the select port. The default is “ijtag_sel”.
- tck : *port_name* ;

  This property defines the name of the tck port. The default is “ijtag_tck”.
- scan_in : *port_name* ;

  This property defines the name of the scan_in port. The default is “ijtag_si”.
- scan_out : *port_name* ;

  This property defines the name of the scan_out port. The default is “ijtag_so”.
- reset : *port_name* ;

  This property defines the name of the reset port. The default is “ijtag_reset”.
- reset_polarity : active_high | <u>active_low</u> ;

  This property defines the active polarity of the reset port. The default is “active_low”.
- shift_en : *port_name* ;

  This property defines the name of the shift_en port. The default is “ijtag_se”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- capture_en : *port_name* ;

  This property defines the name of the capture_en port. The default is “ijtag_ce”.

- update_en : *port_name* ;

  This property defines the name of the update_en port. The default is “ijtag_ue”.

Examples

The following example illustrates fully specifying a unique name for each port of the TDR.

```
Tdr(id) {
  Interface {
    select     : sel ;
    tck        : tck ;
    scan_in    : si ;
    scan_out   : so ;
    reset      : reset ;
    shift_en   : se ;
    capture_en : ce ;
    update_en  : ue ;
  }
}
```

Related Topics

HostScanInterface/Interface

ScanMux/Interface

Sib/Interface

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tdr/DataInPorts

A wrapper that specifies the number (count) of data-in ports to create on the TDR, the naming of the ports, and the connections to make to them.

Usage

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(id) {
      Tdr(id) {
        DataInPorts  {
          count            : integer ;         // default: auto
          port_naming      : port_naming,...; // default:ijtag_data_in[%d]
          connection(range) : port_pin_constant_name; // repeatable
        }
      }
    }
  }
}
```

Description

You can connect ports, such as primary inputs, or constant “1” or “0” values to Tdr/DataInPorts. The tool creates a port if it does not exist except when the design level is “chip”.

The tool supports %d bus notation and the direct use of a bus name when they do not already exist. For example, suppose a port does not exist. In that case, the tool creates a new port named “new_bus[1:0]” by using either one of these notations:

```
connection(3:2) : new_bus[%d]; // new_bus[1:0]
connection(3:2) : new_bus      // new_bus[1:0]
```

Use %d bus notation to distinguish creation of a scalar port or a bus port. For example, the first notation below creates a new scalar port named “new_port_name” and the second notation using %d creates a single pin bus named “new_port_name[0]”:

```
connection(5) : new_port_name;     // scalar new_port_name
connection(5) : new_port_name[%d]; // single pin bus new_port_name[0]
```

Note

For limitations of the %d notation, refer to the Complex Port Types section of the Verilog and SystemVerilog Limitations table of “HDL Limitations in the Tessent Shell Flow for Verilog and SystemVerilog” on page 5083.

When the ports exist, the tool gives an error when connecting a DataOutPort to an input port. When ports do not exist, the tool gives a direction mismatch error if you specify the same port name for Tdr/DataInPorts and at least one of Tdr/DataOutPorts, Tdr/DecodedSignal, or Tap/DataOutPorts. By default, the tool creates an input port for Tdr/DataInPorts. By default, it creates an output port for Tdr/DataOutPorts, Tdr/DecodedSignal, and Tap/DataOutPorts.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- count : *integer* ;

  A property that specifies the number of data-in bits needed on the TDR. If the property is set, or defaults to auto, the count is determined to be large enough to satisfy the *range* value of all connection properties. If no connection property is specified, the count is determined to be large enough to satisfy the port_naming values if they are in a non-indexed format (%d). If the port_naming property is set, or defaults to an indexed format, the count is determined to be large enough to satisfy the length of the TDR.

- port_naming : *port_naming, ... ;*

  A property that specifies the naming of the data-in ports. The port_naming value has two distinct formats. It can be a single string using an index symbol or a list of port names with explicit sizes.

  When using the indexed format, you specify base_name[%d] when you want to use a bus and you specify base_name%<integer>d when you want to create a set of scalar ports. In both cases, the range is <count>-1 to 0. Using <integer> in the scalar naming ensures the integer is padded with 0s to be at least <integer> bits wide. For example, Din%2d creates Din00, Din01, Din02, and so on. Din%d and Din%1d mean the same thing. The default is "ijtag_data_in[%d]".

  When using a list of port names, you control the exact naming and width of each data-in port. For example, if you specify "locked", you get exactly one data-in port called "locked" as illustrated in Example 2.

- connection(*range*) : *port_pin_constant_name* ;

  A repeatable property that specifies the connections to make between the bits of the data-in ports and the existing pins inside the design. Because data-in ports are sinks, the same bit cannot be connected to multiple sources. It is, therefore, illegal to repeat the connection property with overlapping range indexes.

  You can specify *port_pin_constant_name* as either a port or pin object, or the constant "0" or "1". Specifying a constant ties the port to the indicated value and you must specify the value in a binary format, for example "1'b1" or "1'b0" for a single bit range. If the pin name refers to a bused pin and you are connecting to a full bus, you can omit the bus range if the "process_dft_specification -no_insertion" command is not used. If you are using the "process_dft_specification -no_insertion" command, you must specify the full range of the destination pins because the range cannot be extracted from the design.

Examples

Example 1

The following example illustrates the specification of two data-in ports with connections to pins on a design instance.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Sib(orange) {
        Tdr(orange) {
          DataInPorts {
            connection(0) : u2/top_green/go ;
            connection(1) : u2/top_green/done ;
          }
        }
      }
    }
  }
}
```

Example 2

The following example illustrates the explicit naming of the data-in and data-out ports. The data-in port is a single bit wide and called “locked”. There are 7 data-out ports. The port associated with connection(0) is div[0] and the one associated with connection(5) is mult[2].

```
DftSpecification(clk_gen,bottom_up) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Tdr(1) {
        DataOutPorts {
          connection(5:3) : pll/mult[2:0] ;
          connection(2:0) : pll/div[2:0] ;
          port_naming     : mux_sel, mult[2:0], div[2:0] ;
        }
        DataInPorts {
          connection(0) : pll/locked ;
          port_naming   : locked ;
        }
      }
    }
  }
}
```

Related Topics

Tdr/DataOutPorts

Tdr/DecodedSignal

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tdr/DataOutPorts

A wrapper that specifies the number (count) of data-out ports to create on the TDR, the naming of the ports, the output_timing of the ports, and the connections to make to them.

Usage

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(id) {
      Tdr(id) {
        DataOutPorts {
          count           : integer;          // default: auto
          port_naming     : port_naming,...; // default:ijtag_data_out[%d]
          multiplexing    : on | off | auto ;
          output_timing(range) : auto | unlatched | normal |
                                 ijtag_scan_selection ;
          connection(range)    : port_pin_name ; // repeatable
        }
      }
    }
  }
}
```

Description

You can connect ports, such as primary outputs, to Tdr/DataOutPorts. The tool will create a port if it does not exist except when the design level is “chip”.

The tool supports %d bus notation and the direct use of a bus name when they do not already exist. For example, suppose a port does not exist. In that case, the tool will create a new port named “new_bus[1:0]” by using either one of these notations:

```
connection(3:2) : new_bus[%d]; // new_bus[1:0]
connection(3:2) : new_bus      // new_bus[1:0]
```

Use %d bus notation to distinguish creation of a scalar port or a bus port. For example, the first notation below creates a new scalar port named “new_port_name” and the second notation using %d creates a single pin bus named “new_port_name[0]”:

```
connection(5) : new_port_name;     // scalar new_port_name
connection(5) : new_port_name[%d]; // single pin bus new_port_name[0]
```

Note

For limitations of the %d notation, refer to the Complex Port Types section of the Verilog and SystemVerilog Limitations table of “HDL Limitations in the Tessent Shell Flow for Verilog and SystemVerilog” on page 5083.

When the ports exist, the tool gives an error when connecting a DataOutPort to an input port. When ports do not exist, the tool gives a direction mismatch error if you specify the same port name for Tdr/DataInPorts and at least one of Tdr/DataOutPorts, Tdr/DecodedSignal, or Tap/

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DataOutPorts. By default, the tool creates an input port for Tdr/DataInPorts. By default, it creates an output port for Tdr/DataOutPorts, Tdr/DecodedSignal, and Tap/DataOutPorts.

Arguments

- count : *integer* ;

  A property that specifies the number of data-out bits needed on the TDR. If the property is set, or defaults to auto, the count is determined to be large enough to satisfy the *range* value of all connection properties. If no connection property is specified, the count is determined to be large enough to satisfy the port_naming values if they are in a non-indexed format (%d). If the port_naming property is set, or defaults to an indexed format, the count is determined to be large enough to satisfy the length of the TDR.

- port_naming : *port_naming, ... ;*

  A property that specifies the naming of the data-out ports. The port_naming value has two distinct formats. It can be a single string using an index symbol or a list of port names with explicit sizes.

  When using the indexed format, you specify base_name[%d] when you want to use a bus and you specify base_name%<integer>d when you want to create a set of scalar ports. In both cases, the range is <count>-1 to 0. Using <integer> in the scalar naming ensures the integer is padded with 0s to be at least <integer> bits wide. For example, Dout%2d creates Dout00, Dout01, Dout02, and so on. Dout%d and Dout%1d mean the same thing. The default is “ijtag_data_out[%d]”.

  When using a list of port names, you control the exact naming and width of each data-out port. For example, if you specify “mux_sel,mult[2:0],div[2:0]”, you get exactly seven data-out ports where the most significant bit is called mux_sel and the least significant bit is called div[0] as illustrated in DataOutPort Connection Multiplexing.

- multiplexing : on | off | <u>auto</u> ;

  A property that specifies whether to add a multiplexer when making the connection to the specified pin_name of the connection properties. When set to auto, a multiplexer is added unless the pin is unconnected, connected to a constant, or connected to a net with no fanin. As soon as one connection needs multiplexing, an extra data-out port is added to control the select of the multiplexers. The multiplexers are inserted in the direct parent instance of the specified pin_name of the connection properties. See DataOutPort Connection Multiplexing for illustration.

- output_timing(*range)* : <u>auto</u> | unlatched | normal | ijtag_scan_selection ;

  A repeatable property that specifies the output timing of the data-out ports. The default value is auto. The auto becomes ijtag_scan_selection if the DataOutPort is used as a ScanMux select source; otherwise, it becomes normal.

  As you can see in Figure 13-24:

  - unlatched — A data-out port can be directly connected to the shift register when output timing is specified as unlatched. This corresponds to Dout[2] in Figure 13-24.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- normal — A data-out port can be connected to the shift register through an update stage when output timing is specified as normal. This corresponds to Dout[1] in Figure 13-24.
- ijtag_scan_selection — A data-out port can be connected to the shift register through an update stage and another delay stage when output timing is specified as ijtag_scan_selection. This corresponds to Dout[0] in Figure 13-24. For a description of when the delay stage is needed, refer to "Update Stage Timing" on page 3995.

- connection(*range*) : *port_pin_name* ;

  A repeatable property that specifies connections to make between bits of the data-out ports and existing pins inside the design. Because data-out ports are sources, the same bit can be connected to multiple destinations by simply repeating the connection property with the same range indexes.

  If the pin name refers to a bused pin and you are connecting to a full bus, you can omit the bus range if the process_dft_specification -no_insertion option is not used. If you are using the process_dft_specification -no_insertion switch, you must specify the full range of the destination pin because the range cannot be extracted from the design.

Examples

Example 1

The following example illustrates the specification of four data-out ports with connections to pins on a design instance. The example associated with Figure 13-24 shows the use of the output_timing property.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Sib(orange) {
        Tdr(orange) {
          DataOutPorts {
            connection(0)   : u2/top_green/en ;
            connection(3:1) : u2/top_green/mode[2:0] ;
          }
        }
      }
    }
  }
}
```

Example 2

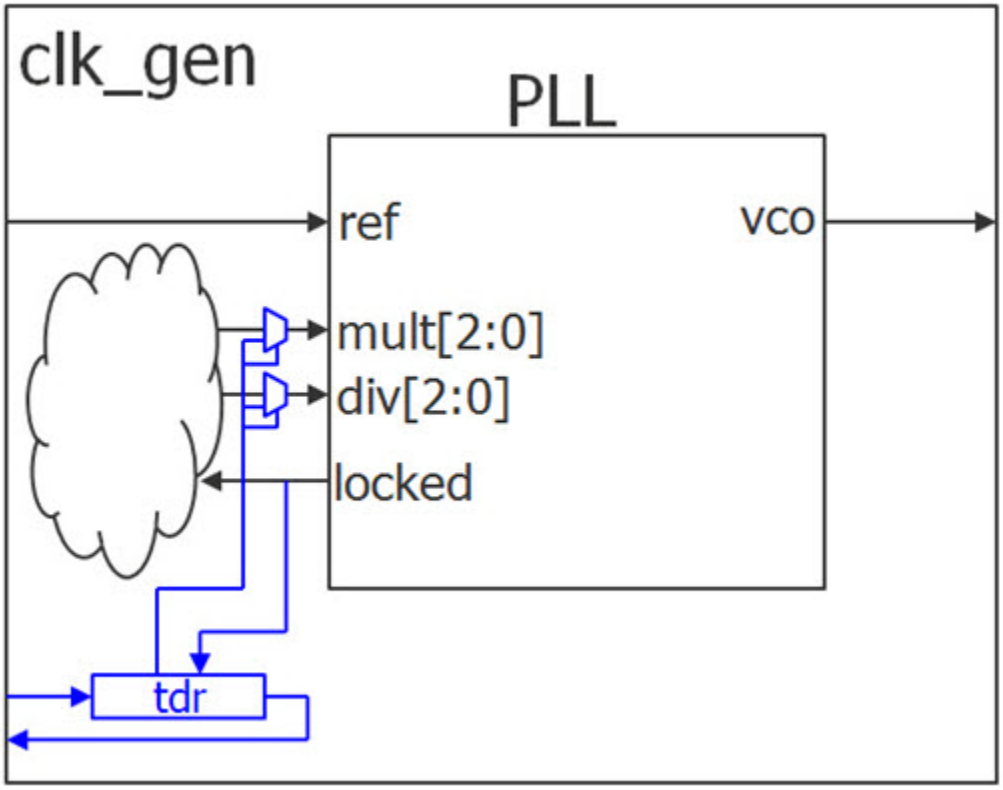
The following example illustrates the explicit naming of the data-in and data-out ports. The data-in port is a single bit wide and called "locked". There are 7 data-out ports. The port associated with connection(0) is div[0] and the one associated with connection(5) is mult[2].

Multiplexing is left unspecified so it defaults to auto. Because the pll/mult[2:0] and pll/div[2:0] pins have a functional source, a multiplexer is inserted and a 7th data-out bit named mux_sel is used to control the multiplexers.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(clk_gen,bottom_up) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Tdr(1) {
        DataOutPorts {
          connection(5:3) : pll/mult[2:0] ;
          connection(2:0) : pll/div[2:0] ;
          port_naming     : mux_sel, mult[2:0], div[2:0] ;
        }
        DataInPorts {
          connection(0) : pll/locked ;
          port_naming   : locked ;
        }
      }
    }
  }
}
```

Figure 13-25. DataOutPort Connection Multiplexing



Related Topics

Tdr/DataInPorts

Tdr/DecodedSignal

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tdr/DecodedSignal

A wrapper that specifies a decoded signal to create.

Usage

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(id) {
      Tdr(id) {
        DecodedSignal(signal_name) {
          decode_values    : binary_code, ... ;
          multiplexing     : on | off | auto ;
          output_timing(range) : auto | unlatched | normal |
                                 ijtag_scan_selection ;
          connection       : port_pin_name ; // repeatable
          Attributes {
            attribute_name : attribute_value ;
          }
        }
      }
    }
  }
}
```

Description

Tdr/DecodedSignal can connect to ports, such as primary inputs and outputs. The tool will create a port if it does not exist except when the design level is “chip”. By default, it will create an output port for Tdr/DecodedSignal.

The decode_values property specifies which values of the TDR make the decoded signal active. The connection property specifies connections from the decoded signals.

Arguments

- decode_values : *binary_code*, ... ;

  A property that specifies a list of binary codes that define when the decoded signal is active. The binary codes must be as wide as the TDR. The binary numbers may contain Xs as long as there is at least one 0 or one 1 in each value.

- multiplexing : on | off | auto ;

  A property that specifies whether to add a multiplexer when making the connection to the specified pin_name of the connection properties. When set to auto, a multiplexer is added unless the pin is unconnected, connected to a constant, or connected to a net with no fanin. As soon as one connection needs multiplexing, an extra data-out port is added on the TDR to control the select of the multiplexers. The multiplexers are inserted in the direct parent instance of the specified pin_name of the connection properties. See DataOutPort Connection Multiplexing of the Tdr/DataOutPorts section for an illustration of this.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- output_timing(*range)* : auto | unlatched | normal | *ijtag_scan_selection* ;

  A property that specifies the output timing of the decoded signal port. The default value is auto. The auto becomes *ijtag_scan_selection* if the DecodedSignal is used as a ScanMux select source; otherwise, it becomes normal.

  As you can see in Figure 13-24:

  - unlatched — A decoded signal is directly connected to the decoding logic when output timing is specified as unlatched; this corresponds to SigC in Figure 13-24.
  - normal — A data-out port is connected to the decoding logic through an update stage when output timing is specified as normal; this corresponds to SigB in Figure 13-24.
  - *ijtag_scan_selection* — A data-out port is connected to the decoding logic through an update stage and another delay stage when output timing is specified as *ijtag_scan_selection*; this corresponds to SigA in Figure 13-24.

  Refer to the "Update Stage Timing" on page 3995 for a description of when the delay stage is needed.

- connection : *port_pin_name* ;

  A repeatable property that specifies connections to make between the decoded signal and existing pins inside the design. Because decoded signals are sources, they can be connected to multiple destinations by simply repeating the connection property.

- Attributes/*attribute_name* : *attribute_value* ;

  A data wrapper that defines arbitrary attributes on the ICL port associated with the decoded signal. This mechanism is useful for finding special attributed ports in the concatenated ICL. The *attribute_name* can be any identifier starting with a letter and followed by any number of letters, numbers and underscores. The *attribute_value* can be any string or integer. If you want to define a personal attribute and later be able to introspect its value using get_attribute_value_list, you must have registered the attribute using the "register_attribute -object_type icl_module" command. For example, assuming you have registered the att1 attribute on the icl_port object type, and you have added the att1 attribute with a value of "v1" to some DecodedSignal ports of your TDR modules using the above Attributes wrapper, you can later quickly find those ports using the "get_icl_ports -of module_name -filter att1==v1" command. The icl_port attributes are also inherited by the icl_pin objects so you can also find the pins using "get_icl_pins -filter att1==v1".

Examples

The following example illustrates the creation of three decoded signals with each having a different output timing. The example corresponds to the schematic shown in Figure 13-24.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Tdr(blue) {
        DecodedSignal(sigA) {
          decode_values    : 3'bx00 ;
          output_timing    : ijtag_scan_selection ;
        }
        DecodedSignal(sigB) {
          decode_values    : 3'bx01 ;
          output_timing    : normal ;
        }
        DecodedSignal(sigC) {
          decode_values    : 3'bx1x ;
          output_timing    : unlatched ;
        }
      }
    }
  }
}
```

Related Topics

Tdr/DataInPorts

Tdr/DataOutPorts

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tap

A wrapper that specifies the creation and, optionally, the insertion of a TAP controller into the IjtagNetwork.

Usage

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        Interface {
        }
        parent_instance                    : instance_name ;
        leaf_instance_name                 : leaf_instance_name ;
        keep_active_during_scan_test       : on | off | auto ;
        bypass_instruction_codes           : auto | binary , ... ;
        add_buffers_on_jtag_signal_sources : on | off | auto ;
        add_negedge_flop_on_tdo_output     : on | off | auto ;
        Attributes {
          attribute_name : attribute_value;
        }
        DataOutPorts {
        }
        HostBscan  {
        }
        DeviceIDRegister  {
        }
        HostIjtag(id)  {    // repeatable
        }
      }
    }
  }
}
```

Description

A wrapper that specifies the creation and, optionally, the insertion of a TAP controller into the IjtagNetwork.

A TAP is a special node in IJTAG. It contains an IEEE1149.1 state machine controlled by a TMS port. The TMS port is used to steer the state machine into various states and control when the capture, shift, and update cycles happen. Unlike a SIB and a TDR, which are only active in the DR States, the TAP has an instruction register that is activated in the IR States. The value in the instruction register determines which data register is selected when in the DR States. The TAP state machine is illustrated in Figure 13-26. A block diagram of a TAP controller is shown in Figure 13-27.

A TAP wrapper is only allowed to exist in a HostScanInterface or inside the Input wrapper of a ScanMux. A TAP is only allowed to be in series with other TAP controllers and can never be in series with Tdr and Sib nodes even through ScanMux nodes. The name of all ports on the TAP controller are user-specifiable using the Interface wrappers. BoundaryScan host ports are present when the Tap/HostBScan wrapper is specified. The device id register is present when

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

the Tap/DeviceIDRegister wrapper is specified. Finally, the IJTAG host ports are present when one or more HostIjtag wrappers are specified. The bypass register is required and always present.

If you are familiar with the LogicVision TAP, you remember that it had an internal DR register used to observe status bits and control user bits and decoded signals. If you need such a register, you can and should use a Tdr wrapper inside a HostIjtag wrapper of the Tap; this gives you access to the Tdr/DataInPorts, Tdr/DataOutPorts, and Tdr/DecodedSignal wrapper syntax.

All registers operate on the rising edge of TCK except for the update stage, labelled "inst_latch" in Figure 13-27, which operates on the falling edge of TCK.

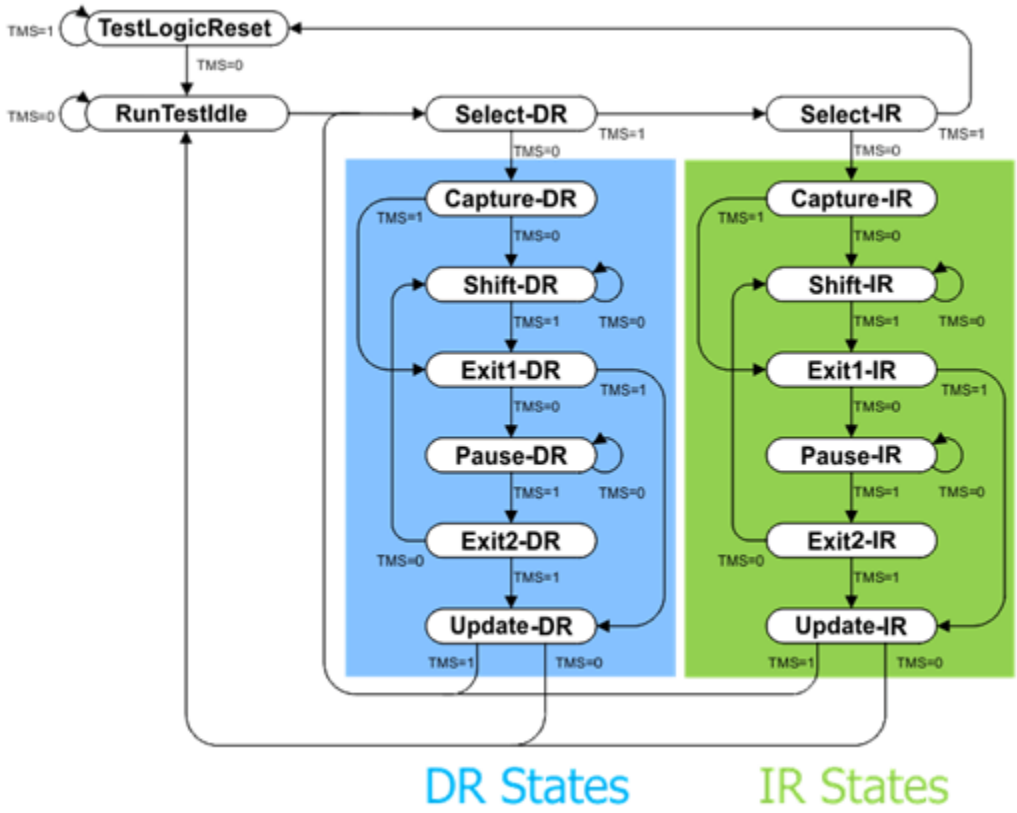
The retiming latch shown in Figure 13-27 is replaced by a retiming flip-flop when the scan_path_retiming property within the IjtagNetwork/ImplementationOptions section is set to "flop". For the reason explained in the description of that property, the retiming flop is moved as compared to where the latch is shown so as not to be in the path between the from_hostX_so ports and the tdo port in order to avoid the cascaded retiming flip-flops situation described there. As stated in that section, make sure third party nodes include a retiming element on their scan_out when you connect them directly to the scan_in port of the HostIjtag scan interfaces Tessent Tap and you specify the scan_path_retiming property to "flop", otherwise your elements are not 1149.1 compliant and are not usable within a board environment.

The TAP controller incorporates a persistent gate (tessent_persistent_cell_tck_cts_stop_buf) at the root of its TCK clock tree to help with network performance. Refer to "IJTAG Network Performance Optimization" in the *Tessent IJTAG User's Manual* for further information.

The tap module names are hardcoded to *design_name_design_id*_tessent_tap_*id*.

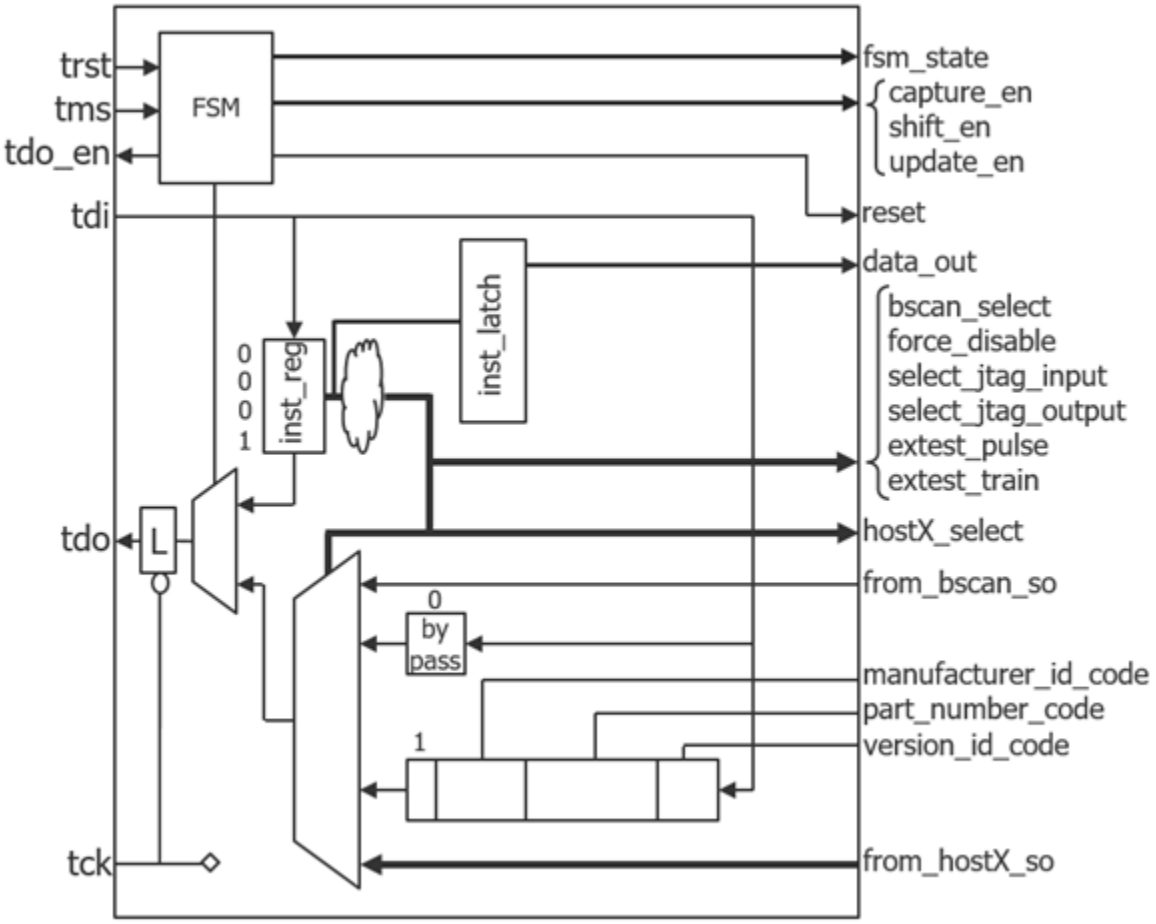
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-26. TAP FSM Diagram



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-27. TAP Block Diagram



Arguments

- *id*

  A unique id that contains only letters, numbers, and underscores. Every Tap wrapper must have a different id value inside the IjtagNetwork wrapper.

- parent_instance : *instance_name* ;

  A string property that specifies the parent instance in which to instantiate the TAP module. If unspecified, the TAP is instantiated in the root module. You can use this property to push a TAP down near the client interfaces it interacts with.

  If the specified instance_name is an instance of a repeated module, the tool issues a warning during validation and automatically issues a uniquify_instances command as part of the insertion process to uniquify the instance_name.

  To perform IJTAG network insertion inside a module that is used more than once without uniquifying it, you can set the current design to that module, insert the IJTAG network in it, and refer to it with a DesignInstance wrapper at the next level up. This bottom-up insertion process can be done by going back and forth between setup and insertion mode within a

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

single Tessent Shell invocation. You simply change the current design using the set_current_design command in setup mode.

- leaf_instance_name : *leaf_instance_name* ;

  A string property that specifies the leaf instance name of the TAP. It defaults to *design_name_dft_spec_id*_tessent_tap_*tap_id*_inst when unspecified. The *dft_spec_id* string is the id specified in the DftSpecification wrapper, and the *tap_id* is the *id* of the Tap wrapper. Tessent Shell checks for conflicts before instantiating the TAP controller and, if needed, applies the uniquification suffix specified by the set_insertion_options command.

- keep_active_during_scan_test : on | off | auto ;

  A property that specifies that all nodes serially connected to the TAP are to be kept active during scan test modes. For information about when to use this property, see the section "Scan Testability of the IJTAG Network" on page 3928.

- bypass_instruction_codes : auto | *binary*, ... ;

  A property used to define one or many opcodes for the BYPASS instruction. The all 1 code is automatically assigned to the BYPASS instruction whether specified or not. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file.

- add_buffers_on_jtag_signal_sources : on | off | auto ;

  A property that adds persistent buffers to the TAP controller's input control signals: TDI, TMS, TCK and TRST. The buffers are added to create an interception point for the TAP and the nodes in the networks it hosts. When set to "auto", the buffers are added if there is at least one InSystemTest controller wrapper in the current DftSpecification. Use this property if you intend to insert InSystemTest controllers to intercept a specific TAP controller.

- add_negedge_flop_on_tdo_output : on | off | auto ;

  A property that when specified to "on" replaces the scan path retiming element on the TDO output of the TAP controller with a negative edge flip-flop. The use of this property supersedes the IjtagNetwork/ImplementationOptions/scan_path_retiming DftSpecification setting for the current design. The default property setting of "auto" resolves to "off".

  You use this property to ease external loop timing by referencing the TDO output from a single clock edge rather than from the various IJTAG or boundary scan elements feeding the TAP. In addition, you can use the "open_pattern_set -scan_out_probe_point" property to provide a full TCK period for TDO sampling, rather than a half period, to further ease the TDO timing path.

- Attributes/*attribute_name* : *attribute_value* ;

  A data wrapper that defines arbitrary attributes in the TAP ICL module. This mechanism is useful for finding special attributed modules in the concatenated ICL at a later time. The *attribute_name* can be any identifier starting with a letter and followed by any number of letters, numbers, and underscores. The *attribute_value* can be any string or integer. If you want to define a personal attribute and later be able to introspect its value using get_attribute_value_list, you must have registered the attribute using the "register_attribute

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

-object_type icl_module" command. For example, assuming you have registered the att1 attribute on the icl_module object type, and you have added the att1 attribute with a value of "v1" in some of your ICL modules using the Attributes wrapper, you can later quickly find those modules using the "get_icl_modules -filter att1==v1" command.

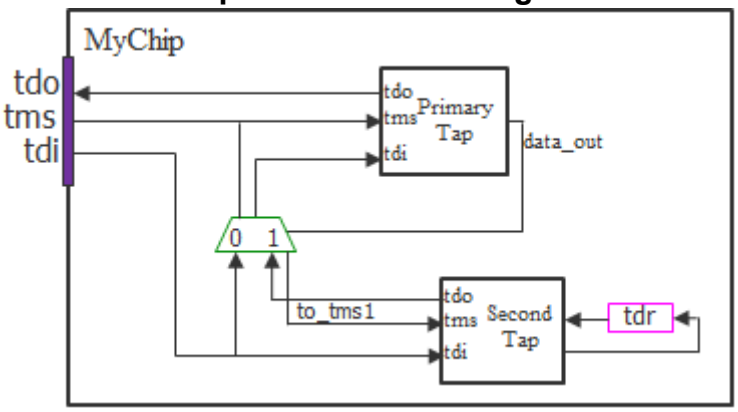
Examples

Example 1

This example defines two TAP controllers where a DataOutPort on the primary TAP is used to control if the secondary TAP is active or not. When the DataOutPort on the primary TAP is high, the secondary TAP is in series with it. When the DataOutPort on the primary TAP is low, the secondary TAP is bypassed and it is held in run_test_idle by gating its tms port low.

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Tap(main) {
        DataOutPorts {
          count : 1 ;
        }
      }
      ScanMux(green) {
        select : Tap(main)/DataOut(0) ;
        Input(1) {
          Tap(secondary) {
            HostIjtag(0) {
              Tdr(pink) {
              }
            }
          }
        }
      }
    }
  }
}
```

Figure 13-28. Example Multi-TAP Configuration Schematic



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

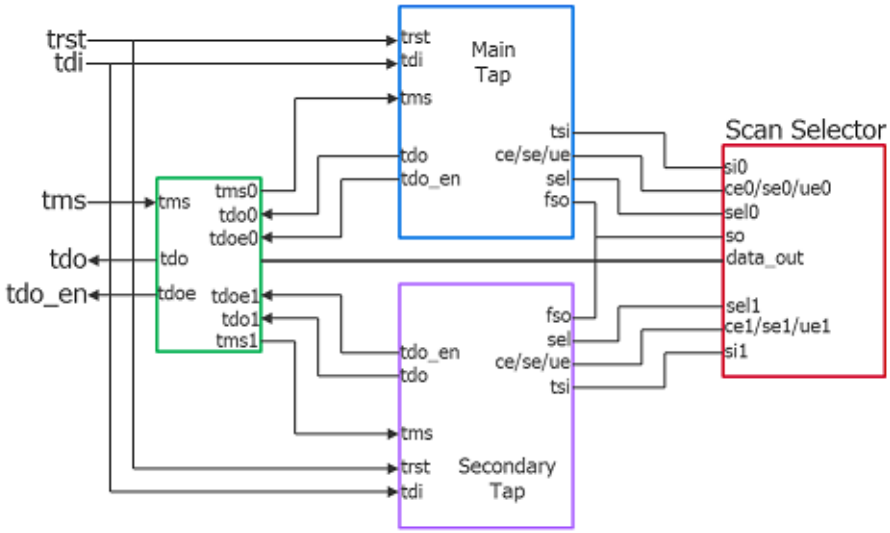
Example 2

The following example uses a special one bit TDR to select between two TAPs. The special aspect of this circuit is that the one bit TDR (called Scan Selector in Figure 13-29) used to select which TAP is active remains active below the selected TAP. Both TAPs have a HostIjtag connected to one ScanInterface of the Scan Selector and the state of the Scan Selector itself decides which ScanInterface controls it. The advantage of this circuit is that the currently selected TAP remains alone between TDI and TDO. Using the instruction on the Main TAP that accesses the TDR enables changing the value in the TDR to give control to the Secondary TAP. The same method in the newly selected Secondary TAP can be used to return control to the Main TAP. The IJTAG solver can automatically switch between TAPs based on the elements it is being asked to access.

Figure 13-29 shows the block diagram of the circuit. Below it is the DftSpecification used to insert two TAP controllers and a ScanMux and connect the HostIjtag of each TAP to the two ScanInterfaces of the Scan Selector. The Verilog description and ICL of the Scan Selector are shown next.

The TDR in the Scan Selector module resets to 0 when the trst port is pulled low or the 5 tms sequence is applied for the active TAP. The process_dft_specification.post_insertion callback shown below is used to insert the AND gate during the process_dft_specification execution. This AND gate is not described in the ICL but it does not matter as it is the implied behavior.

Figure 13-29. Example of TDR Scan Selector



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DftSpecification:

```
DftSpecification(chip,rtl) {
  IjtagNetwork {
    HostScanInterface(tap) {
      Interface {
        tck : tck ;
        tdi : tdi ;
        tdo : tdo ;
        trst : trst ;
        tms : tms ;
      }
      ScanMux(tap_mux) {
        select : DesignInstance(scan_selector)/data_out ;
        Input(0) {
          Tap(main) {
            HostIjtag(0) {
              DesignInstance(scan_selector) {
                scan_interface   : Int0 ;
              }
            }
            HostIjtag(1) {
              tdr(1) {
                DataOutPorts {
                  count : 8 ;
                }
              }
            }
          }
        }
        Input(1) {
          Tap(secondary) {
            HostIjtag(0) {
              DesignInstance(scan_selector) {
                scan_interface   : Int1 ;
              }
            }
            HostIjtag(1) {
              tdr(2) {
                DataOutPorts {
                  count : 8 ;
                }
              }
            }
          }
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The process_dft_specification.post_insertion callback to AND the two rstn pins of the TAP:

```
proc process_dft_specification.post_insertion {root_wrapper args} { \
  create_connections chip_rtl_tessent_tap_main_inst/test_logic_reset/\
  scan_selector/rstn
  intercept_connection scan_selector/rstn -cell_function_name and \
  -input2 chip_rtl_tessent_tap_secondary_inst/test_logic_reset
}
```

Verilog view of the special TDR called scan_selector:

```
module scan_selector (tck, rstn1, sel1, si1, se1, ue1, so1,
                      rstn0, sel0, si0, se0, ue0, so0, data_out);
input tck, rstn1, sel1, si1, se1, ue1, rstn0, sel0, si0, se0, ue0;
output so1, so0, data_out;
reg shift_stage, update_stage;
assign se  = (data_out) ? se1  : se0;
assign ue  = (data_out) ? ue1  : ue0;
assign sel = (data_out) ? sel1 : sel0;
assign si  = (data_out) ? si1  : si0;
assign rstn = (data_out) ? rstn1 : rstn0;
assign data_out = update_stage;
assign so1 = shift_stage;
assign so0 = shift_stage;
assign se_gated = sel & se;
assign ue_gated = sel & ue;
always @ (posedge tck or negedge rstn) begin
  if (~rstn) begin
    update_stage <= 1'b0;
  end else begin
    if (ue_gated) begin
      update_stage <= shift_stage;
    end
  end
endalways @ (posedge tck) begin
  if (se_gated) begin
    shift_stage <= si;
  end
end
endmodule
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL view of the special TDR called scan_selector:

```
Module scan_selector {
  TCKPort      tck;
  SelectPort   sel1;
  ScanInPort   si1;
  ShiftEnPort  se1;
  UpdateEnPort ue1;
  SelectPort   sel0;
  ScanInPort   si0;
  ShiftEnPort  se0;
  UpdateEnPort ue0;
  ScanOutPort  so1 { Source tdr; }
  ScanOutPort  so0 { Source tdr; }
  DataOutPort data_out { Source tdr; }
  ScanInterface Int1 { Port tck; Port sel1; Port si1;
                       Port se1; Port ue1; Port so1;}
  ScanInterface Int0 { Port tck;  Port sel0; Port si0;
                       Port se0; Port ue0; Port so0;}
  ScanMux M1 SelectedBy tdr {
    1'b0 : si0;
    1'b1 : si1;
  }
  ScanRegister tdr {
    ScanInSource M1;
    ResetValue 1'b0;
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tap/Interface

A wrapper that contains a Client and Host wrapper that together fully specify the names and polarity of the client and host ports.

Usage

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        Interface {
          fsm_state          : port_name ; // default: fsm_state[%d]
          Client {
            tdi              : port_name ; // default: tdi
            tdo              : port_name ; // default: tdo
            tck              : port_name ; // default: tck
            tms              : port_name ; // default: tms
            trst             : port_name ; // default: trst
            tdo_en           : port_name ; // default: tdo_en
            tdo_en_polarity  : active_high | active_low | auto ;
          }
          Host {
            reset            : port_name ; // default: test_logic_reset
            capture_en       : port_name ; // default: capture_dr_en
            shift_en         : port_name ; // default: shift_dr_en
            update_en        : port_name ; // default: update_dr_en
            reset_polarity   : active_high | active_low ;
          }
          TdiMux {
            parent_instance     : instance_name;
            leaf_instance_name  : instance_name;
            tdi_local           : port_name; // def: tdi_local
            bypass_devid_select : port_name; // def: bypass_devid_select
            ir_select           : port_name; // def: ir_select
            tdi_selected        : port_name; // def: tdi_selected
          }
        }
      }
    }
  }
}
```

Description

A wrapper that contains a Client and Host wrapper that together fully specify the names and polarity of the client and host ports. It also includes the TdiMux wrapper, which is only used when you specify the tdi_local property within the HostScanInterface/Interface wrapper.

The Client wrapper provides properties to fully specify the names of the six client ports, and also contains a property to define the polarity of the tdo_en port. The Host wrapper provides properties to fully specify the names of the four global host ports as well as a property to define the polarity of the reset port. The rest of the host ports related to the IJTAG and boundary scan ports are specified in the Tap/HostIjtag and Tap/HostBScan wrappers respectively.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- fsm_state : *port_name* ;

  This property defines the name of the port labeled “fsm_state” in Figure 13-27. The specified value must end with “[%d]”. The default is “fsm_state[%d]”.

- Client/tdi: *port_name*;

  This property defines the name of the port labeled “tdi” in Figure 13-27. The default name when unspecified is “tdi”.

- Client/tdo : *port_name*;

  This property defines the name of the port labeled “tdo” in Figure 13-27. The default name when unspecified is “tdo”.

- Client/tck : *port_name*;

  This property defines the name of the port labeled “tck” in Figure 13-27. The default name when unspecified is “tck”.

- Client/tms : *port_name*;

  This property defines the name of the port labeled “tms” in Figure 13-27. The default name when unspecified is “tms”.

- Client/trst : *port_name*;

  This property defines the name of the port labeled “trst” in Figure 13-27. The default name when unspecified is “trst”. This port is always active low.

- Client/tdo_en : *port_name*;

  This property defines the name of the port labeled “tdo_en” in Figure 13-27. The default name when unspecified is “tdo_en”. Its active polarity is controlled by the tdo_en_polarity property. The port is active in the Shift-DR and Shift-IR states. See Figure 13-26 for a diagram of the state machines.

- Client/tdo_en_polarity : active_high | active_low | <u>auto</u> ;

  This property controls the polarity of the tdo_en port. The default is auto. When the DftSpecification is created by the create_dft_specification command, the property is automatically specified to match the polarity of the enable pin for the output pad buffer associated with the tdo port.

- Host/reset : *port_name* ;

  This property defines the name of the port labelled “reset” in Figure 13-27. The default name when unspecified is “test_logic_reset”. Its active polarity is defined using the reset_polarity property. This port is active when the TAP fsm is in the TestLogicReset state which can be synchronously entered by keeping the tms port high for a minimum of 5 TCK cycles or asynchronously entered by setting the trst port to zero.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Host/capture_en : *port_name* ;

  This property defines the name of the port labeled “capture_en” in Figure 13-27. The default name when unspecified is “capture_dr_en”.

- Host/shift_en : *port_name* ;

  This property defines the name of the port labelled “shift_en” in Figure 13-27. The default name when unspecified is “shift_dr_en”.

- Host/update_en : *port_name* ;

  This property defines the name of the port labelled “update_en” in Figure 13-27. The default name when unspecified is “update_dr_en”.

- Host/reset_polarity : active_high | active_low ;

  This property defines the active polarity of the reset port.

- TdiMux/parent_instance : *instance_name* ;

  A property that specifies the parent instance in which to instantiate the TDI local mux module. By default, the module is instantiated in the current design.

- TdiMux/leaf_instance_name : *instance_name* ;

  A property that specifies the leaf instance name of the TDI mux module. The default name is <*design_name*>_<*design_id*>_tessent_tdi_mux_<*tap_id*>_inst. The <*tap_id*> value is the id of the Tap wrapper that contains the Interface/TdiMux wrapper.

- TdiMux/<function> : *port_name* ;

  A set of properties that specify the port names associated with each function on the TDI mux module, and on the TAP controller itself. See the Usage section for a list of the functions and their default values.

  Note

  The TDI mux module contains two control ports: ir_select and bypass_devid_select. These ports are not combined in the TAP module to satisfy ICL modeling rules. IJTAG does not allow combining IRSelect with a Data signal so they must remain separate to the TDI mux. The ICL module models this as two cascaded multiplexers, whereas the Verilog module combines them.

Examples

This example defines all client ports with a “client_” prefix and all host ports with a “host_” prefix. The polarity of tdo_en and reset are both specified as active high.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(module_name,id){
  IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        Interface {
          Client {
            tdi                 : client_si ;
            tdo                 : client_so ;
            tck                 : client_tck ;
            tms                 : client_tms ;
            trst                : client_reset ;
            tdo_en              : client_so_en ;
            tdo_en_polarity     : active_high ;
          }
          Host {
            reset               : host_reset ;
            capture_en          : host_capture_en ;
            shift_en            : host_shift_en ;
            update_en           : host_update_en ;
            reset_polarity      : active_high ;
          }
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tap/HostBScan

The presence of this wrapper specifies that the TAP is to be equipped with a port capable of hosting a boundary scan chain.

Usage

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        HostBscan {
          Interface {
            select             : port_name ; // default:
                                             // host_bscan_to_sel
            force_disable      : port_name ; // default: force_disable
            select_jtag_input  : port_name ; // default:
                                             // select_jtag_input
            select_jtag_output : port_name ; // default:
                                             // select_jtag_output
            extest_pulse       : port_name ; // default: extest_pulse
            extest_train       : port_name ; // default: extest_train
            scan_in            : port_name ; // default:
                                             // host_bscan_from_so
          }
          InstructionCodes {
            CLAMP              : auto | unused | binary, ... ;
            EXTEST             : auto | binary, ... ;
            EXTEST_PULSE       : auto | unused | binary, ... ;
            EXTEST_TRAIN       : auto | unused | binary, ... ;
            INTEST             : auto | unused | binary, ... ;
            SAMPLE_PRELOAD     : auto | binary, ... ;
            HIGHZ              : auto | unused | binary, ... ;
          }
        }
      }
    }
  }
}
```

Description

The presence of this wrapper specifies that the TAP is to be equipped with a port capable of hosting a boundary scan chain.

The interface wrapper is used to change the default port naming of the various functions. When the HostBscan wrapper is present, the ICL description of the Tap includes all the elements described in the section "Requirements on a TAP to be Usable for BoundaryScan" on page 4154 on page 92.

The tool automatically assigns the instruction codes when you specify auto for all instructions. If you want to specify specific binary values, you must specify them for all instructions.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- Interface/select : *port_name* ;

  This property defines the name of the port labelled “bscan_select” in Figure 13-27. The default is “host_bscan_to_sel”.

- Interface/force_disable : *port_name* ;

  This property defines the name of the port labelled “force_disable” in Figure 13-27. The default is “force_disable”.

- Interface/select_jtag_input : *port_name* ;

  This property defines the name of the port labelled “select_jtag_input” in Figure 13-27. The default is “select_jtag_input”.

- Interface/select_jtag_output : *port_name* ;

  This property defines the name of the port labelled “select_jtag_output” in Figure 13-27. The default is “select_jtag_output”.

- Interface/extest_pulse : *port_name* ;

  This property defines the name of the port labelled “extest_pulse” in Figure 13-27. The default is “extest_pulse”. This port is not created when the EXTEST_PULSE property inside the Tap/HostIjtag wrapper is specified as “unused”.

- Interface/extest_train : *port_name* ;

  This property defines the name of the port labelled “extest_train” in Figure 13-27. The default is “extest_train”. This port is not created when the EXTEST_TRAIN property inside the Tap/HostIjtag wrapper is specified as “unused”.

- Interface/scan_in : *port_name* ;

  This property defines the name of the port labelled “from_bscan_so” in Figure 13-27. The default is “host_bscan_from_so”.

- InstructionCodes/CLAMP : auto | unused | binary, ... ;

  A property used to define one or many opcodes for the CLAMP instruction. All codes must differ from the other codes by at least one non-X bit. When left as auto, the tool automatically selects a unique code and describes it in the ICL file when the Tap/HostBScan wrapper is specified. When specified as unused, the instruction is not included in the TAP controller.

- InstructionCodes/EXTEST : auto | binary, ... ;

  A property used to define one or many opcodes for the EXTEST instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the Tap/HostBScan wrapper is specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- InstructionCodes/EXTEST_PULSE : auto | unused | binary, ... ;

  A property used to define one or many opcodes for the EXTEST_PULSE instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the Tap/HostBScan wrapper is specified. When specified as unused, the instruction is not included in the TAP controller.

- InstructionCodes/EXTEST_TRAIN : auto | unused | binary, ... ;

  A property used to define one or many opcodes for the EXTEST_TRAIN instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the Tap/HostBScan wrapper is specified. When specified as unused, the instruction is not included in the TAP controller.

- InstructionCodes/INTEST : auto | unused | binary, ... ;

  A property used to define one or many opcodes for the INTEST instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the Tap/HostBScan wrapper is specified. When specified as unused, the instruction is not included in the TAP controller.

- InstructionCodes/SAMPLE_PRELOAD : auto | binary, ... ;

  A property used to define one or many opcodes for the SAMPLE_PRELOAD instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the Tap/HostBScan wrapper is specified.

- InstructionCodes/HIGHZ : auto | unused | binary, ... ;

  A property used to define one or many opcodes for the HIGHZ instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the Tap/HostBScan wrapper is specified. When specified as unused, the instruction is not included in the TAP controller.

Examples

This example specifies that the TAP is to be equipped with a boundary scan host port. The port naming is left unspecified so the default naming is used. It specifies explicit opcodes for the required instructions related to boundary scan and makes a few of the optional ones unused. Notice that the all 0 opcode is not used as it is reserved for the bypass register. This example assumes no Tap/HostIjtag and no Tap/DataOutPorts wrappers are specified as there would be no room for those opcodes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(mychip,rtl) {
  IjtagNetwork {
    HostScanInterface(tap) {
      Tap(main) {
        bypass_instruction_codes : 2'b00 ;
        HostBscan {
          InstructionCodes {
            CLAMP          : unused ;
            EXTEST         : 2'b01 ;
            EXTEST_PULSE   : unused ;
            EXTEST_TRAIN   : unused ;
            INTEST         : unused ;
            SAMPLE_PRELOAD : 2'b11 ;
            HIGHZ          : unused ;
          }
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tap/DataOutPorts

A wrapper that specifies the number (count) of data-out ports to create on the TAP module, the naming of the ports, the connections to make from them, and whether to insert multiplexing when making the connections.

Usage

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        DataOutPorts {
          count              : integer ;        //default: auto
          port_naming        : port_naming, ;   //default: user_ir_bits[%d]
          multiplexing       : on | off | auto ;
          connection(range)  : port_pin_name ; // repeatable
        }
      }
    }
  }
}
```

Description

You can connect ports, including primary outputs, to Tap/DataOutPorts. The tool will create a port if it does not exist except when the design level is “chip”.

The tool supports %d bus notation and the direct use of a bus name when they do not already exist. For example, suppose a port does not exist. In that case, the tool will create a new port named “new_bus[1:0]” by using either one of these notations:

```
connection(3:2) : new_bus[%d]; // new_bus[1:0]
connection(3:2) : new_bus      // new_bus[1:0]
```

Use %d bus notation to distinguish creation of a scalar port or a bus port. For example, the first notation below creates a new scalar port named “new_port_name” and the second notation using %d creates a single pin bus named “new_port_name[0]”:

```
connection(5) : new_port_name;     // scalar new_port_name
connection(5) : new_port_name[%d]; // single pin bus new_port_name[0]
```

Note

For limitations of the %d notation, refer to the Complex Port Types section of the Verilog and SystemVerilog Limitations table of “HDL Limitations in the Tessent Shell Flow for Verilog and SystemVerilog” on page 5083.

When the ports exist, the tool gives an error when connecting a DataOutPort to an input port. When ports do not exist, the tool gives a direction mismatch error if you specify the same port name for Tdr/DataInPorts and at least one of Tdr/DataOutPorts, Tdr/DecodedSignal, or Tap/

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DataOutPorts. By default, the tool creates an input port for Tdr/DataInPorts. By default, it creates an output port for Tdr/DataOutPorts, Tdr/DecodedSignal, and Tap/DataOutPorts.

Siemens EDA does not recommend using the TAP as a source of DataOut ports. Instead, you should use the Tdr/DataOutPorts wrapper as a source. It is good practice to include a data-out port on a TAP to control the select of a ScanMux that inserts one or many secondary TAP controllers in series with the main one as illustrated in Figure 13-28. Notice that the property output_timing which exists in the Tdr/DataOutPorts wrapper is not present. There is no risk of races here because the TAP data-out ports change in the Update-IR state while all client registers are only reacting to the Update-DR state. When using the data-out port to control the insertion or exclusion of a TAP controller, the TMS signal is clocked on the rising edge of TCK, and the data-out signal changes on the falling edge of TCK thereby removing any risk of races.

Important Note: The bits in the instruction register and, consequently, the value on the ports is overwritten by the following:

- A TAP reset tested in the test_logic_reset test in the PatternsSpecification/Patterns/TestStep/BoundaryScan/RunTest wrapper. The value of the IR register is the determined by the implemented reset value in the hardware.

  You can use a separate TestStep wrapper for the test_logic_reset test in which the user IR bits is overwritten.

- By using the all-1 opcode that is mandated by the IEEE 1149.1 standard.

Arguments

- count : *integer* ;

  A property to request a specific number of data-out ports. If the property is set, or defaults to auto, the count is determined to be large enough to satisfy the *range* value of all connection properties. If no connection property is specified, the count is determined to be large enough to satisfy the port_naming values if they are in a non-indexed format (%d).

- port_naming : *port_naming* ;

  A property that specifies the naming of the data-out ports. The *port_naming* value has two distinct formats: a single string using an index symbol, or a list of port names with explicit sizes.

  When using the indexed format, you specify “base_name[%d]” when you want to use a bus and you specify “base_name%<integer>d” when you want to create a set of scalar ports. In both cases, the range is <count>-1 to 0. Using <integer> in the scalar naming ensures the integer is padded with 0s to be at least <integer> bits wide. For example, “Dout%2d” creates Dout00, Dout01, Dout02, and so on. “Dout%d” and “Dout%1d” mean the same thing. The default is “user_ir_bits[%d]”.

  When using a list of port names, you control the exact naming and width of each data-out port. For example, if you specify “mux_sel,mult[2:0],div[2:0]”, you get exactly seven data-out ports where the most significant bit is called mux_sel and the least significant bit is

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

called div[0] as illustrated in DataOutPort Connection Multiplexing of the Tdr/DataOutPorts.

- multiplexing : on | off | auto ;

  A property that specifies whether to add a multiplexer when making the connection to the specified pin_name of the connection properties. When set to auto, a multiplexer is added unless the pin is unconnected, connected to a constant, or connected to a net with no fanin. As soon as one connection needs multiplexing, an extra data-out port is added to control the select of the multiplexers. The multiplexers are inserted in the direct parent instance of the specified pin_name of the connection properties. See this illustrated in DataOutPort Connection Multiplexing of the Tdr/DataOutPorts wrapper.

- connection(*range*) : *port_pin_name* ;

  A repeatable property that specifies connections to make between bits of the data-out ports and existing pins inside the design. Because data-out ports are sources, the same bit can be connected to multiple destinations by simply repeating the connection property with the same range indexes.

  If the pin name refers to a bused pin and you are connecting to a full bus, you can omit the bus range if the "process_dft_specification -no_insertion" option is not used. If you are using the process_dft_specification -no_insertion switch, you must specify the full range of the destination pin because the range cannot be extracted from the design.

Examples

Example 1

The following example illustrates the specification of four data-out ports with connections to pins on a design instance.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Tap(orange) {
        DataOutPorts {
          connection(0)    : u2/top_green/en ;
          connection(3:1)  : u2/top_green/mode[2:0] ;
        }
      }
    }
  }
}
```

Related Topics

Tdr/DataOutPorts

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tap/DeviceIDRegister

A wrapper that specifies that the TAP controller is to be equipped with a device id register.

Usage

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        DeviceIDRegister {
          Interface {
            version_code       : port_name ; // default: version_code[%d]
            part_number_code   : port_name ; // default:
                                             // part_number_code[%d]
            manufacturer_id_code: port_name ;
                                        // default: manufacturer_id_code[%d]
          }
          capture_source       : internal | external ;
          instruction_codes    : auto | binary, ... ;
          version_code         : binary ;    // default: 4'h0
          manufacturer_id_code : binary ;    // default: 11'h0
          part_number_code     : binary ;    // default: 16'h0
        }
      }
    }
  }
}
```

Description

A wrapper that specifies that the TAP controller is to be equipped with a device id register.

Properties are used to specify the version id, manufacturer id, and the part number codes. When the property capture_source is set to external, the version and part_number codes are captured from ports on the TAP controller. The port names are user-specifiable using the Interface wrapper. The specified version and part_number codes are used as tie values on the version and part_number code ports when instantiating the TAP controller inside the design.

The tool automatically assigns the instruction codes when you specify auto for all instructions. If you want to specify specific binary values, you must specify them for all instructions.

Arguments

- Interface/version_code : *port_name* ;

  This property defines the name of the port labelled “version_code” in Figure 13-27. The specified value must include the “[%d]” or “%<integer>d” symbol. The default is “version_code[%d]”.

- Interface/part_number_code : *port_name* ;

  This property defines the name of the port labelled “part_number_code” in Figure 13-27. The specified value must include the “[%d]” or “%<integer>d” symbol. The default is “part_number_code[%d]”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Interface/manufacturer_id_code : *port_name* ;

  This property defines the name of the port labelled “manufacturer_id_code” in Figure 13-27. The specified value must include the “[%d]” or “%<integer>d” symbol. The default is “manufacturer_id_code[%d]”.

- capture_source : internal | external ;

  A property that defines where the capture value of the part number, version and manufacturer_id codes comes from. When the property is set to external, the manufacturer_id_code, version_code and part_number_code properties are captured from ports on the TAP controller. The manufacturer_id_code, version_code and part_number_code properties are used as tie values on the manufacturer_id, version and part_number ports when instantiating the TAP controller inside the design. When the property is set to internal, no ports are created and the capture value is described in the RTL description of the TAP. If you set this property to internal, you cannot have version and part_number codes specified in the BondingConfigurations wrapper of the BoundaryScan wrapper. The default value is external.

- instruction_codes : auto | *binary*, ... ;

  A property used to define one or many opcodes for the DEVICE_ID instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file.

- version_code : *binary* ;

  A property that specifies a version id code using a 4-bit binary or hexadecimal number. The default value is “4'h0”.

- manufacturer_id_code : *binary* ;

  A property that specifies a manufacturer id code using an 11-bit binary or hexadecimal number. The default value is “11'h0”.

- part_number_code : *binary* ;

  A property that specifies a part number code using a 16-bit binary or hexadecimal number. The default value is “16'h0”.

Examples

This example specifies that the TAP controller is to have a device id register. The version and part number codes are encoded as tie values on ports of the TAP controller. This makes it possible to have version and *part_number* codes specified in the BondingConfigurations wrapper of the BoundaryScan wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification (module_name,id) {
  IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        DeviceIDRegister {
          version_code      : 4'h0 ;
          manufacturer_id_code : 11'hA3 ;
          part_number_code     : 16'h0105 ;
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tap/HostIjtag

A repeatable wrapper that specifies the presence of one or many IJTAG host ports on the TAP controller.

Usage

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        HostIjtag(id) {
          instruction_codes : auto | binary, ... ;
          provide_full_host_scan_interface: auto | on | off ;
          instruction_name  : instruction_name ; // default: HOSTIJTAG_%s
          Interface {
            select          : port_name ; // default: host_%s_to_sel
            scan_in         : port_name ; // default: host_%s_from_so
            scan_out        : port_name ; // default: host_%s_to_si
            reset           : port_name ; // default: host_%s_to_reset
            tck             : port_name ; // default: host_%s_to_tck
            update_en       : port_name ; // default: host_%s_to_ue
            shift_en        : port_name ; // default: host_%s_to_se
            capture_en      : port_name ; // default: host_%s_to_ce
          }
          Sib(id) {                        // repeatable
          }
          Tdr(id) {                        // repeatable
          }
          ScanMux(id) {                    // repeatable
          }
          DesignInstance(instance_name) {  // repeatable
          }
          SecondaryHostScanInterface(id){  // repeatable
          }
        }
      }
    }
  }
}
```

Description

The optional properties enable you to specify a specific opcode to access each IJTAG host port, and to name the ports. You can then specify any other IJTAG node types inside it.

The tool automatically assigns the instruction codes when you specify auto for all instructions. If you want to specify specific binary values, you must specify them for all instructions.

Arguments

- *id*

  A string that uniquely identifies the HostIjtag wrapper. The string contains only letters, numbers, and underscores.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- instruction_codes : auto | *binary*, ... ;

  A property that defines one or many opcodes for the instruction that selects the IJTAG host port. All codes must differ from the other codes specified in the other wrappers such as inside the Tap/HostBScan wrapper by at least one non-X bit and the width must be the same. When left to auto, the tool automatically selects a unique code and describes it in the ICL file.

- provide_full_host_scan_interface : auto | on | off ;

  A property that specifies whether the current TAP IJTAG host port provides all control, scan data, and clock signals to its clients. The default value is auto that resolves to off.

  When the property is set or resolves to off, you can set HostIjtag/Interface/port_names, but the tool does not create those ports for the HostIjtag. When the property is set to on, the tool creates the ports according to their default or specified values.

- instruction_name : *instruction_name* ;

  An optional property that specifies the instruction name associated to the IJTAG host port selection. The specified name may include the %s symbol, which is replaced by the id of the HostIjtag wrapper. The tool capitalizes the instruction name, or it defaults to "HOSTIJTAG_%s" when it is unspecified.

- Interface/select : *port_name* ;

  This property defines the name of the select port of the current host port. The specified value may include the %s symbol, which is replaced by the id of the HostIjtag wrapper. The default is "host_%s_to_sel". This property is processed independently from the provide_full_host_scan_interface property.

- Interface/scan_in : *port_name* ;

  This property defines the name of the scan_in port of the current host port. The specified value may include the %s symbol, which is replaced by the id of the HostIjtag wrapper. The default is "host_%s_from_so". This property is processed independently from the provide_full_host_scan_interface property.

- Interface/scan_out : *port_name* ;

  This property defines the name of the scan_out port of the current host port. The specified value may include the %s symbol, which is replaced by the id of the HostIjtag wrapper. The default is "host_%s_to_si". This property is only processed when the provide_full_host_scan_interface property is set or resolves to on.

- Interface/reset : *port_name* ;

  This property defines the name of the reset port of the current host port. The specified value may include the %s symbol, which is replaced by the id of the HostIjtag wrapper. The default is "host_%s_to_reset". This property is only processed when the provide_full_host_scan_interface property is set or resolves to on.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Interface/tck : *port_name* ;

  This property defines the name of the tck port of the current host port. The specified value may include the %s symbol, which is replaced by the id of the HostIjtag wrapper. The default is “host_%s_to_tck”. This property is only processed when the provide_full_host_scan_interface property is set or resolves to on.

- Interface/update_en : *port_name* ;

  This property defines the name of the update_en port of the current host port. The specified value may include the %s symbol, which is replaced by the id of the HostIjtag wrapper. The default is “host_%s_to_update_en”. This property is only processed when the provide_full_host_scan_interface property is set or resolves to on.

- Interface/shift_en : *port_name* ;

  This property defines the name of the shift_en port of the current host port. The specified value may include the %s symbol, which is replaced by the id of the HostIjtag wrapper. The default is “host_%s_to_shift_en”. This property is only processed when the provide_full_host_scan_interface property is set or resolves to on.

- Interface/capture_en : *port_name* ;

  This property defines the name of the capture_en port of the current host port. The specified value may include the %s symbol, which is replaced by the id of the HostIjtag wrapper. The default is “host_%s_to_capture_en”. This property is only processed when the provide_full_host_scan_interface property is set or resolves to on.

Examples

This example defines two IJTAG host scan interfaces.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(my_chip,rtl) {
  IjtagNetwork {
    HostScanInterface(tap) {
      Tap(main) {
        HostIjtag(mission) {
          provide_full_host_scan_interface : off;
          Interface {
            select  : host_%s_to_sel;  // host_mission_to_sel
            scan_in : host_%s_from_so; // host_mission_from_so
          }
        }
        HostIjtag(debug) {
          provide_full_host_scan_interface : off;
          Interface {
            select     : second_interface_to_sel ;
            scan_in    : second_interface_from_so ;
            scan_out   : second_interface_to_si ;
            reset      : second_interface_to_reset ;
            tck        : second_interface_to_tck ;
            update_en  : second_interface_%s_to_ue ;
                                        // second_interface_debug_to_ue
            shift_en   : second_interface_%s_to_se ;
                                        // second_interface_debug_to_se
            capture_en : second_interface_%s_to_ce ;
                                        // second_interface_debug_to_ce
          }
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tap/HostSTap

A wrapper that specifies the creation of a host STAP port on the TAP, which creates an IEEE 1838-compliant PTAP. You can also specify the creation of STAP controller and STAP pipeline nodes within this wrapper.

Usage

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        HostSTap {
          instruction_code : auto | binary ;
          Interface {
            scan_out   : port_name ; // default: stap_to_si
            scan_in    : port_name ; // default: stap_from_so
            tms        : port_name ; // default: stap_to_tms
            capture_en : port_name ; // default: stap_to_ce
            shift_en   : port_name ; // default: stap_to_se
            update_en  : port_name ; // default: stap_to_ue
          }

          STap(id) {
            parent_instance    : instance_name ;
            leaf_instance_name : instance_name ;
            so_retiming        : on | off ;
            Interface {
              Client {
                scan_in    : port_name ; // default: stap_si
                scan_out   : port_name ; // default: stap_so
                tms        : port_name ; // default: tms
                tck        : port_name ; // default: tck
                trst       : port_name ; // default: trst
                reset      : port_name ; // default: ijtag_reset
                capture_en : port_name ; // default: stap_ce
                shift_en   : port_name ; // default: stap_se
                update_en  : port_name ; // default: stap_ue
              }
              Host {
                scan_out : port_name ; // default: stap_to_tdi
                scan_in  : port_name ; // default: stap_from_tdo
                tms      : port_name ; // default: stap_to_tms
              }
            }

            SecondaryHostScanInterface(id) {
              to_tck   : port_pin_name ; // default: %s_to_tck   *DefSpec
              to_trst  : port_pin_name ; // default: %s_to_trst  *DefSpec
              to_tms   : port_pin_name ; // default: %s_to_tms   *DefSpec
              from_tdo : port_pin_name ; // default: %s_from_tdo *DefSpec
              to_tdi   : port_pin_name ; // default: %s_to_tdi   *DefSpec
            }
          }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
            STapPipeline(id) {
              parent_instance    : instance_name ;
              leaf_instance_name : instance_name ;
              so_retiming        : on | off ;
              Interface {
                scan_in  : port_name ; // default: stap_si
                scan_out : port_name ; // default: stap_so
                tck      : port_name ; // default: tck
                shift_en : port_name ; // default: stap_se
              }
            }

            SecondaryHostScanInterface(id) {
              to_tck   : port_pin_name ; //def: %s_ijtag_to_tck    *DefSpec
              to_trst  : port_pin_name ; //def: %s_to_trst         *DefSpec
              to_reset : port_pin_name ; //def: %s_ijtag_to_reset  *DefSpec
              to_reset_polarity : active_low | active_high ;
              to_tms   : port_pin_name ; //def: %s_to_tms          *DefSpec
              to_capture_en: port_pin_name ; //def: %s_ijtag_to_ce *DefSpec
              to_shift_en  : port_pin_name ; //def: %s_ijtag_to_se *DefSpec
              to_update_en : port_pin_name ; //def: %s_ijtag_to_ue *DefSpec
              from_tdo     : port_pin_name ; //def: %s_from_tdo    *DefSpec
              to_tdi       : port_pin_name ; //def: %s_to_tdi      *DefSpec
            }
          }
        }
      }
    }
  }
```

Description

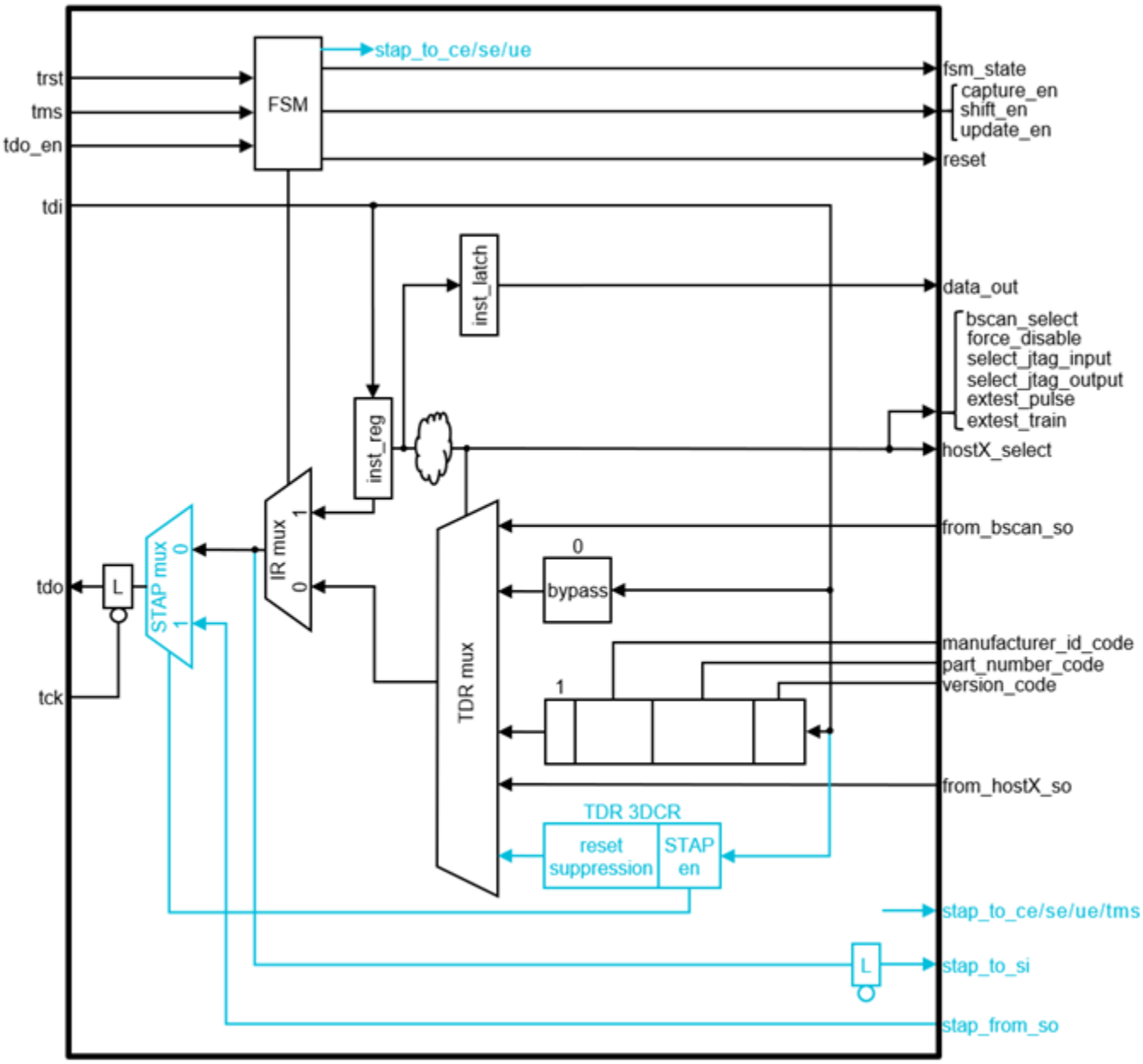
Specifying the HostSTap wrapper within the Tap wrapper creates an IEEE 1838 PTAP controller. A PTAP is a normal TAP controller with a special STAP host port that enables daisy-chaining STAP controllers after the TAP's instruction or selected data register.

Figure 13-30 shows a block diagram of a normal TAP controller with the additions that make it a PTAP shown in blue. The STAP multiplexer driving TDO places the nodes connected between the stap_to_si and stap_from_so ports in series with the IRMux of the normal TAP controller. The elements between the stap_to_si and stap_from_so ports are active during IR and DR TAP operations and therefore require a special set of controls (stap_to_ce/se/ue) that are active during both IR and DR operations. The normal capture_en, shift_en, and update_en ports on the TAP are only active during the DR operations.

A 2-bit data register within the TAP controller controls the select of the STAP mux. Bit 1 controls the select of the STAP mux, while bit 0 determines if a 5-TMS reset resets bit 1. The IEEE 1838 standard refers to this as reset suppression. A 5-TMS reset never resets bit 0, which is the reset suppression bit. However, both bits reset when TRST is active. This feature lets you configure the STAP network and have it undisturbed by 5-TMS resets.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-30. Addition to TAP to Make it a PTAP



STAP controllers and pipeline stages are two types of nodes that connect to the STAP host port of a PTAP. The pipeline stage is a positive edge flip-flop followed by an optional retiming flop. The positive edge flip-flop holds its value when the stap_to_se port of the PTAP is low. Adding a pipeline stage that does not hold when the scan_en is low violates the standard and would prevent entering the pause_ir or pause_dr states in the middle of scan load, as supported by a compliant IEEE 1687 or IEEE 1838 network.

The STAP node is like a SIB within an IJTAG network. The STAP opens or closes the path to the PTAP in the die above. Figure 13-31 shows a block diagram of an STAP controller implemented within the tool. The black elements show the minimum element set required by

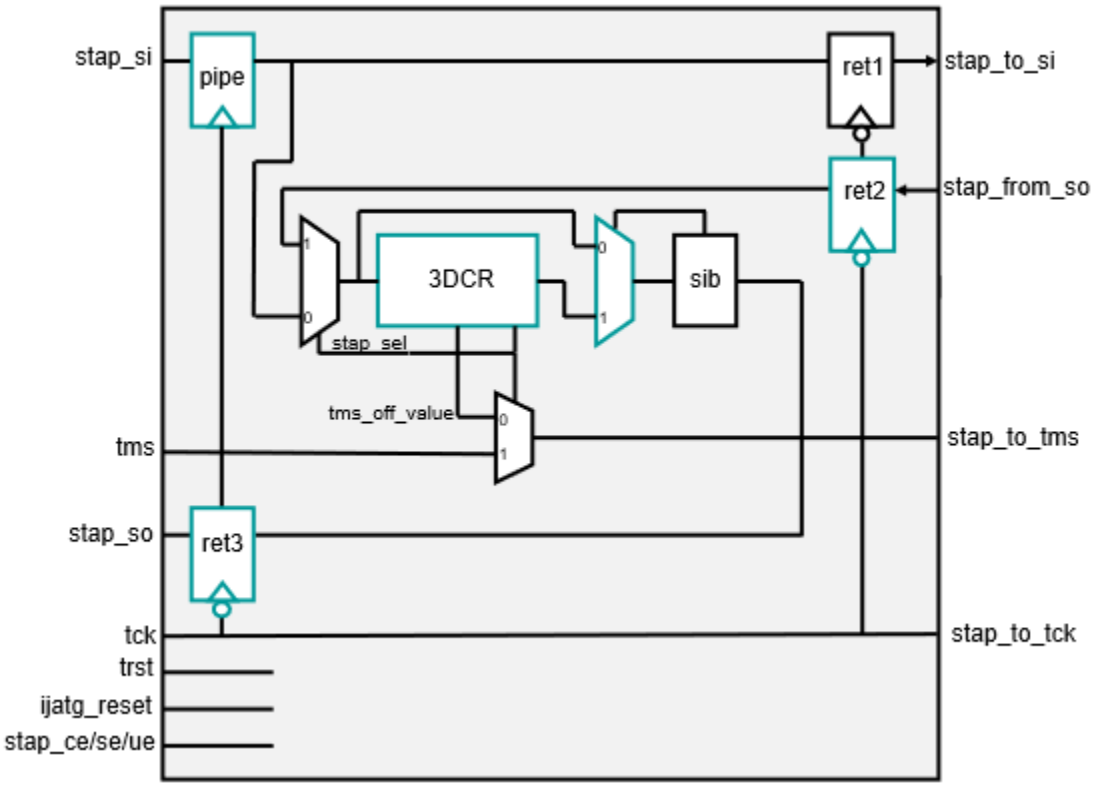
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

the IEEE 1838 standard. The green elements show additional, and IEEE 1838 compliant, pipeline and retiming elements for timing closure and to localize the enabling of the STAP with an in-line 3-bit register, named 3DCR.

Bit 0 of the 3DCR register is the reset suppression bit, which, when set, disables the 5-TMS reset on the next two bits. Bit 1 controls the value of the stap_to_tms port when the STAP is deactivated. While the STAP is disabled and stap_to_tms is held high, the PTAP it hosts holds in the reset state. While the STAP is disabled and stap_to_tms is held low, the PTAP it hosts holds in the run_test_idle state. Lastly, bit 2 controls whether the STAP opens the path to its hosted PTAP or bypasses it.

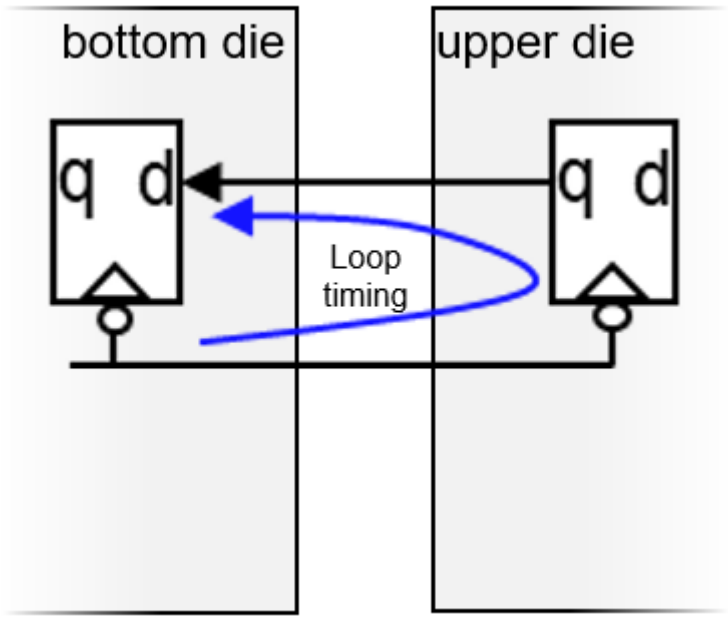
The “ret2” register on the stap_from_so port provides a full TCK period for the TCK clock to propagate from the lower die to the upper die and then for the upper die to return the TDO data on the negative TCK edge. This loop timing delay, shown in Figure 13-32, is always positive, so we can use a negedge-to-negedge path and provide the full TCK period to meet the loop timing path.

Figure 13-31. STAP Block Diagram



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-32. TDO Loop Timing From Upper Die to Lower Die



You can also specify the STap and STapPipeline nodes in the IjtagNetwork/HostScanInterface wrapper, as shown in Example 2. You typically have a second HostScanInterface wrapper to specify the IJTAG interface. In that case, you must specify different port names for the capture_en, shift_en, and update_en ports within the HostScanInterface/Interface wrapper compared to the STAP network wrapper. This is because their default names would default to the same name as the IJTAG scan interface ports, and the ports must be distinct because they have different behaviors. For example, the IJTAG ports are only active during the DR operations, while those for STAP are active during both the DR and IR operations. The proper port naming automatically occurs if you use “create_dft_specification -stap_host_list”, as shown in Example 2. The IjtagNetwork/HostScanInterface(stap) wrapper in the DefaultsSpecification specifies the default port names for a STAP HostScanInterface.

Arguments

- instruction_code : auto | *binary* ;

  An optional property that explicitly specifies the Instruction value that selects the 3DCR register within the PTAP. The default value is auto, which automatically assigns an opcode along with all other opcodes used to select the registers of the TAP controller.

- Interface/scan_out : *port_name* ;

  An optional property that specifies the scan_out port name of the STAP host port. The default name is “stap_to_si”.

- Interface/scan_in : *port_name* ;

  An optional property that specifies the scan_in port name of the STAP host port. The default name is “stap_from_so”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Interface/tms : *port_name* ;

  An optional property that specifies the to_tms port name of the STAP host port. The default name is “stap_to_tms”. This output port is the TAP TMS input port, which is driven low when the stap_en bit of the 3DCR is off.

- Interface/capture_en : *port_name* ;

  An optional property that specifies the to_capture_en port name of the STAP host port. The default name is “stap_to_ce”. This port is active during both the capture_dr and capture_ir states of the TAP and launches from the negedge of TCK to meet setup and hold time requirements around the posedge of TCK at the destination.

- Interface/shift_en : *port_name* ;

  An optional property that specifies the to_shift_en port name of the STAP host port. The default name is “stap_to_se”. This port is active during both the shift_dr and shift_ir states of the TAP and launches from the negedge of TCK to meet setup and hold time requirements around the posedge of TCK at the destination.

- Interface/update_en : *port_name* ;

  An optional property that specifies the to_update_en port name of the STAP host port. The default name is “stap_to_ue”. This port is active during both the update_dr and update_ir states of the TAP and launches from the posedge of TCK to meet setup and hold time requirements around the negedge of TCK at the destination.

- STap

  An optional wrapper within the Tap/HostSTap wrapper that specifies the generation of an STAP node, as shown in Figure 13-31. You can also specify the STap wrapper within the IjtagNetwork/HostScanInterface wrapper when the PTAP and the STAP nodes are in different physical blocks. See the tiling flow example in Example 2.

- STap/parent_instance : *instance_name* ;

  An optional property that specifies the instance in which to instantiate the STAP node. By default, the STAP node is instantiated in the current design.

- STap/leaf_instance_name : *instance_name* ;

  An optional string property that specifies the leaf instance name of the STAP. The default name is <*design_name*>_<*dft_spec_id*>_tessent_stap_<*stap_id*>_inst. The <*dft_spec_id*> string is the id specified in the DftSpecification wrapper, and the <*stap_id*> is the id of the STap wrapper. Tessent Shell checks for conflicts before instantiating the STAP controller and, if needed, applies the uniquification suffix specified by the set_insertion_options command.

- STap/so_retiming : <u>on</u> | off ;

  An optional switch that specifies the presence of a retiming flop (identified as “ret3” in Figure 13-31) on the scan_out port of the STAP node. You can remove the retiming node when the TCK clock skew to the STAP node is balanced with the node that follows it, or when it is the last node of a HostScanInterface of type STAP in a tile design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- STap/Interface/Client/<function> : *port_name* ;

  An optional set of properties that name the ports on the Client interface of the STAP node. The left side of the block diagram in Figure 13-31 shows these ports. Refer to the Usage section for a list of the functions and their default values.

- STap/Interface/Host/<function>: *port_pin_name* ;

  An optional set of properties that name the ports on the Host interface of the STAP node. The right of the block diagram in Figure 13-31 shows these ports. Refer to the Usage section for a list of the functions and their default values.

- STapPipeline

  An optional wrapper within the Tap/HostSTap wrapper that specifies the generation of an STAP pipeline node. You can also specify the STapPipeline wrapper within the IjtagNetwork/HostScanInterface wrapper when the PTAP and the STAP nodes are in different physical blocks. See the tiling flow example in Example 2.

- STapPipeline/parent_instance : *instance_name* ;

  An optional property that specifies the instance in which to instantiate the STAP pipeline node. By default, the STAP pipeline node is instantiated in the current design.

- STapPipeline/leaf_instance_name : *instance_name* ;

  An optional string property that specifies the leaf instance name of the STAP pipeline. The default name is <*design_name*>_<*dft_spec_id*>_tessent_stap_pipeline_<*stap_id*>_inst. The <*dft_spec_id*> string is the id specified in the DftSpecification wrapper, and the <*stap_id*> is the id of the STapPipeline wrapper. Tessent Shell checks for conflicts before instantiating the STAP pipeline and, if needed, applies the uniquification suffix specified by the set_insertion_options.

- STapPipeline/so_retiming : <u>on</u> | off ;

  An optional switch that specifies the presence of a retiming flop on the scan_out port of the STAP pipeline node. You can remove the retiming node when the TCK clock skew to the STAP pipeline node is balanced with the node that follows it.

  You also remove it when the pipeline stage is the last one in a tile, and you have a proper loop timing interface with the node sourcing the TCK clock and receiving the scan_out data as shown in Example 2.

- STapPipeline/Interface/<function> : *port_name* ;

  An optional set of properties that name the ports of the STAP pipeline node. Refer to the Usage section for a list of the functions and their default values.

- HostSTap/SecondaryHostScanInterface

  An optional wrapper that specifies an exit point for the STAP network when the PTAP and the STAP controllers are in different physical blocks as shown in Example 2 .

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Refer to the HostSTap/SecondaryHostScanInterface wrapper in the Usage section to find the relevant properties within this context and their default values. You can refer to SecondaryHostScanInterface for a description of these properties.

When you create this wrapper using "create_dft_specification -stap_host_list", the proper port naming occurs automatically using the default values specified in the IjtagNetwork/SecondaryHostScanInterface(STap) wrapper.

- STap/SecondaryHostScanInterface

  An optional wrapper that specifies the port names of the STAP host ports connected to the host interface of the STAP node.

  Refer to the STap/SecondaryHostScanInterface wrapper in the Usage section to find the relevant properties within this context and their default values. You can refer to SecondaryHostScanInterface for a description of these properties.

  When you create this wrapper using "create_dft_specification -stap_host_list", the proper port naming occurs automatically using the default values specified in the IjtagNetwork/SecondaryHostScanInterface(Tap) wrapper.

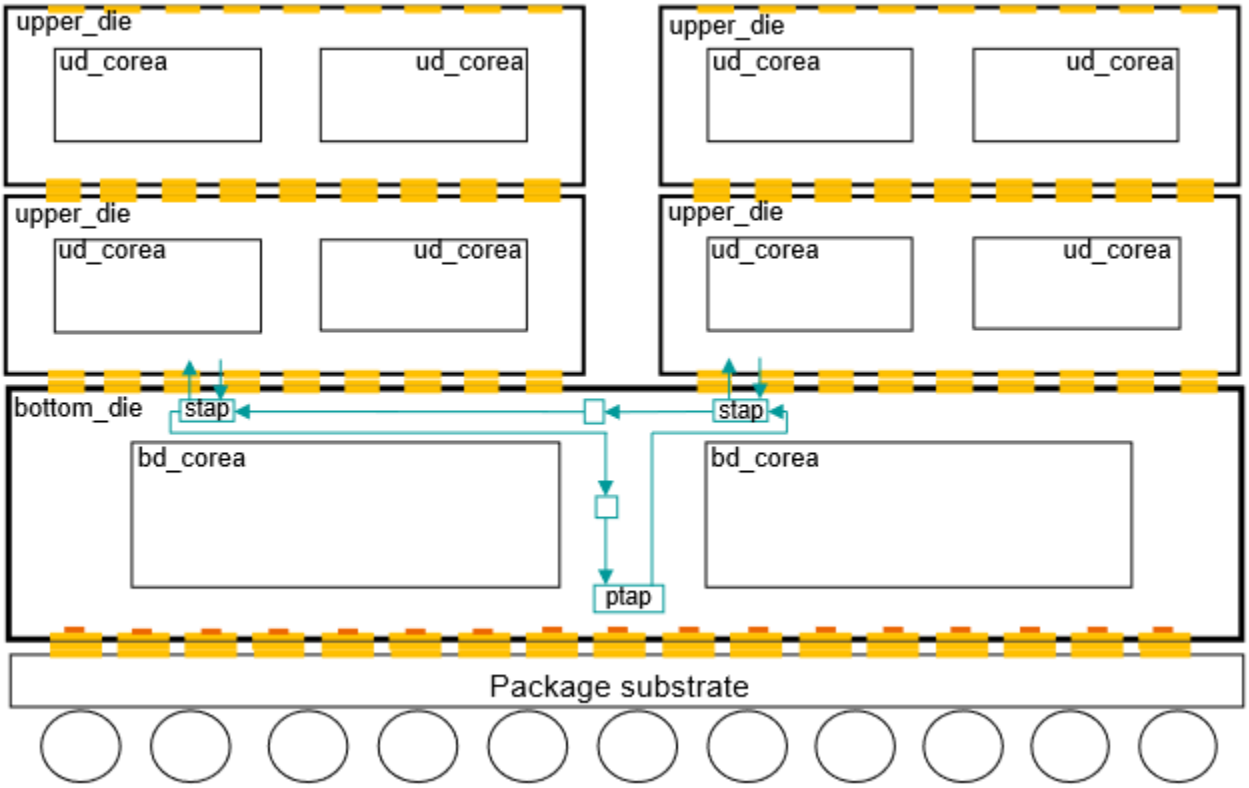
Examples

Example 1

This example describes how you insert an STAP network within the top physical blocks of a multi-die stack. One die is at the bottom of the 3D stack with two columns of die above it, as shown in Figure 13-33.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-33. PTAP/STAP Network Inserted Within the Top Physical Blocks of a Multi-die Stack



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

To insert the STAP network, you request the creation of the two STAP nodes and the STAP pipelines using the -stap_host_list switch when calling the create_dft_specification command:

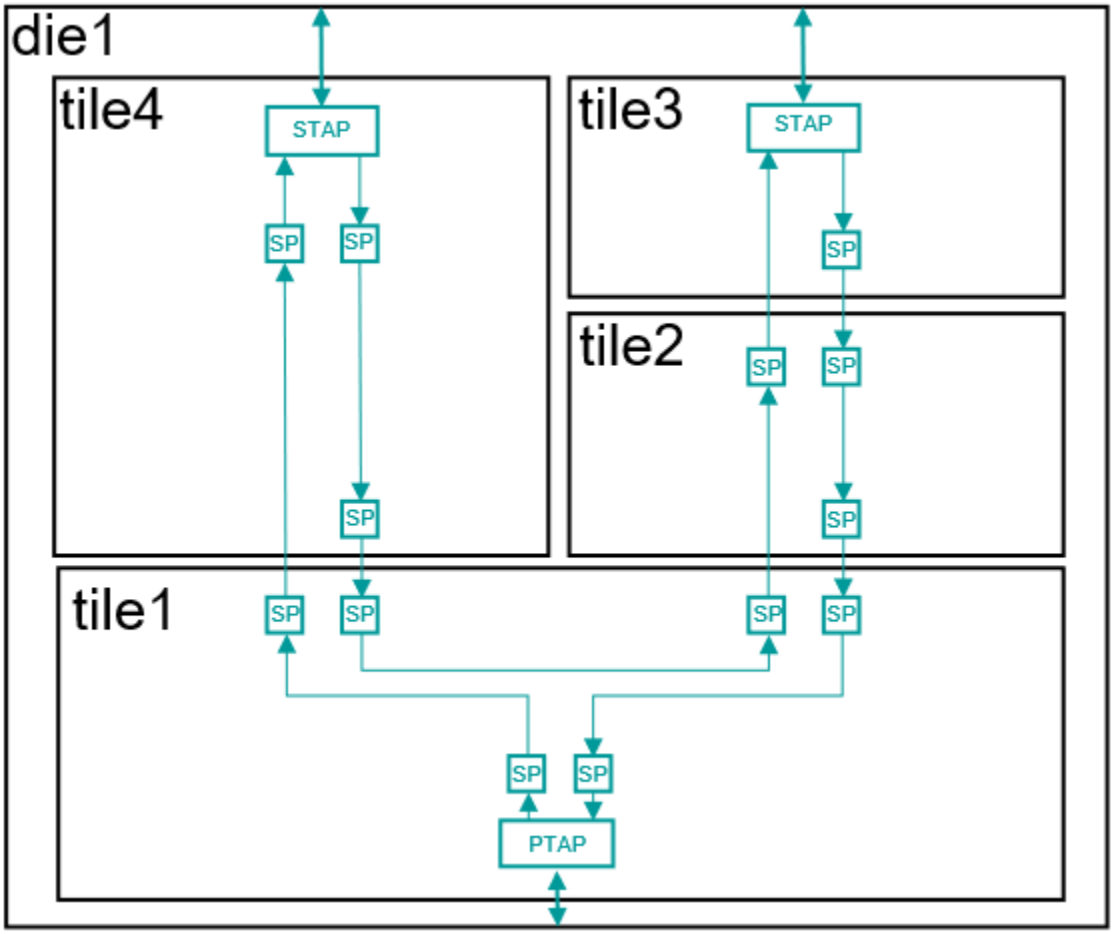
```
> set spec [create_dft_spec -stap_host_list [list right \
                                                   pipe(between) \
                                                   left \
                                                   pipe(end)]]
> report_config_data $spec
DftSpecification(bottom_die,rtl1) {
  IjtagNetwork {
    HostScanInterface(tap) {
      Interface {
        tck : my_tck;
        trst : my_trst;
        tms : my_tms;
        tdi : my_tdi;
        tdo : my_tdo;
      }
      Tap(main) {
        HostIjtag(1) {
         // Normal IJTAG network removed for clarity
        }
        HostBscan {
        }
        HostSTap {
          STapPipeline(end) {
          }
          STap(left) {
            SecondaryHostScanInterface(stap_left) {
            }
          }
          STapPipeline(between) {
          }
          STap(right) {
            SecondaryHostScanInterface(stap_right) {
            }
          }
        }
      }
    }
  }
}
```

Example 2

This example describes how you insert a STAP network within a tiled design. As shown in Figure 13-34, the example inserts a PTAP into Tile1 and an STAP node into Tile3 and Tile4. Tile2 only includes STAP pipeline nodes, which connect the STAP network between Tile1 and Tile3 through Tile2.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-34. PTAP/STAP Network Inserted into a Tiled Die



The following code shows the commands to process each tile and the DFT specification that results for each tile. Notice how the node closest to TDO in the DftSpecification of tile2, tile3, and tile4 have the so_retiming property set to “off”. This is due to specifying the set_dft_specification_requirements -design_type switch as “tile”. The removal of the scan out retiming element provides a full TCK period for the interface loop timing.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tile1

```
> set_dft_specification_requirements -design_type tile \
                                     -host_scan_interface_type tap
> check_design_rules
> set spec [create_dft_specification -stap_host_list {
                pipe(start) \
                pipe(to_left) \
                sec_host(host_left_stap) \
                pipe(from_left) \
                pipe(to_right) \
                sec_host(host_right_stap) \
                pipe(from_right) \
                pipe(end)}]
> report_config_data $spec
DftSpecification(tile1,rtl) {
  IjtagNetwork {
    HostScanInterface(tap) {
      Tap(main) {
        HostIjtag(1) {
          // Content removed for clarity
        }
        HostBScan {
        }
        HostSTap {
          STapPipeline(end) {
          }
          STapPipeline(from_right) {
          }
          SecondaryHostScanInterface(host_right_stap) {
            to_tck           : %s_to_tck;
          }
          STapPipeline(to_right) {
          }
          STapPipeline(from_left) {
          }
          SecondaryHostScanInterface(host_left_stap) {
            to_tck           : %s_to_tck;
          }
          STapPipeline(to_left) {
          }

          STapPipeline(start) {
          }
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tile2

```
> set_dft_specification_requirements -design_type tile
> check_design_rules
> set spec [create_dft_specification -stap_host_list {
                pipe(to_left) \
                sec_host(host_stap) \
                pipe(from_host) \
                pipe(end)}]
> report_config_data $spec
DftSpecification(tile1,rtl) {
  IjtagNetwork {
    HostScanInterface(stap) {
      Interface {
        tdi        : stap_tdi;
        tdo        : stap_tdo;
        tms        : stap_tms;
        tck        : stap_tck;
        trst       : stap_trst;
        reset      : stap_ijtag_reset;
        capture_en : stap_ijtag_ce;
        shift_en   : stap_ijtag_se;
        update_en  : stap_ijtag_ue;
      }
      STapPipeline(end) {
        so_retiming : off;
      }
      STapPipeline(from_host) {
      }
      SecondaryHostScanInterface(host_stap) {
        to_tck     : %s_to_tck;
      }
      STapPipeline(to_host) {
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Tile 3
> set_dft_specification_requirements -design_type tile
> check_design_rules
> set spec [create_dft_specification -stap_host_list {
                host \
                pipe(end)}]
> report_config_data $spec
DftSpecification(tile1,rtl) {
  IjtagNetwork{
    HostScanInterface(stap) {
      Interface {
        tdi     : stap_tdi;
        tdo        : stap_tdo;
        tms        : stap_tms;
        tck        : stap_tck;
        trst       : stap_trst;
     reset      : stap_ijtag_reset;
        capture_en : stap_ijtag_ce;
        shift_en   : stap_ijtag_se;
        update_en  : stap_ijtag_ue;
      }
      STapPipeline(end) {
        so_retiming : off;
      }
      STap(host) {
        SecondaryHostScanInterface(stap_host) {
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tile4

```
> set_dft_specification_requirements -design_type tile
> check_design_rules
> set spec [create_dft_specification -stap_host_list {
                pipe(to_host) \
                host \
                pipe(from_host)\
                pipe(end)}]
> report_config_data $spec
DftSpecification(tile1,rtl) {
  IjtagNetwork {
    HostScanInterface(stap) {
      Interface {
        tdi        : stap_tdi;
        tdo        : stap_tdo;
        tms        : stap_tms;
        tck        : stap_tck;
        trst       : stap_trst;
        reset      : stap_ijtag_reset;
        capture_en : stap_ijtag_ce;
        shift_en   : stap_ijtag_se;
        update_en  : stap_ijtag_ue;
      }
      STapPipeline(end) {
        so_retiming : off;
      }
      STapPipeline(from_host) {
      }
      STap(host) {
        SecondaryHostScanInterface(stap_host) {
        }
      }
      STapPipeline(to_host) {
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TwoPinSerialPort

A wrapper that specifies the contents and position of a Two-Pin Serial Port (TPSP) controller in the IjtagNetwork.

Usage

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(id) {
      TwoPinSerialPort {
      parent_instance              : instance_name ;
      leaf_instance_name           : leaf_instance_name ;
      Interface {
        Client {
           spio_in  : port_name ; // default: spio_in
           spclk    : port_name ; // default: spclk
           spio_out : port_name ; // default: spio_out
           spio_en  : port_name ; // default: spio_en
           spio_en_polarity : auto | active_high | active_low;
        }
        Host {
           tck      : port_name ; // default: tck
           tms      : port_name ; // default: tms
           tdi      : port_name ; // default: tdi
           tdo      : port_name ; // default: tdo
           tdo_en   : port_name ; // default: tdo_en
           tdo_en_polarity : auto | active_high | active_low;
        }
      }
      Tap(id) { // repeatable
      }
      DesignInstance(id) { // repeatable
      }
      ScanMux(id) { // repeatable
      }
    }
  }
}
```

Description

Figure 1 shows the block diagram of the TPSP module. A TPSP is an interface consisting of only two top level pins at the device's IO, that internally drive a standard JTAG TAP controller, which in turn drives an IJTAG network. The TPSP controller has a client side (the spclk and spio_* ports) and a host side (tdi, tck, tms, trst and tdo). The client side is directly connected to the device's IO ports, where the SPCLK (serial protocol clock) is one input to the TPSP's spclk, and a bidirectional SPIO port SPIO provides both input to spio_in and output to spio_out, with the output enable signal spio_en. The ports at the TPSP module are either input or output. It does not have bidi pins itself. Only the top level design has such a port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-35. TPSP Connections Schematic

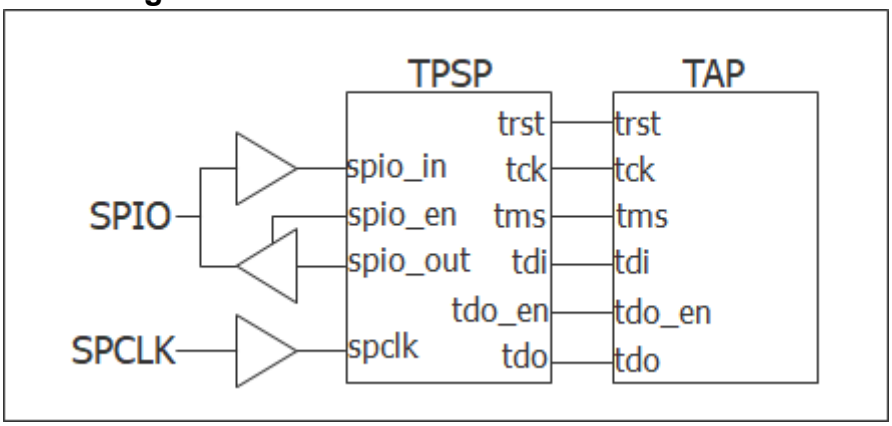


Figure 2 shows the timing diagram for the TPSP controller protocol. The iReset is required to initialize the whole IJTAG network at the beginning of the test. Following the iReset, the TAP state machine gets set depending on the value of the TMS signal, which is generated by the TPSP controller, at the rising edge of TCK.

Figure 13-36. TPSP Timing Diagram

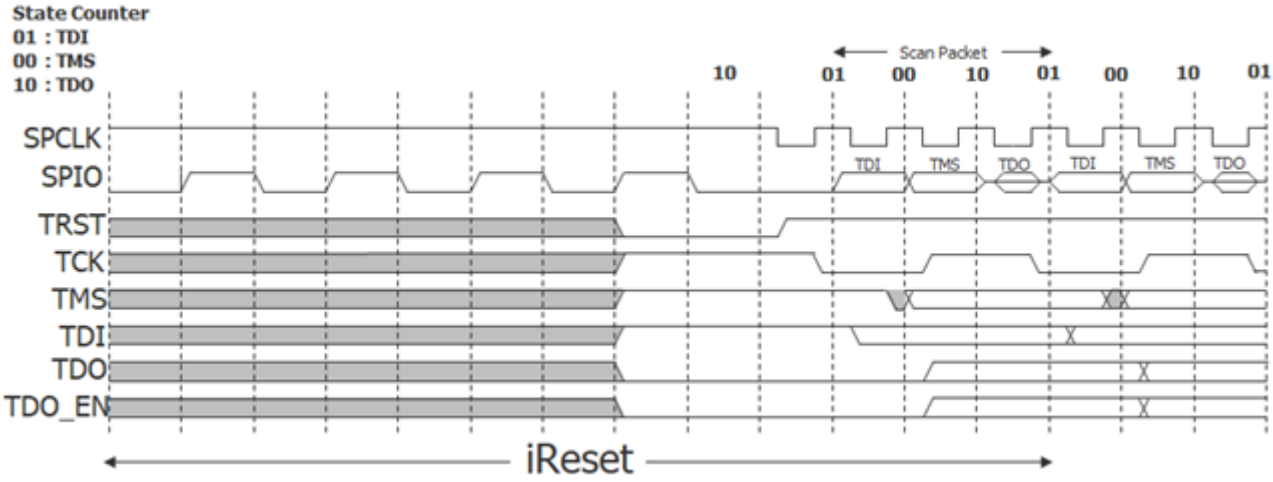
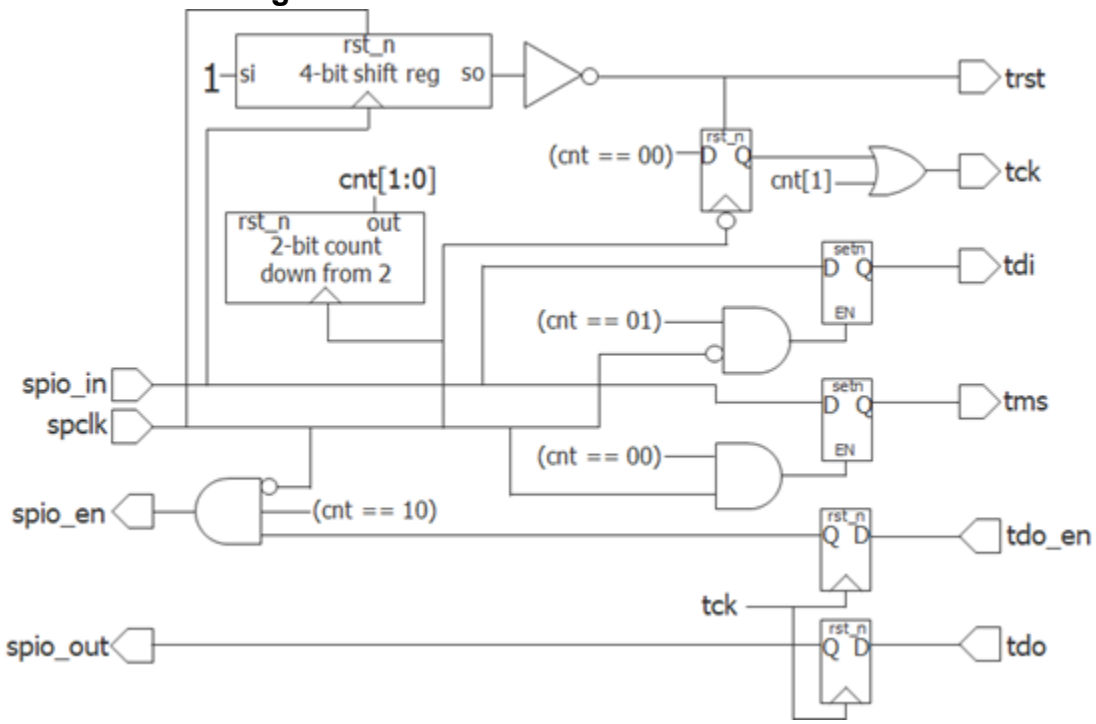


Figure 3 shows the internal hardware for the TPSP controller. The hardware of the TPSP controller will vary slightly depending on the chosen polarity for tdo_en and spio_en.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-37. TPSP Hardware Schematic



Note

The TPSP requires an inout port with a traceable PAD cell which has an internal pull resistor.

Tip

For a simulation tip, refer to the topic "How to Avoid Simulation Issues When Using the Two-Pin Serial Port Controller" in the *Tessent Shell User's Manual* .

Note

The TPSP controller is not compatible with Tessent BoundaryScan. The IEEE 1149.1 BSDL standard only supports a TAP interface. It does not support a 2-pin interface.

Arguments

- parent_instance : *instance_name* ;

  A string property that specifies the parent instance in which to instantiate the TPSP module. If unspecified, the TPSP is instantiated in the root module. You can use this property to push a TPSP down near the client interfaces it interacts with.

  If the specified instance_name is an instance of a repeated module, the tool issues a warning during validation and automatically issues a uniquify_instances command as part of the insertion process to uniquify the instance_name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

To perform IJTAG network insertion inside a module that is used more than once without uniquifying it, you can set the current design to that module, insert the IJTAG network in it, and refer to it with a DesignInstance wrapper at the next level up. This bottom-up insertion process can be done by going back and forth between setup and insertion mode within a single Tessent Shell invocation. You simply change the current design using the set_current_design command in setup mode.

- leaf_instance_name : *leaf_instance_name* ;

  A string property that specifies the leaf instance name of the TPSP.

- Client/spio_in: *port_name*;

  This property defines the name of the port labeled “spio_in” in Figure 13-28. The default name when unspecified is “spio_in”.

- Client/spclk: *port_name*;

  This property defines the name of the port labeled “spclk” in Figure 13-28. The default name when unspecified is “spclk”.

- Client/spio_out: *port_name*;

  This property defines the name of the port labeled “spio_out” in Figure 13-28. The default name when unspecified is “spio_out”.

- Client/spio_en: *port_name*;

  This property defines the name of the port labeled “spio_en” in Figure 13-28. The default name when unspecified is “spio_en”.

- Client/spio_en_polarity: *port_name*;

  This property controls the polarity of the spio_en port. The default is auto. When the DftSpecification is validated by the process_dft_specification command, if the value is specified or defaults to "auto", the property is automatically resolved to match the polarity of the enable pin for the output pad buffer associated with the spio port.

- Host/tck : *port_name* ;

  This property defines the name of the port labelled “tck” in Figure 13-28. The default name when unspecified is “tck”.

- Host/tms : *port_name* ;

  This property defines the name of the port labelled “tms” in Figure 13-28. The default name when unspecified is “tms”.

- Host/tdi : *port_name* ;

  This property defines the name of the port labelled “tdi” in Figure 13-28. The default name when unspecified is “tdi”.

- Host/tdo : *port_name* ;

  This property defines the name of the port labelled “tdo” in Figure 13-28. The default name when unspecified is “tdo”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Host/tdo_en : *port_name* ;

  This property defines the name of the port labelled “tdo_en” in Figure 13-28. The default name when unspecified is “tdo_en”.

- Host/tdo_en_polarity: *port_name*;

  This property controls the polarity of the tdo_en port. The default is auto. When the DftSpecification is validated by the process_dft_specification command, if the value is specified or defaults to "auto", the property is automatically resolved to match the polarity of the tdo_en of TAP controllers hosted by this TPSP controller.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

InSystemTest

Specifies the generation and optional insertion of an In-System Test (IST) controller in the design using the DftSpecification wrapper.

Usage

```
DftSpecification (module_name, id) {
  InSystemTest {
    Controller(id) {
      host_interface            : name ;       // mutually exclusive
                                               // with DesignInstance
      DesignInstance (instance_name) {         // mutually exclusive
        client_interface        : name ;       // with host_interface
      }
      protocol                  : direct_memory_access | cpu_interface ;
      data_width                : integer ;    // default: 8
      parent_instance           : name ;       // default: ""
      leaf_instance_name        : name ;
      async_reset_all_registers : on | off ;
      use_enable_sync_cell      : on | off ;
      tap_reset_method          : asynchronous | synchronous ;
      ControllerChain {
        present                 : on | off ;
        clock                   : tck | ccm_clock | ist_clock ;
        segment_per_instrument  : on | off ;
        max_segment_length      : int | unlimited; // int >= 32
      }
      DirectMemoryAccessOptions {
      }
      Interface {
      }
      Connections {
      }
    }
  }
}
```

Description

InSystemTest is an optional wrapper within the DftSpecification that you use to generate and optionally insert an IST controller in the design. You can run in-system tests during power up of the device (Power-On Self Test or POST) or run on demand during system operation.

The IST controller is a unique, simple alphanumeric *id*. The *id* should start with an alphanumeric character and can contain no special characters except for underscores (_) after the first character. For example, the following controller ids are valid:

```
Controller(1)
Controller(c1)
Controller(controller_1)
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The Controller wrapper is a repeatable wrapper. You can specify more than one IST controller. For any controller, you can specify the ControllerChain wrapper to configure Controller Chain Mode.

Specify the IST controller using the host_interface property or the DesignInstance wrapper. Use the host_interface property to specify the scan interface that connects the IST controller when the IJTAG network is inserted at the same time as the IST controller. The *name* is the HostScanInterface() wrapper name inside the IjtagNetwork specification.

Use the DesignInstance wrapper to specify the scan interface connected to the controller when the IJTAG network was inserted prior to the IST controller. The *instance_name* is an instance name of a module with a matching ICL description. Specifying an IST controller without either a pre-existing or co-generated intercept is not allowed.

For details about Tessent MissionMode, refer to the *Tessent MissionMode User's Manual*.

Arguments

- host_interface : *name* **;**

  A required property that specifies the scan interface that is connected to the controller when the IJTAG network is inserted at the same time as the controller. This property points to the HostScanInterface() wrapper inside the IjtagNetwork Specification. This is the interface at which the controller is connected and the in-system test applied. This property cannot be specified with the DesignInstance() wrapper.

  Note

  You may have a situation with two host scan interfaces, where the external host scan interface is defined on the design ports, while the internal host scan interface is defined on the internal pins to be connected to the Ijtag network. An example of this is shown in Example 2 and Figure 13-5, Existing Compliance Enable Modules. In this case, the host_interface property should refer to the internal host scan interface.

- DesignInstance(*instance_name*) {client_interface : *name* ;}

  A required wrapper that specifies a design instance with a scan interface that is to be connected to the controller when the IJTAG network is inserted prior to the controller generation. The ID of the wrapper is an *instance_name* of a module with matching ICL description or "." for the current design.

  If there are multiple client scan interfaces in the specified instance, point to a single client using the client_interface property.

  This wrapper cannot be specified with the host_interface property.

- protocol : direct_memory_access | cpu_interface ;

  An optional property that specifies the access controller type. The default is direct_memory_access.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- data_width : *integer* ;

  A required property that specifies a size or a width. The property specifies the word size of the memory for a direct memory access controller. The property specifies system data bus width for CpuInterface controller types. The default value is 8.

- parent_instance : *name* ;

  An optional property that specifies the parent instance *name* of the IST controller. The default is an empty string, “”. The controller is inserted at the top level of the design when this property is not specified.

- leaf_instance_name : *name* ;

  An optional property that specifies the instance *name* of the IST controller. The default for *name* is “*designname_passID*_tessent_in_system_test_*ctrlID*_inst”.

- async_reset_all_registers : on | <u>off</u> ;

  An optional property that specifies whether all registers are reset using an asynchronous reset input port. By default (off), only a subset of registers that are functionally required to be reset (such as the FSM state register) have asynchronous reset. Set this property to on to ensure all registers in the IST controller have an asynchronous reset; this may be required to conform to certain RTL coding guidelines that require all registers to have asynchronous control.

- use_enable_sync_cell: on | <u>off</u> ;

  An optional property that specifies whether the tool adds a sync cell for the IST enable input. The default value is off, which specifies to not add the sync cell.

  If you cannot ensure that the ist_enable signal changes synchronous to its ist_clock, you can add a sync cell. This avoids a potential glitch on the to_ijtag_tck output because the circuitry uses ist_enable for the select input of this clock mux.

  If you choose to add a sync cell, your circuitry must comply with the following requirements:

  - Keep the Ijtag reset signal inactive for InSystemTest mode. (This is the ijtag_reset signal for IjtagScanInterface and the trst signal for TapScanInterface.)
  - Requires two extra clock pulses of ist_clock when entering InSystemTest mode.
  - When using libcells, you must use a sync_cell with async reset. The alternative is to generate an RTL sync cell. This is required even with DftSpecification/ use_synchronizer_cell_with_reset: off

- tap_reset_method : <u>asynchronous</u> | synchronous ;

  An optional property that specifies if the tool should apply an asynchronous reset to TAP controllers (TAPs) by activating their TRST port. Synchronous reset involves TMS and other signals. The default is asynchronous.

  An asynchronous reset gives the IST controller a simple way to reset secondary controllers when multiple TAPs are connected in a test network using a primary/secondary architecture.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TRST ports are optional, and the tool can only apply synchronous reset if the port is not present. In that case, the IST controller can reset primary TAPs by holding TMS active for five TCK cycles and introspect the test network to generate a signal sequence to reset the secondary TAPs.

- ControllerChain/present : on | off ;

  An optional property that specifies whether to enable or disable Controller Chain Mode. The default is off. When turned on, the tool creates an RTL scan chain segment within the IP to detect faults within the In-System Test controller using separate controller chain mode ATPG patterns.

- ControllerChain/clock : *enum* ;

  An optional property that specifies whether the tool uses the tck, ccm_clock, or ist_clock during Controller Chain Mode. The tck and ist_clock options indicate that the controller reuses the existing tck or ist_clock inputs, respectively. When set to ccm_clock, the tool creates a new port to connect externally to generate Controller Chain Mode ATPG patterns. The default is ccm_clock.

- ControllerChain/segment_per_instrument : on | off ;

  An optional switch and literal pair that specifies whether, during IP insertion, the tool automatically connects the controller scan chain segment in each In-System Test controller as a separate scan chain, or leaves the segments unconnected. The default is on (leave segments unconnected).

  This option cannot be set to off when an integer value is set for the max_segment _length property.

- ControllerChain/max_segment_length : int | unlimited ;

  This property specifies the maximum controller scan chain segment length in Controller Chain Mode. The minimum segment length is 32. The default is unlimited, meaning a single scan chain segment for each instrument. When the total number of scan cells is fewer than the specified length, the tool creates a single chain.

- TestProgramGroup/pattern: *id* ;

  An optional property that specifies the Patterns wrapper used by the IST controller.

Examples

The following example generates a controller attached to a TAP controller generated in the same session as the IST controller.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(piccpu,rtl) {
    IjtagNetwork {
        HostScanInterface(dft) {
            Tap(t1) {
                HostIjtag(i1) {
                    Sib(sri) {
                        ...
                    }
                    Sib(sti) {
                        ...
                    }
                }
            }
        }
    }

    MemoryBist {
        ijtag_host_interface : Sib(mbist) ;
        ...
    }
    OCC {
        ijtag_host_interface : Sib(occ) ;
        ...
    }

    InSystemTest {
        Controller(c0) {
            protocol : cpu_interface ;
            data_width: 32 ;
            host_interface : HostScanInterface(dft) ;
            Connections {
                reset : istb/reset ;
                CpuInterface {
                    Generic {
                        clock : istb/clock ;
                        enable : istb/enable ;
                        write_en : istb/write_en ;
                        data_in : istb/data_out ;
                        data_out : istb/data_in ;
                    }
                }
            }
        }
    }
}
```

The following dofile snippet example inserts three IST controllers. These controllers drive the IJTAG scan interface of the design instances m8051_A, m8051_B, and piccpu. The design instances are either sub-blocks or physical blocks on which the IJTAG network was previously inserted.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
... }
    InSystemTest {
        Controller(m8051_A) {
            protocol: cpu_interface;
            DesignInstance(m8051_A) {
            }
            data_width: 8;
            Connections {
                reset: istb/reset;
                CpuInterface {
                    Generic {
                        clock: istb/clock;
                        enable: istb/en1;
                        write_en: istb/write_en;
                        data_in: istb/data_out[7:0];
                        data_out: istb/data_in1;
                    }
                }
            }
        }

        Controller(m8051_B) {
            protocol: cpu_interface;
            DesignInstance(m8051_B) {
                client_interface: C0;
            }
            data_width: 10;
            Connections {
                reset: istb/reset;
                CpuInterface {
                    Generic {
                        clock: istb/clock;
                        enable: istb/en2;
                        write_en: istb/write_en;
                        data_in: istb/data_out[9:0];
                        data_out: istb/data_in2;
                    }
                }
            }
        }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
            Controller(piccpu) {
                protocol: cpu_interface;
                DesignInstance(piccpu) {
                }
                data_width: 12;
                Connections {
                    reset: istb/reset;
                    CpuInterface {
                        Generic {
                            clock: istb/clock;
                            enable: istb/en3;
                            write_en: istb/write_en;
                            data_in: istb/data_out;
                            data_out: istb/data_in3;
                        }
                    }
                }
            }
        }
    }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DirectMemoryAccessOptions

An optional wrapper that specifies properties related to the direct memory access IST controller.

Usage

```
DftSpecification (module_name, id) {
  InSystemTest {
    Controller(id) {
      DirectMemoryAccessOptions {
        begin_address          : integer ; // default: 0
        end_address            : integer ; // default:
                                           // max addressable memory
        address_width          : integer ; // required
        max_wait_cycles        : integer ; // default: 1 million (20 bits)
        max_test_program_count : integer ; // default: 1
        test_program_execution : single | consecutive ;
        misr                   : none | integer ; // in range [2,32]
        ijtag_clock_control    : test_active | ist_enable ;
      }
    }
  }
}
```

Description

An optional wrapper that groups properties related to the direct memory access controller.

The begin_address and end_address arguments specify the memory space that is utilized for in-system test when sharing with a functional memory. The controller address counter is set to begin_address when the controller is reset. The tool generates an error message during pattern generation if the entire program does not fit in memory between the begin_address and end_address.

Arguments

- begin_address : *int*eger ;

  An optional property used to specify the begin address for the memory block. The default is 0.

- end_address : *integer* ;

  An optional property used to specify the end address for the memory block. The default is the max addressable memory size.

- address_width : *integer* ;

  A required property used to specify the design memory width.

- max_wait_cycles : *integer* ;

  An optional property used to specify the max wait cycles. The default is 1 million cycles, which corresponds to a 20-bit wait register.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- max_test_program_count : *integer* ;

  An optional property that specifies the maximum test program count. The default is 1.

  When test_program_execution is consecutive, the following applies:

  - When *int* is a power-of-2 value, the tool sizes the fail_flag port to *int* bits. For example, when *int*=8, you must specify eight connection port/pins for the fail_flag port.
  - When *int* is not a power-of-2 value, the tool sizes the fail_flag port to the next higher power-of-2 number of bits. For example, when *int*=5, The tool sizes the fail_flag port to eight bits, although you can specify either five or eight connection port/pins for the fail_flag port. When you specify five port/pins, the tool does not connect the higher-order three bits of fail_flag.

- test_program_execution : single | consecutive ;

  An optional property that specifies whether to build an IST controller that can execute multiple test programs consecutively. The default value is "single," which builds a controller that executes only one test program at a time. The other legal value is "consecutive," which enables the capability of running multiple test programs back-to-back. For more information, see Example 1.

- misr : none | *integer* ;

  An optional property that indicates whether to create a MISR and specifies its size. Specify an integer value in the range of 2 to 32 for the tool to create a MISR of that size. It uses the corresponding Interface/DirectMemoryAccess/misr and Connections/DirectMemoryAccess/misr properties to configure the MISR output ports. You determine the required MISR size during functional safety analyses under ISO 26262 or similar standards. By default, the tool does not create a MISR.

  The IST controller provides flags to monitor test fails. For automotive and safety-critical applications, you can also add a MISR to the controller to compact the test responses from the IJTAG network. The fail flags are still available after adding a MISR. This means that MISR aliasing is not a concern, and you can use a small MISR if area is a concern.

  The MISR provides another test point for safety from a fail flag's potential single point of failure. It captures the shift-out values from the IJTAG network (TDO at the top level or ijtag_so at the block level) and creates a compacted signature per pattern. This signature does not provide any extra information for diagnosis. It supports IJTAG instruments such as LBIST, MBIST, and third-party instruments. You can use the MISR signature in a simulation testbench or access it at the end of the test to detect a failure.

- ijtag_clock_control : test_active | ist_enable ;

  An optional property that specifies which select signal the IST controller uses for the to_ijtag_tck mux. The default is ist_enable, which means that the output clock does not change to input ijtag_tck when the test is complete, but rather when ist_enable de-asserts.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When you specify the test_active value to build an IST controller, the test_active signal controls the mux, and therefore the output clock changes back from the controller clock to the input ijtag_tck when the test is complete.

Examples

Example 1

The following example demonstrates how to specify an IST controller that can run multiple test programs consecutively:

```
DftSpecification(design, id) {
   IjtagNetwork {
     HostScanInterface(tap) {
       ...
     }
   }
   InSystemTest {
     Controller(c0) {
       host_interface: HostScanInterface(tap);
       data_width: 8;
       DirectMemoryAccessOptions {
         address_width: 16;
         max_wait_cycles: 10000;
         max_test_program_count: 4;
         test_program_execution: consecutive;
       }
       Connections {
         reset: power_on_reset/out;
         DirectMemoryAccess {
           enable: ist_bridge/en;
           clock: ist_bridge/clk;
           fail_flag: ist_bridge/instrument_fail[0:3];
           test_program_done: ist_bridge/done;
           test_program_start_index: ist_bridge/start_index[1:0];
           test_program_end_index: ist_bridge/end_index[1:0];
           invalid_test_program_index: dma_invalid_test_program_index;
           mem_data: rom/data;
           mem_address: rom/address;
         }
       }
     }
   }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example demonstrates how to specify an IST controller that includes a MISR for compacting test responses:

```
DftSpecification(design, id) {
  IjtagNetwork {
    HostScanInterface(tap) {
      ..
    }
  }
  InSystemTest {
    Controller(1) {
      protocol: direct_memory_access;
      host_interface: HostScanInterface(tap);
      DirectMemoryAccessOptions {
        misr: 12;
        max_test_program_count: 4;
      }
      Connections {
        reset: RST_g/power_on_reset;
        DirectMemoryAccess {
          clock: ist_bridge/clock;
          test_program_index: ist_bridge/index[1:0];
          fail_flag: ist_bridge/fail_h;
          misr: ist_bridge/misr_resp[11:0];
          enable: ist_bridge/post_en;
          mem_address: ist_rom/address;
          mem_data: ist_rom/data;
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Interface

An optional wrapper that specifies the interface port names for the In-System Test (IST) controller.

Usage

```
DftSpecification (module_name, id) {
  InSystemTest {
    Controller(id) {
      Interface {
        reset          : port_name ;     // default: reset
        reset_polarity: active_high | active_low ;
        test_active    : port_name ;     // default: test_active
        ControllerChain {
           enable          : port_name; // default: ccm_en
           scan_in         : port_name; // default: ccm_scan_in
           scan_out        : port_name; // default: ccm_scan_out
           scan_en         : port_name; // default: scan_en
           clock           : port_name; // default: ccm_clock
        }
        CpuInterface {
          Generic {
          }
          ApbSecondary {
          }
        }
        DirectMemoryAccess {
        }
        TapScanInterface {
        }
        IjtagScanInterface {
        }
      }
    }
  }
}
```

Description

The TapScanInterface tck, trst, tms, tdi, tdo, and tdo_en ports specify the client side of the controller connected to the design source pins (that is, the pads). Ports to_tck, to_trst, to_tms, to_tdi, from_tdo, and from_tdo_en specify the host side of the controller connected to the instrument interface, such as TAP controller pins.

Within the Interface wrapper, use the ControllerChain wrapper to define the ports for Controller Chain Mode.

The CpuInterface wrapper is an optional wrapper that defines interface port names for a generic CPU-based access controller or an APB-based controller. Use a Generic wrapper for a CPU-based controller. Use an ApbSecondary wrapper to get an APB-based controller.

The ApbSecondary wrapper produces an IST controller implementing the Advanced Peripheral Bus (APB) protocol. APB is part of the Arm® AMBA® protocol family.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- reset : *port_name* ;

  An optional property that defines the asynchronous reset input port of the IST controller.

- reset_polarity : active_high | active_low ;

  An optional property that defines the polarity of the reset input port. A reset polarity of active_high applies the reset. A reset polarity of active_low does not apply the reset. The default is active_low.

- test_active : *port_name* ;

  An optional property that defines the test active output port name. The default *port_name* is test_active. This property is typically used to make connections when test mode is running. For example, use the test_active property to configure the memory to be accessible by the IST controller during test.

- ControllerChain/enable : *port_name* ;

  An optional property that specifies the name of the controller chain enable port. This port is only used when DftSpecification/InSystemTest/Controller/ControllerChain/present is set to on. The default is ccm_en.

- ControllerChain/scan_in : *port_name* ;

  An optional property that specifies the name of the controller chain scan in port. This port is only used when DftSpecification/InSystemTest/Controller/ControllerChain/present is set to on. The default is ccm_scan_in.

- ControllerChain/scan_out : *port_name* ;

  An optional property that specifies the name of the controller chain scan out port. This port is only used when DftSpecification/InSystemTest/Controller/ControllerChain/present is set to on. The default is ccm_scan_out.

- ControllerChain/scan_en : *port_name* ;

  An optional property that specifies the name of the scan enable port. This port is only used when DftSpecification/InSystemTest/Controller/ControllerChain/present is set to on. The default is scan_en.

- ControllerChain/clock : *port_name* ;

  An optional property that specifies the name of the controller chain clock port. This port is only used when DftSpecification/InSystemTest/Controller/ControllerChain/present is set to on, and the controller chain clock is set to ccm_clock. The default is ccm_clock.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Generic

A required wrapper within the CpuInterface wrapper that specifies the required input and output port names for a generic CPU-based In-System Test (IST) controller.

Usage

```
DftSpecification (module_name, id) {
  InSystemTest {
    Controller(id) {
      Interface {
        CpuInterface {
          Generic {
            clock   : port_name ; // default: cpu_interface_clock
            data_in : port_name ; // default: data_from_cpu[%d]
            data_out: port_name ; // default: data_to_cpu[%d]
            write_en: port_name ; // default: write_en_from_cpu
            enable  : port_name ; // default: cpu_interface_en
          }
        }
      }
    }
  }
}
```

Description

The Generic wrapper specifies an In-System Test controller implementing a generic CPU interface.

Arguments

- clock : *port_name* ;

  A required property that specifies the clock input port name. The CPU native clock and this clock may be asynchronous. This clock drives the connected IJTAG network in in-system test mode. The default is cpu_interface_clock.

- data_in : *port_name* ;

  A required property that specifies the name of the parallel input bus port name. This bus contains instruction opcodes for the IST controller, and data to write to the IJTAG network through the TAP controller. The two most significant bits represent an opcode for the IST controller, and the remaining bits are the write data to shift into the network. The default is data_from_cpu[%d], where [%d] is a literal string that expands based on the data_width argument in the DftSpecification/InSystemTest wrapper.

- data_out : *port_name* ;

  A required property that specifies the name of the parallel output bus port name. This bus contains the status of the controller read operation and the data from the IJTAG network. The two most significant bits represent a status code, and the remaining bits are the data read from the network. The default is data_to_cpu[%d], where [%d] is a literal string that expands based on the data_width argument in the DftSpecification/InSystemTest wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- write_en : *port_name* ;

  A required property that specifies the write enable input port name. This input signal indicates valid data is present in the DataIn bus and is to be asserted by the CPU after writing to DataIn. The default is write_en_from_cpu.

- enable : *port_name* ;

  A required property that specifies the name of the IST controller enable input port name. This port is the input signal used to enable In-System Test mode. This signal should be asserted and stay asserted during the entire test. This signal provides multiplexed access to the IJTAG network inputs from the controller. The default is cpu_interface_en.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ApbSecondary

A required wrapper within the CpuInterface wrapper that specifies the input and output port names for an Advanced Peripheral Bus (APB) based In-System Test (IST) controller.

Usage

```
DftSpecification (module_name, id) {
  InSystemTest {
    Controller(id) {
      Interface {
        CpuInterface {
          ApbSecondary {
            clock    : port_name ; // default PCLK
            select   : port_name ; // default PSEL
            enable   : port_name ; // default PENABLE
            write_en : port_name ; // default PWRITE
            data_in  : port_name ; // default PWDATA[%d]
            data_out : port_name ; // default PRDATA[%d]
            ready    : port_name ; // default PREADY
          }
        }
      }
    }
  }
}
```

Description

The ApbSecondary wrapper specifies an IST controller implementing the APB protocol. APB is part of the Arm® AMBA® protocol family.

Arguments

- clock : *port_name* ;

  An optional property that specifies the clock input port. The default port name is PCLK. It is the clock for all connections on the bus. APB transactions are synchronous to the rising edge of this clock.

- select : *port_name* ;

  An optional property that specifies the name of the select input port. The default port name is PSEL. The APB bridge address decoder generates the select signal. A data transfer is required when selected.

- enable : *port_name* ;

  An optional property that specifies the name of the enable input port. The default port name is PENABLE. The enable signal is used in conjunction with the select signal to enable write and read transactions. It indicates the second and subsequent cycles of an APB data transfer.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- write_en : *port_name* ;

  An optional property that specifies the name of the write_enable input port. The default port name is PWRITE. The signal indicates the direction of data transfer. A high value indicates APB write access. A low value indicates APB read access.

- data_in : *port_name* ;

  An optional property that specifies the name of the data input port. The default port name is PWDATA[%d], where [%d] is a literal string that expands based on the data_width argument in the DftSpecification/InSystemTest wrapper. The range of values for data_width is 8 to 32 bits as defined by the APB specification. This bus contains the controller instruction opcode and data to write to the IJTAG network. The data_width is the same for both read and write data.

- data_out : *port_name* ;

  An optional property that specifies the name of the data output port. The default port name is PRDATA[%d], where [%d] is a literal string that expands based on the data_width argument in the DftSpecification/InSystemTest wrapper. The range of values for data_width is 8 to 32 bits as defined by the APB specification. This bus contains the data read from the IJTAG network, but the two most significant bits are unused. The data_width is the same for both read and write data.

- ready : *port_name* ;

  An optional property that specifies the name of the ready output port. The default port name is PREADY. The controller signals transaction status using the ready port, so the APB bridge knows when the write/read transaction ends. The ready signal is low during an IST controller write transaction until it writes the provided data to the IJTAG network. The number of wait cycles is equal to the data_width times the clock divider ratio cycles of PCLK. Read transactions are immediate and are not blocked because the previous write has already shifted out the read data.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DirectMemoryAccess

An optional wrapper defined within the Interface wrapper that groups properties related to the direct memory access IST controller.

Usage

```
DftSpecification (module_name, id) {
  InSystemTest {
    Controller(id) {
      Interface {
        DirectMemoryAccess {
          clock              : port_name ; // default: dma_clock
          test_program_index: port_name ; // default:
                                          // dma_test_program_index[%d]
          test_program_start_index: port_name ; // default:
                                          // dma_test_program_start_index[%d]
          test_program_end_index: port_name ; // default:
                                           // dma_test_program_end_index[%d]
          invalid_test_program_index : port_name // default: -
          test_program_done : port_name ; // default:
                                          // dma_test_program_done
          fail_flag          : port_name ; // default: dma_fail_flag[%d]
          misr               : port_name ; // default: dma_misr[%d]
          mem_address        : port_name ; // default: dma_address[%d]
          mem_data           : port_name ; // default: dma_data[%d]
          enable             : port_name ; // default: dma_en
        }
      }
    }
  }
}
```

Description

The DirectMemoryAccess wrapper is an optional wrapper defined within the Interface wrapper that groups properties related to the direct memory access controller. This wrapper is applied when using the direct memory access protocol.

Arguments

- clock : *port_name* ;

  An optional property that specifies the name of the controller or direct memory access controller clock input port. The default *port_name* is dma_clock.

- test_program_index : *port_name* ;

  An optional property that specifies the name of the test program index input port. The default *port_name* is dma_test_program[%d], where %d is a literal string that is expanded to the range of the vector during execution. Use this property only when DirectMemoryAccessOptions/test_program_execution is single.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- test_program_start_index : *port_name* ;

  An optional property that specifies the name of the input port for the test program start index. The default *port_name* is dma_test_program_start_index[%d], where %d is a literal string that is expanded to the range of the vector during execution. Use this property only when DirectMemoryAccessOptions/test_program_execution is consecutive.

- test_program_end_index : *port_name* ;

  An optional property that specifies the name of the input port for the test program end index. The default *port_name* is dma_test_program_end_index[%d], where %d is a literal string that is expanded to the range of the vector during execution. Use this property only when DirectMemoryAccessOptions/test_program_execution is consecutive.

- invalid_test_program_index : *port_name*;

  A property that specifies the name of an output port that indicates when the selected test program index is out of range of valid test programs. This signal is cleared when the FSM goes to the init state.

- test_program_done : *port_name* ;

  An optional property that specifies the name of the test program done output port. The default *port_name* is dma_test_program_done.

- fail_flag : *port_name* ;

  An optional property that specifies the name of the fail flag output port. The default *port_name* is dma_fail_flag. Use this port to determine test status. The signal indicates when mismatches between read and expect data has occurred. This signal is cleared when the FSM goes to the init state.

- misr : *port_name* ;

  An optional property that specifies the name of the MISR output port. The default *port_name* is dma_misr[%d], where %d is a literal string that is expanded to the range of the MISR size, as specified in the DirectMemoryAccessOptions/misr property. This signal is reset to “0” at the start of each test program.

- mem_address : *port_name* ;

  An optional property that specifies the name of the memory address output port. The default *port_name* is “dma_address[%d]”, where %d is a literal string that is expanded to the range of the vector. This bus provides the address for the next memory read. When memory is shared with functional logic, ensure the controller has exclusive access to the memory address bus.

- mem_data : *port_name* ;

  An optional property that specifies the name of the memory data input port. The default *port_name* is “dma_data[%d]”, where %d is a literal string that is expanded to the range of the vector during processing. Use this input bus to provide the data corresponding to the current address.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- enable : *port_name*;

  An optional property that specifies the name of the enable input port. The default *port_name* is dma_en. Use this port to start the in-system test. The enable must stay active for the entire duration of the test. This signal can be de-asserted at anytime to stop a running test.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TapScanInterface

An optional wrapper defined within the Interface wrapper that specifies the interface port names when the IST controller uses a TAP controller to execute tests.

Usage

```
DftSpecification (module_name, id) {
  InSystemTest {
    Controller(id) {
      Interface {
        TapScanInterface {
          tck        : port_name ; // default: ijtag_tck
          trst       : port_name ; // default: trst
          tms        : port_name ; // default: tms
          tdi        : port_name ; // default: tdi
          tdo        : port_name ; // default: tdo
          tdo_en     : port_name ; // default: tdo_en
          to_tck     : port_name ; // default: to_ijtag_tck
          to_trst    : port_name ; // default: to_trst
          to_tms     : port_name ; // default: to_tms
          to_tdi     : port_name ; // default: to_tdi
          from_tdo   : port_name ; // default: from_tdo
          from_tdo_en: port_name ; // default: from_tdo_en
        }
      }
    }
  }
}
```

Description

The TapScanInterface wrapper defined within the Interface wrapper is an optional wrapper that specifies the interface port names when the IST controller uses a TAP controller to execute tests.

The top-level TAP signals are tck, trst, tms, tdi, and tdo. These signals are brought into the controller for multiplexing with the controller generated signals. The output signals from the controller are to_tck, to_trst, to_tms, to_tdi, and from_tdo.

Arguments

- tck : *port_name* ;

  An optional property that specifies the name of the test clock input port. The default *port_name* is ijtag_tck.

- trst : *port_name* ;

  An optional property that specifies the name of the test reset input port. The default *port_name* is trst.

- tms : *port_name* ;

  An optional property that specifies the name of the tms input port. The default *port_name* is tms.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- tdi : *port_name* ;

  An optional property that specifies the name of the tdi input port. The default *port_name* is tdi.

- tdo : *port_name* ;

  An optional property that specifies the name of the tdo input port. The default *port_name* is tdo.

- tdo_en : *port_name* ;

  An optional property that specifies the name of the tdo enable input port. The default *port_name* is tdo_en.

- to_tck : *port_name* ;

  An optional property that specifies the name of the output clock port. The default *port_name* is to_ijtag_tck. This property handles the output signal from the IST controller. When the controller is active the clock input of the controller drives this port, otherwise the tck input drives this port.

- to_trst : *port_name* ;

  An optional property that specifies the name of the output test reset port. The default *port_name* is to_trst. This property handles the output signal from the IST controller that is held inactive when the controller is running, otherwise it is driven by the top-level trst port.

- to_tms : *port_name* ;

  An optional property that specifies the name of the output tms port. The default *port_name* is to_tms. This property handles the output signal from the IST controller that is multiplexed with the top-level tms port at the input of the TAP controller.

- to_tdi : *port_name* ;

  An optional property that specifies the name of the output tdi port. The default *port_name* is to_tdi. This property handles the output signal from the IST controller that is multiplexed with the top-level tdi port at the source of the TAP controller tdi input.

- from_tdo : *port_name* ;

  An optional property that specifies the name of the from tdo output port. The default *port_name* is from_tdo. This property handles the input signal to the IST controller that comes from the internal tdo signal (before the tristate buffer) of the TAP controller.

- from_tdo_en : *port_name* ;

  An optional property that specifies the name of the output tdo enable port. The default *port_name* is from_tdo_en.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

IjtagScanInterface

An optional wrapper defined within the Interface wrapper that specifies the interface port names when the IST controller uses an IJTAG scan interface to execute tests.

Usage

```
DftSpecification (module_name, id) {
  InSystemTest {
    Controller(id) {
      Interface {
        IjtagScanInterface {
          tck          : port_name ; // default: ijtag_tck
          reset        : port_name ; // default: ijtag_reset
          select       : port_name ; // default: ijtag_sel
          capture_en   : port_name ; // default: ijtag_ce
          shift_en     : port_name ; // default: ijtag_se
          update_en    : port_name ; // default: ijtag_ue
          scan_in      : port_name ; // default: ijtag_si
          scan_out     : port_name ; // default: ijtag_so

          to_tck       : port_name ; // default: to_ijtag_tck
          to_reset     : port_name ; // default: to_ijtag_reset
          to_select    : port_name ; // default: to_ijtag_sel
          to_capture_en: port_name ; // default: to_ijtag_ce
          to_shift_en  : port_name ; // default: to_ijtag_se
          to_update_en : port_name ; // default: to_ijtag_ue
          to_scan_in   : port_name ; // default: to_ijtag_si
          from_scan_out: port_name ; // default: from_ijtag_so
        }
      }
    }
  }
}
```

Description

The IjtagScanInterface wrapper is an optional wrapper defined within the Interface wrapper that specifies the interface port names when the IST controller uses an IJTAG scan interface.

Arguments

- tck : *port_name*;

  An optional property that specifies the name of the test clock port. The default *port_name* is ijtag_tck.

- reset : *port_name*;

  An optional property that specifies the name of the reset port. The default *port_name* is ijtag_reset.

- select : *port_name*;

  An optional property that specifies the name of the select port. The default *port_name* is ijtag_sel.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- capture_en : *port_name*;

  An optional property that specifies the name of the enable capture port. The default *port_name* is ijtag_ce.

- shift_en : *port_name*;

  An optional property that specifies the name of the enable shift port. The default *port_name* is ijtag_se.

- update_en : *port_name*;

  An optional property that specifies the name of the enable update port. The default *port_name* is ijtag_ue.

- scan_in : *port_name*;

  An optional property that specifies the name of the scan in port. The default *port_name* is ijtag_si.

- scan_out : *port_name*;

  An optional property that specifies the name of the scan out port. The default *port_name* is ijtag_so.

- to_tck : *port_name*;

  An optional property that specifies the name of the output clock port. The default *port_name* is to_ijtag_tck.

- to_reset : *port_name*;

  An optional property that specifies the name of the output reset port. The default *port_name* is to_ijtag_reset.

- to_select : *port_name*;

  An optional property that specifies the name of the select output port. The default *port_name* is to_ijtag_sel.

- to_capture_en : *port_name*;

  An optional property that specifies the name of the output capture enable port. The default *port_name* is to_ijtag_ce.

- to_shift_en : *port_name*;

  An optional property that specifies the name of the output shift enable port. The default *port_name* is to_ijtag_se.

- to_update_en : *port_name*;

  An optional property that specifies the name of the output enable update port. The default *port_name* is to_ijtag_ue.

- to_scan_in : *port_name*;

  An optional property that specifies the name of the output scan in port. The default *port_name* is to_ijtag_si.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- from_scan_out : *port_name*;

  An optional property that specifies the name of the from scan output port. The default *port_name* is from_ijtag_so.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Connections

An optional wrapper that specifies the design node used to connect the corresponding In-System Test (IST) controller.

Usage

```
DftSpecification (module_name, id) {
  InSystemTest {
    Controller(id) {
      Connections {
        reset          : port_pin_name ; // required
        test_active    : port_pin_name ; // default: "" (no connection)
        controller_chain_enable    : port_pin_name ;
                     // default: OptionalDftSignal(controller_chain_mode)
        controller_chain_clock     : port_pin_name ;
                              // default: OptionalDftSignal(test_clock)
        scan_en                    : port_pin_name ;
                              // default: OptionalDftSignal(scan_en)
        controller_chain_scan_in   : port_pin_name ;
                              // default: control_chain_%s_scan_in
        controller_chain_scan_out : port_pin_name ;
                              // default: control_chain_%s_scan_out
        CpuInterface {                  // required
          Generic {
          }
          ApbSecondary {
          }
        }
        DirectMemoryAccess {            // required
        }
      }
    }
  }
}
```

Description

Connections do not exist for the TapScanInterface or IjtagScanInterface intercept pins. The intercept pins are connected to the location defined by host_interface or DesignInstance() arguments.

The CpuInterface wrapper is an optional wrapper that defines design node connections for a generic CPU-based access controller or an APB-based controller. Use a Generic wrapper for a generic CPU-based controller. Use an ApbSecondary wrapper to get an APB-based controller.

The ApbSecondary wrapper produces an IST controller implementing the Advanced Peripheral Bus (APB) protocol. APB is part of the Arm® AMBA® protocol family.

When design nodes for the wrapper do not exist or you do not want to make connections, specify “0” for the input port or pin name and empty double quotes “” representing an empty string for the output port or pin name. This includes the reset and test_active properties, as well as the properties defined within the CpuInterface and DirectMemoryAccess wrappers.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Pin names should be existing pin names from the design. Port names can be any name, but must be the correct direction.

For Controller Chain Mode connections, you can connect one In-System Test controller into one scan chain with the scan_in/scan_out pins. For more complex scenarios, such as merging with other LogicBIST instruments, enable the ControllerChain/segment_per_instrument property and use the scan insertion tool.

Arguments

- reset : *port_pin_name* ;

  A required property that specifies the reset input port or pin name. This property specifies the asynchronous input, for example a design's reset or poweron_reset.

- test_active : *port_pin_name*;

  An optional property that specifies the test active output port or pin name. The default is an empty string, "", if there is no connection.

- controller_chain_enable : *port_pin_name* ;

  An optional property used to specify the controller chain enable pin name.

- controller_chain_clock : *port_pin_name* ;

  An optional property used to specify the controller chain clock pin name. This connection is used only when the Controller/ControllerChain/Clock property is set to ccm_clock

- scan_en : *port_pin_name* ;

  An optional property used to specify the scan enable pin name.

- controller_chain_scan_in : *port_pin_name* ;

  An optional property used to specify the controller chain scan in pin name. This connection is only used when ControllerChain/segment_per_instrument is off. The "%s" in the name resolves to the controller ID so a name pattern can be specified.

  When segment_per_instrument is on, this connection value is not used. Instead, the controller_chain_scan_in is tied to 0 during IP insertion; scan insertion later connects this pin to controller chain mode scan chains.

- controller_chain_scan_out : *port_pin_name* ;

  An optional property used to specify the controller chain scan out pin name. This connection is only used when ControllerChain/segment_per_instrument is off. The "%s" in the name resolves to the controller ID so a name pattern can be specified.

  When segment_per_instrument is on, this connection value is not used. Instead, the controller_chain_scan_out is left unconnected during IP insertion; scan insertion later connects this pin to controller chain mode scan chains.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Generic

A required wrapper within the CpuInterface wrapper that defines design node connections for a generic CPU-based In-System Test (IST) controller.

Usage

```
DftSpecification (module_name, id) {
  InSystemTest {
    Controller(id) {
      Connections {
        CpuInterface {
          Generic {
            clock   : port_pin_name ; // required input
            data_in : port_pin_name ; // required input
            data_out: port_pin_name ; // required output
            write_en: port_pin_name ; // required input
            enable  : port_pin_name ; // required input
          }
        }
      }
    }
  }
}
```

Description

The Generic wrapper connects an In-System Test controller to a generic CPU interface.

When design nodes for the wrapper do not exist, or you do not want to make connections, specify “0” for the input ***port_pin_name*** and empty string “” for the output ***port_pin_name***.

Arguments

- clock : *port_pin_name* ;

  A required property that specifies the design port or pin to connect to the IST controller clock port.

- data_in : *port_pin_name* ;

  A required property that specifies the design port or pin to connect to the IST controller data_in port.

- data_out : *port_pin_name* ;

  A required property that specifies the design port or pin to connect to the IST controller data_out port.

- write_en : *port_pin_name* ;

  A required property that specifies the design port or pin to connect to the IST controller write_en port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- enable : *port_pin_name* ;

  A required property that specifies the design port or pin to connect to the IST controller enable port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ApbSecondary

A required wrapper within the CpuInterface wrapper that defines design node connections for an Advanced Peripheral Bus (APB) based In-System Test (IST) controller.

Usage

```
DftSpecification (module_name, id) {
  InSystemTest {
    Controller(id) {
      Connections {
        CpuInterface {
          ApbSecondary {
            clock    : port_pin_name ; // required input
            select   : port_pin_name ; // required input
            enable   : port_pin_name ; // required input
            write_en : port_pin_name ; // required input
            data_in  : port_pin_name ; // required input
            data_out : port_pin_name ; // required output
            ready    : port_pin_name ; // required output
          }
        }
      }
    }
  }
}
```

Description

The ApbSecondary wrapper connects an In-System Test controller to an Advanced Peripheral Bus. APB is part of the Arm® AMBA® protocol family.

Arguments

- clock : *port_pin_name* ;

  A required property that specifies the design port or pin to connect to the IST controller clock input port.

- select : *port_pin_name* ;

  A required property that specifies the design port or pin to connect to the IST controller select input port.

- enable : *port_pin_name* ;

  A required property that specifies the design port or pin to connect to the IST controller enable input port.

- write_en : *port_pin_name* ;

  A required property that specifies the design port or pin to connect to the IST controller write_en input port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- data_in : *port_pin_name* ;

  A required property that specifies the design port or pin to connect to the IST controller data_in input port.

- data_out : *port_pin_name* ;

  A required property that specifies the design port or pin to connect to the IST controller data_out output port.

- ready : *port_pin_name* ;

  A required property that specifies the design port or pin to connect to the IST controller ready output port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DirectMemoryAccess

A required wrapper defined within the Connections wrapper that groups properties related to the direct memory access IST controller.

Usage

```
DftSpecification (module_name, id) {
  InSystemTest {
    Controller(id) {
      Connections {
        DirectMemoryAccess {
          clock                : port_pin_name ; // required
          test_program_index : port_pin_name ; // required
          test_program_start_index : port_pin_name, ... ; // required
          test_program_end_index   : port_pin_name, ... ; // required
          invalid_test_program_index : port_pin_name ;    // required
          test_program_done : port_pin_name ; // required
          fail_flag         : port_pin_name, ... ; // required
          misr              : port_pin_name, ... ; // required
          mem_address       : port_pin_name ; // required
          mem_data          : port_pin_name ; // required
          enable            : port_pin_name ; // required
        }
      }
    }
  }
}
```

Description

The DirectMemoryAccess wrapper is a required wrapper defined within the Connections wrapper that groups properties related to the direct memory access controller.

When design nodes for the Connections wrapper do not exist or you do not want to make connections, specify “0” for the input *port_pin_name* and empty string, “”, for the output *port_pin_name*.

Arguments

- clock : *port_pin_name* ;

  A required property that specifies the input clock port or pin name for the IST controller. The CPU’s native clock is not required to be synchronous with this clock. This clock drives the connected IJTAG network in test mode.

- test_program_index : *port_pin_name* ;

  A required property that specifies the name of the test program index input port. Use this property only when DirectMemoryAccessOptions/test_program_execution is single.

- test_program_start_index : *port_pin_name*, ... ;

  An optional property that specifies the name of one or more input ports or pin names for the test program start index. Use this property only when DirectMemoryAccessOptions/test_program_execution is consecutive.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- test_program_end_index : *port_pin_name*, ... ;

  An optional property that specifies the name of one or more input ports or pin names for the test program end index. Use this property only when DirectMemoryAccessOptions/ test_program_execution is consecutive.

- invalid_test_program_index : *port_pin_name*;

  A property that specifies an output port or pin connection that indicates when the currently specified test program index is valid. This signal is cleared at the start of a test program when FSM state is init. This property is required when the controller has more than one test program (DirectMemoryAccessOptions/max_test_program_count > 1) and this port is requested (Interface/DirectMemoryAccess/invalid_test_program_index is any value other than "-"). The tool creates this port only when you provide a port name.

- test_program_done : *port_pin_name* ;

  A required property that specifies the test program done output port or pin name.

- fail_flag : *port_pin_name, ...* ;

  A required property that specifies one or more output ports or pin names for the fail flag. Use these ports to determine test status. The signal indicates when mismatches between read and expect data have occurred. This signal is cleared when the FSM goes to the init state.

- misr : *port_pin_name*, ... ;

  A required property that specifies one or more output port or pin name connections for the MISR. This signal indicates the MISR signature at the end of the test and is available when the DirectMemoryAccessOptions/misr property is specified. This signal is reset to "0" at the start of each test program

- mem_address : *port_pin_name* ;

  A required property that specifies the memory address output port or pin name. This bus provides the address for the next memory read. When memory is shared with functional logic, ensure the controller has exclusive access to the memory address bus.

- mem_data : *port_pin_name* ;

  A required property that specifies the memory data input port or pin name. This bus provides the data corresponding to the current address.

- enable : *port_pin_name* ;

  A required property that specifies the input enable port or pin name. This input enables InSystemTest mode and should be asserted and stay asserted during the entire test. You can also use this signal outside the IST controller to provide multiplexed access to the IJTAG network inputs from the controller.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

EmbeddedBoundaryScan

Specifies the creation of a boundary scan segment inside a module.

Usage

```
DftSpecification(module_name,id) {
  EmbeddedBoundaryScan {
    pad_io_ports                      : port_name, ... ;
    interface_parent_instance         : parent_instance ;
    outputs_per_enable_cell           : int ; // default: 16 *DefSpec
    default_cell_on_bidir_port        : combined | separated ;
    max_segment_length_for_logictest  : int | unlimited | off ;
    Interface  {                      // *DefSpec
    }
    ImplementationOptions  {          // *DefSpec
    }
    BoundaryScanCellOptions {
    }
    LogicalGroups {
    }
    BondingConfigurations {
    }
    AuxiliaryInputOutputPorts {
    }
    ACModeOptions {
    }
    EnableGroups {
    }
    InternalBScanCells {
    }
    InternalBScanSegments {
    }
  }
}
```

Description

A wrapper that specifies the creation of a boundary scan segment inside a module. This module can then be reused as a building block when constructing the boundary scan chain at the next level up. This feature is useful in two situations. The first usage is when you have a physical region containing pads and you want to insert the boundary-scan cells into it before laying it out. The second usage is when you want to insert the boundary-scan cells into non-unique sub-modules without having to uniquify them. You can make the sub-module the current design, insert a boundary scan segment into it, and then reuse the module more than once at the next level.

A *.tcd_bscan* file containing a BoundaryScan wrapper (see "Tessent Core Description" on page 4793) is created for the current design. Once loaded in memory, the tool automatically recognizes the existing boundary scan segments in the design and includes them in the boundary scan chain at the current level. The legacy *.lvbscan* file format is also natively supported and translated into the current BoundaryScan wrapper syntax when loaded into Tessent Shell. Refer to the set_module_matching_options and the "set_design_sources -format" commands to learn

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

how to have them automatically searched and loaded. Refer to the read_core_descriptions command to learn how to explicit load them in.

Arguments

- pad_io_ports : *port_name*, ... ;

  A property that lists all ports that are to be equipped with a boundary-scan cell. Those ports must all be directly connected to an input and/or output pad cell. The order of the list is used to order the boundary scan segment where the cell associated with the first port is closer to scanin and the cell associated with the last port is closer to scanout. You have to list each port one by one; no wildcards are allowed. However you can use the %#d[left_bit:right_bit] syntax to specify a group of numbered ports. A bused port can be specified with the order of the index following the required order of the boundary scan segment, and it does not necessarily have to be the order in which it is declared in the design. For example, the bus GPIO[7:0] can be specified as GPIO[4:7], GPIO[0:3] to specify that bit 4 is closest to scanin and is followed by bit 5 even if the bus is declared with the larger index on the left.

  This property is automatically populated by the create_dft_specification command with the port list you have specified with the set_boundary_scan_port_options -pad_io_ports option. The port list is physically ordered if you read in a DEF file using the read_def command.

- interface_parent_instance : *parent_instance* ;

  An optional property that identifies the design module in which the tool places the boundary scan interface. The default placement is at the top level if this property is not specified.

- outputs_per_enable_cell : *int* ;

  An optional property that instructs the tool to limit the number of output pins controlled by a given output enable cell. If the enable pin of the output pad cell is tied on or off, the tool is free to assign any output pad cell to any given enable cell in order to meet the outputs_per_enable_cell constraints. If, on the other hand, a group of output pad cells have their enable pin controlled by a single functional net, the tool inserts N enable boundary-scan cells and automatically separate the fanout of the enable signal such that the fanout of each enable cell is smaller than the specified outputs_per_enable_cell value. The splitting of the functional enable fanout is illustrated in the example of the LogicalGroups wrapper in Figure 13-42 and Figure 13-43.

- default_cell_on_bidir_port : combined | separated;

  An optional property that instructs the tool to use either a single boundary scan register (combined) or two independent unidirectional boundary scan registers (separated) for bidirectional ports.

- max_segment_length_for_logictest : int | unlimited | <u>off</u> ;

  An optional property that enables logic test support at the block level. The tool adds a mini-OCC and additional logic to control the clocks and the control signal of the BoundaryScan cells, and to provide access to the BoundaryScan chain, during logic test. Use the max_segment_length_for_logictest property to define the maximum segment length that the tool can use when splitting the chain into smaller segments for these tests.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When this property is set to int or unlimited, a *.ctl* file is created for each logical group module. This file describes the scan chains created from embedded Boundary Scan segments and may be used during scan chain stitching in third-party tools.

Examples

The following example defines a boundary scan chain associated with the GPIO[7:0] ports inside a module. This module is then used as a building block for the boundary scan chain at the next level up, either to construct a larger segment in a parent module or as part of the chain for a chip.

```
DftSpecification(my_block,rtl) {
  EmbeddedBoundaryScan {
    pad_io_ports : GPIO[4:7], GPIO[0:3] ;
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Interface

Specifies the naming of all ports created on the current design to provide access to the boundary scan segment.

Usage

```
DftSpecification(module_name,id) {
  EmbeddedBoundaryScan {
    Interface { *DefSpec
      select                   : port_naming ; // bscan_select
      reset                    : port_naming ; // bscan_reset
      force_disable            : port_naming ; // bscan_force_disable
      select_jtag_input        : port_naming ; // bscan_select_jtag_input
      select_jtag_output       : port_naming ; // bscan_select_jtag_output
      bscan_clock              : port_naming ; // bscan_clock
      capture_en               : port_naming ; // bscan_capture_en
      shift_en                 : port_naming ; // bscan_shift_en
      update_en                : port_naming ; // bscan_update_en
      scan_in                  : port_naming ; // bscan_scan_in
      scan_out                 : port_naming ; // bscan_scan_out
      scan_out_pipeline        : on | off;
      auxiliary_output         : port_naming ; // bscan_%s_aux_out
      auxiliary_output_enable  : port_naming ; // bscan_%s_aux_out_en
      auxiliary_input          : port_naming ; // bscan_%s_aux_in
      auxiliary_input_enable   : port_naming ; // bscan_%s_aux_in_en
      ac_init_clock0           : port_naming ; // bscan_ac_init_clk0
      ac_init_clock1           : port_naming ; // bscan_ac_init_clk1
      ac_signal                : port_naming ; // bscan_ac_signal
      ac_mode_en               : port_naming ; // bscan_ac_mode_en
    }
  }
}
```

Description

A wrapper that specifies the naming of all ports created on the current design to provide access to the boundary scan segment.

Arguments

- select : *port_naming* ;

  An optional property that defines the name of the select signal. The select signal is used to enable or disable the embedded boundary scan interface of the core. When not specified, the port is named “bscan_select”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- reset : *port_naming* ;

  An attribute used to provide the name for the port carrying the reset function. When not specified, the port is named “bscan_reset”. The reset function is active low.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- force_disable : *port_naming* ;

  An attribute used to provide the name for the port carrying the force_disable function. When not specified, the port is named “bscan_force_disable”.

- select_jtag_input : *port_naming* ;

  An attribute used to provide the name for the port carrying the select_jtag_input function. When not specified, the port is named “bscan_select_jtag_input”.

- select_jtag_output : *port_naming* ;

  An attribute used to provide the name for the port carrying the select_jtag_output function. When not specified, the port is named “bscan_select_jtag_output”.

- bscan_clock : *port_naming* ;

  An attribute used to provide the name for the port carrying the bscan_clock function. When not specified, the port is named “bscan_clock”.

- capture_en : *port_naming* ;

  An attribute used to provide the name for the port carrying the capture_en function. When not specified, the port is named “bscan_capture_en”.

- shift_en : *port_naming* ;

  An attribute used to provide the name for the port carrying the shift_en function. When not specified, the port is named “bscan_shift_en”.

- update_en : *port_naming* ;

  An attribute used to provide the name for the port carrying the update_en function. When not specified, the port is named “bscan_update_en”.

- scan_in : *port_naming* ;

  An attribute used to provide the name for the port carrying the scan_in function. When not specified, the port is named “bscan_scan_in”.

- scan_out : *port_naming* ;

  An attribute used to provide the name for the port carrying the scan_out function. When not specified, the port is named “bscan_scan_out”.

- scan_out_pipeline : on | off;

  A Boolean property to specify the insertion of a pipeline stage with no retiming element after the last boundary scan cell sourcing the scan_out port. Set it to on in a tiling flow where the boundary scan clock tree is localized to the tile, and you have a loop timing interface with the tile that sources the boundary scan clock and gets the returned scan_out data. Adding a pipeline stage with no retiming stage makes sure that the loop timing interface has the full TCK clock period from when TCK enters the tile to when the scan_out data is returned. See the C1 part of Example 1 in the HostBScanInterface reference section for a usage example of this switch.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This tool automatically infers this switch to on using the create_dft_specification command when you use the "set_dft_specification_requirements -design_type tile" command.

- auxiliary_output : *port_naming* ;

  An attribute used to provide the name for the port carrying the auxiliary_output function. When not specified, the port is named "bscan_%s_aux_out". The presence of the %s symbol somewhere in the string is required. The symbol %s is replaced by the port_name the auxiliary output is associated with. If the port is an element of a bus, the opening bracket is replaced by an underscore. For example, the default name for the auxiliary output port associated with port GPIO[4] is "bscan_GPIO_4_aux_out".

- auxiliary_output_enable : *port_naming* ;

  An attribute used to provide the name for the port carrying the auxiliary_output_enable function. When not specified, the port is named "bscan_%s_aux_out_en". The presence of the %s symbol somewhere in the string is required. The symbol %s is replaced by the port_name the auxiliary output enable is associated with. If the port is an element of a bus, the opening bracket is replaced by an underscore. For example, the default name for the auxiliary output enable port associated with port GPIO[4] is "bscan_GPIO_4_aux_out_en".

- auxiliary_input : *port_naming*, ... ;

  An attribute used to provide the name for the port carrying the auxiliary_input function. When not specified, the port is named "bscan_%s_aux_in". The presence of the %s symbol somewhere in the string is required. The symbol %s is replaced by the port_name the auxiliary input is associated with. If the port is an element of a bus, the opening bracket is replaced by an underscore. For example, the default name for the auxiliary input port associated with port GPIO[4] is "bscan_GPIO_4_aux_in".

- auxiliary_input_enable : *port_naming*, ... ;

  An attribute used to provide the name for the port carrying the auxiliary_input_enable function. When not specified, the port is named "bscan_%s_aux_in_en". The presence of the %s symbol somewhere in the string is required. The symbol %s is replaced by the port_name the auxiliary output enable is associated with. If the port is an element of a bus, the opening bracket is replaced by an underscore. For example, the default name for the auxiliary input enable port associated with port GPIO[4] is "bscan_GPIO_4_aux_in_en". This function is rarely used. It is only used when the input pad cell has a built-in JTAG mux or when the input pad buffer has a bscan_select port function.

- ac_init_clock0 : *port_naming* ;

  An attribute used to provide the name for the port carrying the ac_init_clock0 function, which has a low "off" state. When not specified, the port is named "bscan_ac_init_clk0". This port is only created when AC pads are present.

- ac_init_clock1 : *port_naming* ;

  An attribute used to provide the name for the port carrying the ac_init_clock1 function, which has a high "off" state. When not specified, the port is named "bscan_ac_init_clk1". This port is only created when AC pads are present.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ac_signal : *port_naming* ;

  An attribute used to provide the name for the port carrying the ac_signal function. When not specified, the port is named “bscan_ac_signal”. This port is only created when AC pads are present.

- ac_mode_en : *port_naming* ;

  An attribute used to provide the name for the port carrying the ac_mode_en function. When not specified, the port is named “bscan_ac_mode_en”. This port is only created when AC pads are present.

Examples

The following example changes the interface port naming by adding the prefix “ts_bscan_” to all created ports. Not all port names are affected because it depends on the values specified in the ImplementationOptions wrappers and the use of those functions in reused BScanSegment elements.

```
DftSpecification(my_block,rtl) {
    EmbeddedBoundaryScan {
    Interface {
      force_disable           : ts_bscan_force_disable ;
      select_jtag_input       : ts_bscan_select_jtag_input ;
      select_jtag_output      : ts_bscan_select_jtag_output ;
      capture_shift_clock     : ts_bscan_cs_clock ;
      capture_shift_clock_inv : ts_bscan_cs_clock_inv ;
      update_clock            : ts_bscan_update_clock ;
      capture_en              : ts_bscan_capture_en ;
      shift_en                : ts_bscan_shift_en ;
      update_en               : ts_bscan_update_en ;
      scan_in                 : ts_bscan_scan_in ;
      scan_out                : ts_bscan_scan_out ;
      scan_out_pipeline       : ts_bscan_scan_out_pipeline ;
      auxiliary_output        : ts_bscan_%s_aux_out ;
      auxiliary_output_enable : ts_bscan_%s_aux_out_en ;
      auxiliary_input         : ts_bscan_%s_aux_in ;
      auxiliary_input_enable  : ts_bscan_%s_aux_in_en ;
      ac_init_clock0          : ts_bscan_ac_init_clk0 ;
      ac_init_clock1          : ts_bscan_ac_init_clk1 ;
      ac_signal               : ts_bscan_ac_signal ;
      ac_mode_en              : ts_bscan_ac_mode_en ;
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ImplementationOptions

Specifies implementation options when building the boundary-scan cells.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan | EmbeddedBoundaryScan {
    ImplementationOptions {
      clocking           : tck | gated_tck | gated_tck_inv ; // *DefSpec
      update_stage       : flop | latch | auto ;             // *DefSpec
      scan_path_retiming : flop | latch ;                    // *DefSpec
    }
  }
}
```

Description

A wrapper that specifies implementation options when building the boundary-scan cells. The options consist of the clocking, and the use of latches or flip-flops as an update stage and for scan path retiming.

Arguments

- clocking: tck | gated_tck | gated_tck_inv ;

  A property that specifies how the clocking of the boundary-scan cell should be constructed. When specifying tck, the boundary-scan cells are clocked with the free-running TCK clock and the capture and shift is enabled using the capture_en and shift_en functions. This option requires one extra multiplexer per boundary-scan cell as compared to the gated_tck or gated_tck_inv implementations. When specifying gated_tck, a gated low version of TCK is used and the gating lets the TCK go through during the capture and shift enable pulse. When specifying gated_tck_inv, a gated high version of TCK is used and the gating lets the inverted TCK go through during the capture and shift enable pulse. All three clocking specifications behaves identically in IEEE 1149.1 mode. Using TCK versus the other two gated TCK options is a trade-off between area and the number and sizes of clock trees. Note that in all three cases, the boundary-scan cells are always built such that the scanin of each cell is strobed on the rising edge of TCK and the scanout is launched off the negedge of TCK to make sure boundary-scan cells always work without having to balance TCK to all boundary scan cells which makes the ground bounce issue worst.

  The gated_tck_inv option is a legacy clocking mode and is the one that is compatible with the existing LogicVision burst mode architecture. For more information, see the "BurstMode Logic BIST Architecture" section in the *LV Flow User's Manual*.

  Table 13-4 illustrates the boundary scan cell implementations based on the clocking, update_stage and scan_path_retiming values.

- update_stage: flop | latch | auto ;

  A property that specifies if the update stage is to be implemented using a latch or a flop. Table 13-4 illustrates the boundary scan cell implementations based on the clocking, update_stage, and scan_path_retiming values.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When this property is set to auto, a flop is used if the value of the max_segment_length_for_logictest property is not “off” or if the value of the clocking property is “tck”. Using a latch causes false E5 DRC violations and loss of coverage if X-bounding is used. (If X-bounding is not used, fault coverage is not affected.)

- scan_path_retiming : flop | latch ;

  A property that specifies if the scan path retiming element is to be implemented using a latch or a flop.

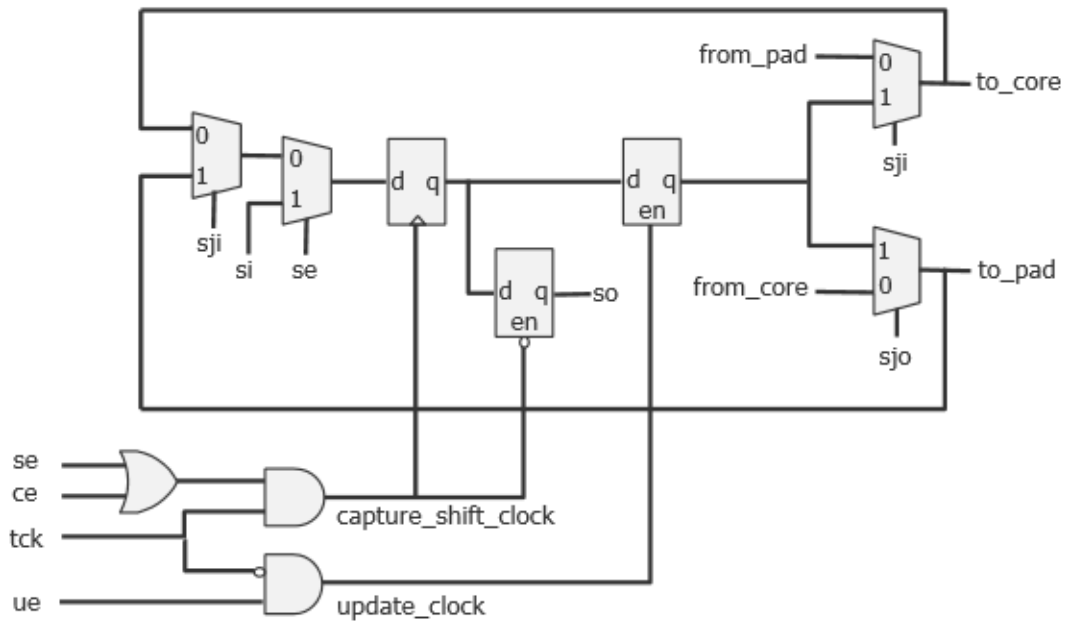
  Table 13-4 illustrates the boundary scan cell implementations based on the clocking, update_stage, and scan_path_retiming values. Refer to Table 13-6 to see the implementation of an input and sample_only boundary scan cell using the default implementation options. The following function abbreviations are used in the table figures:

  - se — shift enable
  - ce — capture enable
  - ue — update enable
  - si — scan input
  - so — scan output
  - sji — select jtag input
  - sjo — select jtag output
  - tck — test clock

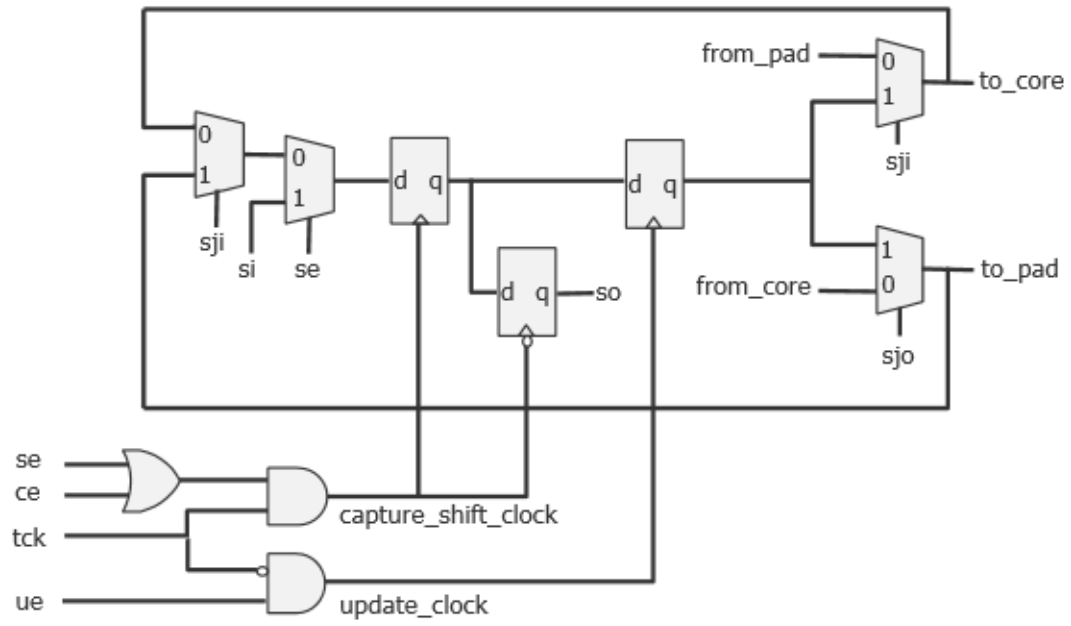
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-4. Illustration of Different Boundary Scan Implementations

1) This example shows how the boundary-scan cells are constructed when clocking is set to gated_tck, and update_stage and scan_path_retiming are both set to latch.



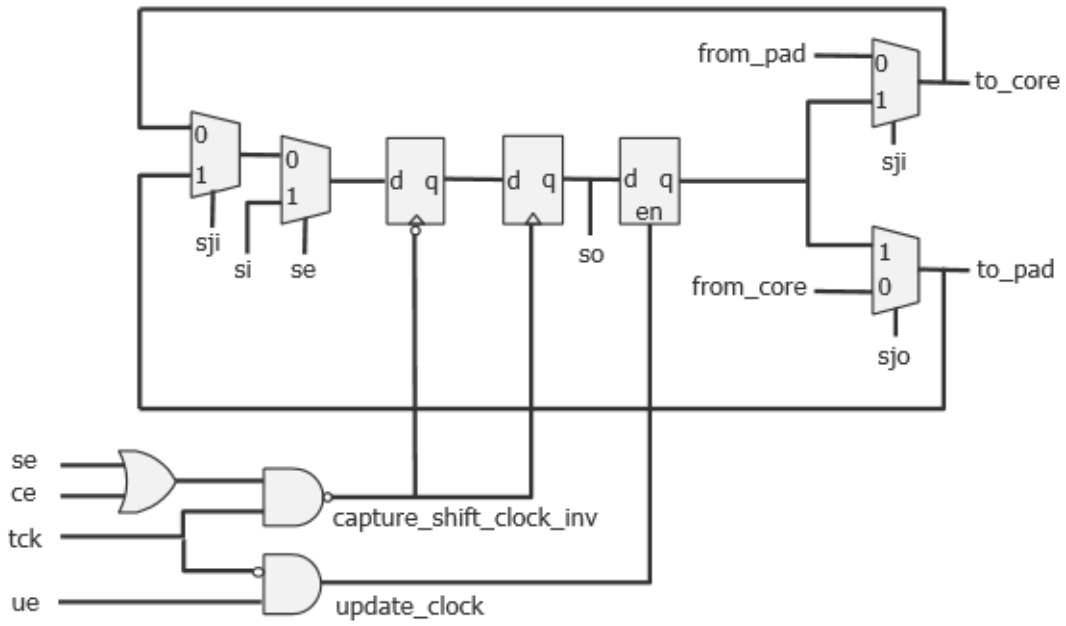
2) This example shows how the boundary-scan cells are constructed when the clocking is set to gated_tck and update_stage and scan_path_retiming are both set to flop. Compared to the prior example, this shows how changing update_stage and scan_path_retiming from latch to flop simply converts the latch into a flop.



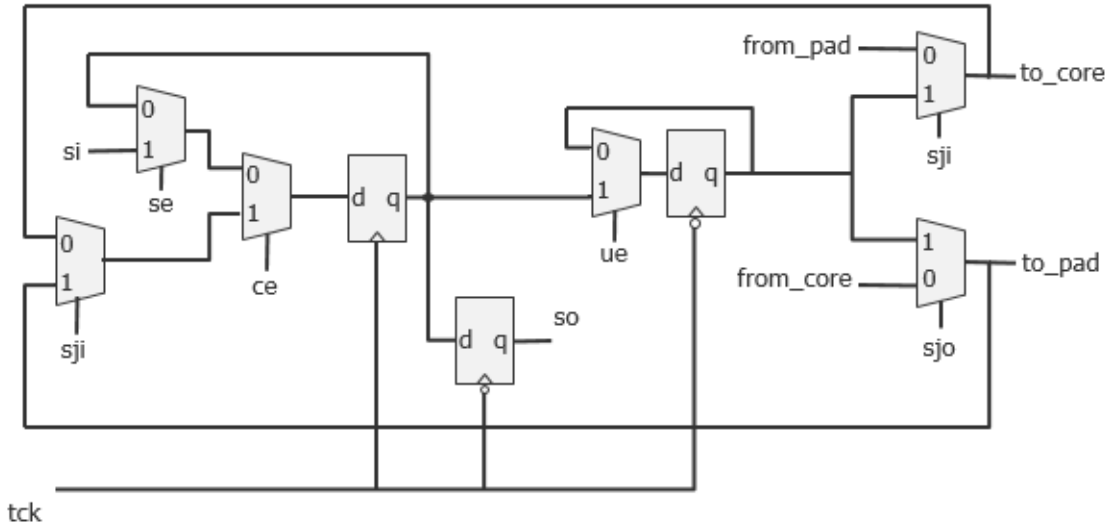
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-4. Illustration of Different Boundary Scan Implementations (cont.)

3) This example shows how the boundary-scan cells are constructed when clocking is set to gated_tck_inv, and update_stage and scan_path_retiming are both set to latch. This is a legacy mode that makes the boundary-scan cell compatible with the LogicVision burst mode architecture. Note how the implementation for gated_tck and gated_tck_inv are almost the same; only the active-low retiming flop is placed before the data flop.



4) This example shows how the boundary-scan cells are constructed when clocking is set to tck, and both update_stage and scan_path_retiming are set to flop. Notice the two extra multiplexers that are needed in each cell.



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example sets the clocking property to gated_tck_inv and the update_stage to latch. The scan_path_retiming is unspecified so it defaults to latch.

```
DftSpecification(mychip,rtl) {
  BoundaryScan {
    ImplementationOptions {
      clocking     : gate_tck_inv ;
      update_stage : latch ;
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BoundaryScanCellOptions

Assigns options to boundary scan ports.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan | EmbeddedBoundaryScan {
    BoundaryScanCellOptions {
      port_name_pattern : option, ... ; // repeatable
    }
  }
}
```

Description

This wrapper is used to assign options to boundary scan ports. This wrapper replaces the Overrides wrapper in the *.etassemble* file.

Arguments

- *port_pattern_name* : *option*, ... ;

  A syntax used to assign one or more options to a set of ports matched by the specified *port_pattern_name*. The *port_pattern_name* can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and, when specified, is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches "ABC00", "ABC01", and "ABC02".

  Table 13-5 lists the option names and their meanings. The option can be specified in the BoundaryScanCellOptions wrapper or in the third column of the *pin_order_file*.

Table 13-5. BoundaryScanCellOptions list

Option name	Meaning
add_dot6_from_pad_cell	Used to identify AC input and inout ports where the from_pad data signal is intercepted with an additional boundary scan cell. The boundary scan cells sampling the AC pad Test Receivers is added as normal.
analog	Used to identify input, output, and inout ports that are analog. These pins do not have a boundary scan cell and are reported as linkage pins in the BSDL file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-5. BoundaryScanCellOptions list (cont.)

Option name	Meaning
compliance_enable0	Used to identify input and inout ports that are active low compliance enable pins. These ports are reflected as such in the BSDL file to reflect the fact that these pins must be driven low for the chip to behave in accordance with the BSDL description. The compliance enable zero pins are automatically asserted low in any pattern that makes use of the TAP controller. This option is automatically set when you use the -enable_low_compliance_enables switch of the create_dft_specification command. If the compliance enable logic is already present in your design, you cannot use the -enable_low_compliance_enables switch of the create_dft_specification command. Instead, you follow the steps illustrated in Example 2 of the IjtagNetwork wrapper section.
compliance_enable1	Used to identify input and inout ports that are active high compliance enable pins. Those ports are reflected as such in the BSDL file to reflect the fact that these pins must be driven high for the chip to behave in accordance with the BSDL description. The compliance enable one ports are automatically asserted high in any pattern that makes use of the TAP controller. This option is automatically set when you use the -enable_high_compliance_enables switch of the create_dft_specification command. If the compliance enable logic is already present in your design, you cannot use the -enable_high_compliance_enables switch of the create_dft_specification command. Instead, you follow the steps illustrated in Example 2 of the IjtagNetwork wrapper section.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-5. BoundaryScanCellOptions list (cont.)

Option name	Meaning
clock	Identifies input and inout ports that are clocks. Specifying this value alters the RTL and the BSDL definition of the generated boundary-scan cells associated with the port. • If the port is of type input, the boundary-scan cell has a BSDL function of "clock". No JTAG multiplexer is inserted along the functional input path, just as if the sample_only option was specified. • If the port is of type inout, the boundary-scan cell has a BSDL function of "bidir" to cover both the input and output directions. There is no JTAG multiplexer inserted along the functional input path, just as if the sample_only option was specified. The boundary-scan cell is modified to capture itself instead of the from_core signal in the output direction, just as if the no_capture_core_signal option was specified. If the output direction is not used, you can use the "input_only" option in addition to the "clock" option. In this case, the BSDL function is "clock".
combined_bidir_cells	For bidirectional ports, specifies that the tool add a single boundary scan TS_BC_7 data cell to handle both input and output signal flow. This is the default. If you need an alternative method that specifies two separate boundary scan cells for a port, use the separated_bidir_cells option instead.
dont_touch	Used to identify input, output, and inout ports that are to be fully excluded from boundary scan. It is similar to using the no_bscan_cell option except that for output and inout ports, the output enable pin of the output pad buffer is not intercepted with the force_disable signal such that the port cannot go tri-state during the HIGHZ instruction. These ports are reported as linkage pins in the BSDL file.
ground	Used to identify input and inout ports that are ground pins in the package part. These ports are reported as linkage in the BSDL file. The ports associated with the ground option may or may not exist in the design. If they do not exist in the design file, the port name pattern cannot include the asterisk wildcard symbol "*" but you can use the %d[range] syntax to define a group of numbered ports as ground ports. The ports on which you have set the function attribute to ground prior to running the check_design_rules command are automatically identified as ground pins in the created DftSpecification wrapper when you run the create_dft_specification command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-5. BoundaryScanCellOptions list (cont.)

Option name	Meaning
input_only	Used to identify inout ports that are to be treated as input-only ports during boundary-scan insertion, even if the actual port direction is inout in the netlist and the port is connected to a bidirectional pad cell. The generated BSDL file documents the port as input, and the port is only equipped with an input-type boundary-scan cell. **Note:** The input_only option works in conjunction with the clock or sample_only option if you enclose both options in braces. For example: "{input_only sample_only}".
no_bscan_cell	Used to identify input, output, and inout ports that are not to be equipped with a boundary-scan cell. It is similar to using the dont_touch option except that for output and inout ports, the output enable pin of the output pad buffer is still intercepted with the force_disable signal such that the port still goes tri-state during the HIGHZ instruction. These ports are reported as linkage pins in the BSDL file.
no_connect	Used to identify input, output, and inout ports that are not connected to package pins. Those ports are treated as dont_touch pins and they are not reported at all in the BSDL file.
no_capture_core_signal	Used to identify output and inout ports that are to be equipped with boundary-scan cells that do not capture the functional core signal but instead hold their values during the capture cycle of the INTEST instruction. This option is useful when you are reusing the boundary scan chain as isolation during logictest and you want to avoid the boundary-scan cells from capturing unknowns when the from_core signal is not controllable during logictest. Avoiding unknowns is a requirement if you are using LogicBIST.
output_only	Used to identify inout ports that are to be treated as output-only ports during boundary-scan insertion even if the actual port direction is inout in the netlist and the port is connected to a bidirectional pad cell. The generated BSDL file documents the port as output, and the port is only equipped with an output-type boundary-scan cell.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-5. BoundaryScanCellOptions list (cont.)

Option name	Meaning
power	Used to identify input and inout ports that are power pins in the package part. These ports are reported as linkage in the BSDL file. The ports associated with the power option may or may not exist in the design. If they do not exist in the design file, the port name pattern cannot include the asterisk wildcard symbol "*" but you can use the %d[range] syntax to define a group of numbered ports as ground ports. The ports on which you have set the function attribute to power prior to running the check_design_rules command are automatically identified as power pins in the created DftSpecification wrapper when you run the create_dft_specification command.
sample_only	Identifies input and inout ports that are not to have a JTAG multiplexer inserted along the functional input path. The fanout of these ports cannot be intercepted by the boundary-scan cell value during the INTEST instruction. Also, if you reuse the boundary scan as isolation during logictest, these ports are not isolated. This option is inferred for input or inout ports having the clock option. You typically also want to use it for Async reset input ports.
separated_bidir_cells	For bidirectional ports, specifies that the tool add two boundary scan BC_2 data cells, one to handle input signal flow and the other to handle output. The default is combined_bidir_cells, which specifies a single boundary scan cell for a port.

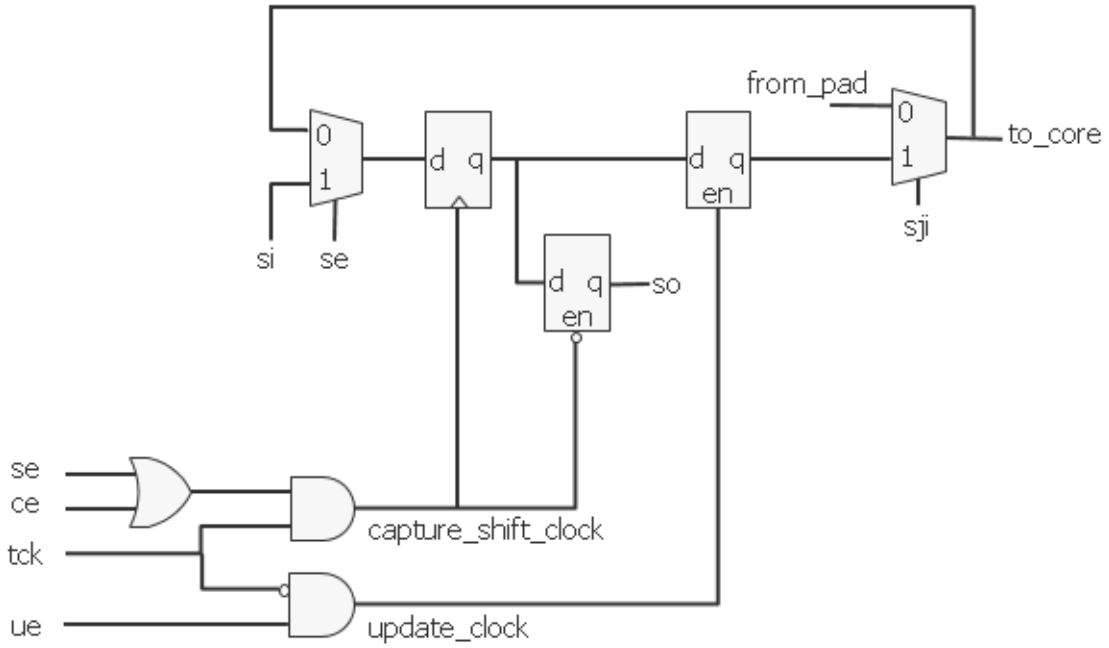
The differences between a normal input boundary scan cell and one assigned the sample_only option are illustrated in Table 13-6 below. Refer to Table 13-4 for information on how an inout cell is constructed. The following function abbreviations are used in the table figures:

- se — shift enable
- ce — capture enable
- ue — update enable
- si — scan input
- so — scan output
- sji — select jtag input
- sjo — select jtag output
- tck — test clock

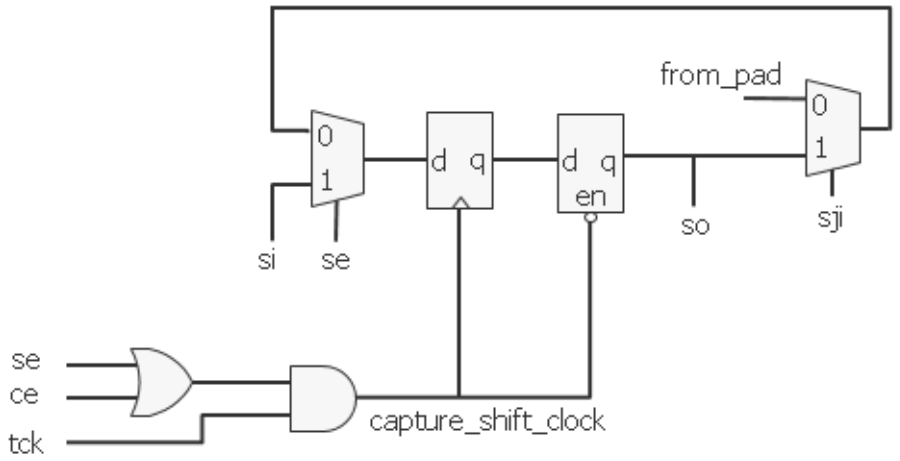
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-6. Illustration of the "sample_only" Boundary Scan Cell Option

1) This example shows a normal input boundary scan cell. The multiplexer shown on the right intercepts the from_pad connection of the pad cell. The boundary scan cell can drive a value to the core in INTEST mode.



2) The example below shows a sample_only input boundary scan cell. The multiplexer shown on the right no longer intercepts the from_pad connection of the pad cell, but simply connects to it, preserving the direct connection between the pad and core. Note that compared to the normal input boundary scan cell, the update latch has been removed and the retiming latch is now connected to the multiplexer on the right.



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example uses the BoundaryScanCellOptions wrapper to identify VDD and VSS ports in the BSDL file. These ports do not exist in the design. The pattern clk* matches any ports in the design starting with the string “clk” as clock ports. They do not have a JTAG mux inserted along their functional input path, and they are identified as Clock ports in the BSDL file.

```
DftSpecification(mychip,rtl) {
  BoundaryScan {
    BoundaryScanCellOptions {
      VDD%d[0:32] : power ;
      VSS%d[0:56] : ground ;
      clk* : clock;
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BondingConfigurations

Specifies different bonding configurations for design cores with embedded boundary scan chains.

Usage

```
DftSpecification(module_name,id) {
  EmbeddedBoundaryScan {
    BondingConfigurations {
      BondingConfiguration(name) {
        enable_signal           : port_pin_or_net_name ;
        bypassed_logical_groups : logical_group_name, ... ;
        active_logical_groups   : logical_group_name, ... ;
        TcdBscanSegment(instance_name) {
          segment_selection : segment_selection_name;
        }
      }
    }
  }
}
```

Description

A wrapper that specifies different boundary scan chain configurations based on bonding variations in the cores of a design. Use this wrapper to bypass certain parts of the boundary scan chain in the cores.

Arguments

- *name*

  A string that uniquely identifies each bonding configuration.

- enable_signal : *pin_or_net_name* ;

  A property that specifies a signal inside the design that is 1 when the given bonding configuration is used, and 0 for all other bonding configurations.

  The *pin_or_net_name* can be a DftSignal(*name*) that you previously registered with the register_static_dft_signal_names command as a global DFT control signal and added with the add_dft_signals command.

  You must ensure the enable_signal has the correct value during simulation. The current enable_signal should be 1 and all others 0. If it is a DftSignal, you can use the Patterns/DftControlSettings property to set the value. For a pin or net, you are responsible for setting the value during simulation.

  You must specify an enable signal if you define bypassed_logical_groups or active_logical_groups.

  Note

  The tool does not create the enable signal if it is not already present in the design and the DftSpecification/allow_port_creation_on_current_design property is off.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- bypassed_logical_groups : *logical_group_name*, ... ;

  A property used to list one or more logical grouping modules to be bypassed in the current bonding configuration. You use this option to optimize the length of the boundary scan chain for each bonding configuration, or if the segment is completely powered down in the current bonding configuration.

  This option is mutually exclusive with the active_logical_groups property.

- active_logical_groups : *logical_group_name*, ... ;

  A property used to list one or more logical grouping modules that are to be active in the current bonding configurations. You use this option if you want to optimize the length of the boundary scan chain for each bonding configuration, or if the segments specified here are the only ones powered up in the current bonding configuration.

  This option is mutually exclusive with the bypassed_logical_groups property.

- TcdBscanSegment(*instance_name*) { segment_selection: *segment_selection_name* }

  An instance name and segment selection name pair that specifies bypassed segments for lower-level modules with embedded boundary scan. You must specify this for every BondingConfiguration if there are any lower-level embedded boundary scan segments that can be bypassed.

Examples

The following example shows the DftSpecification for a block “core1” in a design “chip1.” Two logical groups “group1 and “group2” are created, and with enable_signal “bonding_enable” set to high (1’b1), group1 is bypassed:

```
DftSpecification(core1,rtl) {
  EmbeddedBoundaryScan {
    ...
    LogicalGroups {
      LogicalGroup(group1) {
        ...
      }
      LogicalGroup(group2) {
        ...
      }
    }
    BondingConfigurations {
      BondingConfiguration(allsegments) {
      }
      BondingConfiguration(bypass_group1) {
        enable_signal : bonding_enable;
        bypassed_logical_groups : group1;
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If there is another block “core2” within which core1 is instantiated, the DftSpecification for core2 may be defined as follows, with the TcdBscanSegment definitions for the bypass groups:

```
DftSpecification(core2,rtl) {
  ...
  EmbeddedBoundaryScan {
    ...
    BondingConfigurations {
      BondingConfiguration(config1) {
        TcdBscanSegment(core1_i) {
          segment_selection : allsegments;
        }
      }
      BondingConfiguration(config2) {
        TcdBscanSegment(core1_i) {
          segment_selection : bypass_group1;
        }
      }
    }
  }
}
```

Finally, at the top level, the DftSpecification for chip1 defines the TcdBscanSegments for core2:

```
DftSpecification(chip1,rtl) {
  ...
  BoundaryScan {
    ...
    BondingConfigurations {
      BondingConfiguration(config1) {
        TcdBscanSegment(core2_i) {
          segment_selection : allsegments;
        }
      }
      BondingConfiguration(config2) {
        TcdBscanSegment(core2_i) {
          segment_selection : bypass_group1;
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

HostBScanInterface

A wrapper to describe the host boundary scan.

Usage

```
DftSpecification(<module_name>,<id>) {
  EmbeddedBoundaryScan {
    HostBScanInterface(<id>) {

      Interface {
        ijtag_host_interface       : <id>;

        select                     : string; // def: bscan_select
        reset                      : string; // def: bscan_reset
        reset_present              : on | off | auto;
        bscan_clock                : string; // def: bscan_clock
        capture_en                 : string; // def: bscan_capture_en
        shift_en                   : string; // def: bscan_shift_en
        update_en                  : string; // def: bscan_update_en
        scan_in                    : string; // def: bscan_scan_in
        scan_out                   : string; // def: bscan_scan_out

        force_disable              : string; // def: bscan_force_disable
        force_disable_present      : on | off | auto;
        select_jtag_input      : string; // def: bscan_select_jtag_input
        select_jtag_input_present  : on | off | auto;
        select_jtag_output     : string; // def: bscan_select_jtag_output
        select_jtag_output_present : on | off | auto;

        ac_init_clock0             : string; // def: bscan_ac_init_clk0
        ac_init_clock0_present     : on | off | auto; // def: auto
        ac_init_clock1             : string; // def: bscan_ac_init_clk1
        ac_init_clock1_present     : on | off | auto; // def: auto
        ac_signal                  : string; // def: bscan_ac_signal
        ac_signal_present          : on | off | auto; // def: auto
        ac_mode_en                 : string; // def: bscan_ac_mode_en
        ac_mode_en_present         : on | off | auto; // def: auto
      }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
        EBScanInstance(<instance_name>) {
        }
        EBScanPipeline(<id>) {
          parent_instance    : instance_name;
          leaf_instance_name : leaf_instance_name;
          so_retiming        : on | off;
          Interface {
            bscan_clock : string; // def: bscan_clock
            select      : string; // def: bscan_select
            shift_en    : string; // def: bscan_shift_en
            scan_in     : string; // def: bscan_scan_in
            scan_out    : string; // def: bscan_scan_out
          }
        }
        SecondaryEBScanInterface(<id>) {
          select                    : string; // def: %s_bscan_to_select
          reset                     : string; // def: %s_bscan_to_reset
          reset_present             : on | off | auto; // default: auto
          bscan_clock               : string; // def: %s_bscan_to_clock
          capture_en                : string; // def: %s_bscan_to_capture_en
          shift_en                  : string; // def: %s_bscan_to_shift_en
          update_en                 : string; // def: %s_bscan_to_update_en
          scan_out                  : string; // def: %s_bscan_to_scan_in
          scan_in                   : string; // def: %s_bscan_from_scan_out

          force_disable          : string; // def: %s_bscan_to_force_disable
          force_disable_present     : on | off | auto;
          select_jtag_input : string; // def: %s_bscan_to_select_jtag_input
          select_jtag_input_present  : on | off | auto;
          select_jtag_output: string;// def: %s_bscan_to_select_jtag_output
          select_jtag_output_present : on | off | auto;

          ac_init_clock0          : string; // def: %s_bscan_to_ac_init_clk0
          ac_init_clock0_present : on | off | auto;
          ac_init_clock1          : string; // def: %s_bscan_to_ac_init_clk1
          ac_init_clock1_present : on | off | auto;
          ac_signal               : string; // def: %s_bscan_to_ac_signal
          ac_signal_present       : on | off | auto;
          ac_mode_en              : string; // def: %s_bscan_to_ac_mode_en
          ac_mode_en_present      : on | off | auto;
        }
      }
    }
  }
```

Description

Use this wrapper in a tiling design flow to create SecondaryEBScanInterface ports, and connect EBScan segments to a TAP controller. See Example 1 for usage details.

Note

You cannot specify any other child wrappers within the EmbeddedBoundaryScan wrapper when you specify the HostBScanInterface wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- HostBScanInterface(*id*)

  A wrapper that defines the start and end point of a boundary scan chain. The tool uses the specified *id* as the ICL ScanInterface id when the Interface is defined on ports of the current design.

- Interface/ijtag_host_interface : HostScanInterface(*id*) | Tap(*id*)/HostBscan;

  Use this property when connecting the boundary scan chain to a TAP controller that is inserted in the same insertion pass, or to a TAP controller that you inserted in a previous insertion pass.

  When you insert the TAP at the same time, use the Tap(id)/HostBscan value to reference the TAP controller.

  If you insert the TAP in a previous insertion pass, use a HostScanInterface(tap) wrapper within the IjtagNetwork wrapper, and use the design_instance and scan_interface properties to point to the HostBscan interface of the TAP instance.

  To connect the boundary scan chain to the top-level ports of a tile module, leave the ijtag_host_interface property unspecified. You can use the HostBScanInterface/Interface/*function_name* properties to control the port naming instead of the default naming.

- Interface/*function_name* : *string*;

  A set of properties to control the port naming associated with each function. See the Usage summary for a list of function properties and their default values.

- Interface/*function_name*_present : on | off | auto;

  A set of properties to control the presence of a function in the port list. The auto value translates to on when the given function is used by any child element. The default is auto.

- EBScanInstance(*instance_name*)

  An optional wrapper that references a design instance that has an EBScan chain segment. The EBScan chain segment was either created with the EmbeddedBoundaryScan flow on the child module or using a third-party method. In either case, you need a proper *.tcd_bscan* description for the module. See "Tessent Core Description" in the *Tessent BoundaryScan User's Manual* for more details.

- EBScanPipeline(*id*)

  An optional wrapper that specifies the insertion of a one-bit pipeline along the boundary scan chain, described as an internal cell within the BSDL file.

- EBScanPipeline(*id*)/parent_instance : *instance_name*;

  A property to specify the parent instance in which the pipeline flop is to be instantiated into. By default, the pipeline flop is instantiated in the current design.

- EBScanPipeline(*id*)/leaf_instance_name : *leaf_instance_name*;

  A property to control the leaf instance name of the pipelining flop. The default value is <*current_design*>_<*design_id*>_tessent_bscan_ebscan_pipe_<*id*>_inst.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- EBScanPipeline(*id*)/so_retiming : on | off;

  A Boolean property to control the presence of a retiming cell on the scan_out output of the pipelining cell. Set this property to off on the last pipeline stage in a tiled child physical block to achieve a full clock period delay in the loop timing interface, as shown with the C2 module in the example below.

- EBScanPipeline(*id*)/Interface/*function_name* : *string*;

  A set of properties to control the naming of the port associated with each function. See the Usage summary for the list of functions and their default values.

- SecondaryEBScanInterface(*id*)

  A wrapper to specify the presence of a secondary EBScan interface to serve as the host for an EBScan interface within an adjacent tile. See the C2 and C3 examples in Example 1.

- SecondaryEBScanInterface(*id*)/*function_name* : *port_name*;

  A set of properties to control the port naming associated with each function. See the Usage summary to see the list of functions and their default port naming. The *id* of the SecondaryEBScanInterface wrapper replaces the %s string within the default value string.

- SecondaryEBScanInterface(*id*)/*function_name*_present : on | off | auto;

  A set of properties that controls the presence of the given function in the secondary EBScan interface. The default value means on for the force_disable, select_jtag_input, and select_jtag_output properties. For all other properties, auto means on only when you specify the HostBscanInterface/Interface/*function_name*_present property or it is inferred to on by the tool.

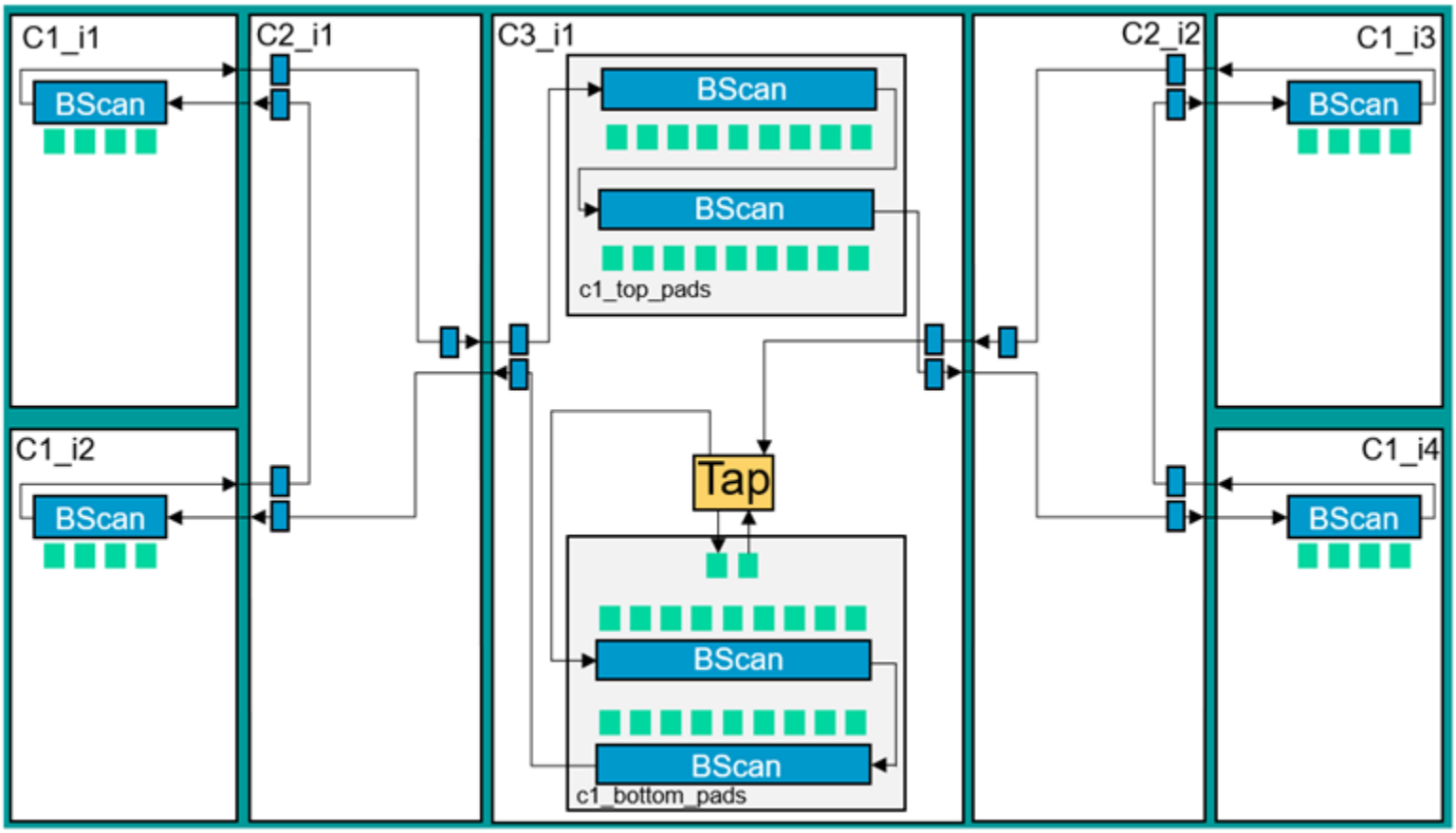
Examples

Example 1

The following example is based on the reference design described in Example 4 of the OCC wrapper. This is a tiling flow-based chip, and all pads exist in child physical blocks. The EmbeddedBoundaryScan/HostBScanInterface wrapper assembles the boundary scan chain in C2 and C3, and Figure 13-38 shows the result. The boundary scan insertion process within C1 is the standard EmbeddedBoundaryScan flow. There are no pad buffer cells inside C2, but the EmbeddedBoundaryScan/HostBScanInterface wrapper creates the two secondary EBscan interfaces and the pipelining flops, described as internal cells in the BSDL file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-38. Boundary Scan Chain Distributed Across a Tiled Design



Below is a section of the dofile to insert the boundary scan chain within C1. The flow is identical to the normal embedded boundary scan insertion flow except that you use the "set_dft_specification_requirements -design_type tile" switch. This instructs the create_dft_specification command to infer the **EmbeddedBoundaryScan/Interface/scan_out_pipeline property** set to on to get a full TCK clock period in the loop timing interface with the C2. The **allow_port_creation_on_current_design** property is set to off to stop the tool creating any new ports in the bottom-up insertion process. See the process_top_module_connections command reference page for a way to pre-create the DFT ports on the tiles and pre-connect them in the chip-level netlist.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
read_verilog ../../design/c1/c1.v_with_dft_port

set_current_design c1
set_design_level physical_block

set_dft_specification_requirements -boundary_scan on -design_type tile
set_boundary_scan_port_options -pad_io_ports {clk_ref rx_data_in \
                                              tx_data_out}
check_design_rules

set spec [create_dft_specification]

# Prevents ports to be created in tiling flow and
# makes sure all DFT ports were properly added and
# connected in the die netlist.
set_config_value allow_port_creation_on_current_design -in $spec off

# Use EBScan segment as Dedicated wrapper cells during logictest
set_config_value EmbeddedBoundaryScan/max_segment_length_for_logictest
-in $spec 200

process_dft_specification

extract_icl -create_ijtag_graybox on
```

C2 Boundary Scan Insertion

Below is a section of the dofile to insert the boundary scan chain within C2. You specify the so_retiming function to off in the EBScanPipeline(client_out) wrapper because C2 is a tiled physical block and gets its clock from C3 and returns the bscan_so to C3. This implements a loop timing interface for the scan_out. The lack of a retiming element gives the timing path a full clock cycle.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_current_design c2
set_design_level physical_block

read_config_data -from_string {
  DftSpecification(c2,rtl1) {
    // Prevents ports to be created in tiling flow and
    // makes sure all DFT ports were properly added and
    // connected in the die netlist.
    allow_port_creation_on_current_design : off;
    EmbeddedBoundaryScan {
      HostBScanInterface(client) {
        EBScanPipeline(client_out) {
          so_retiming : off;
        }
        EBScanPipeline(from_r1) {
        }
        SecondaryEBScanInterface(r1) {
        }
        EBScanPipeline(to_r1) {
        }
        EBScanPipeline(from_r2) {
        }
        SecondaryEBScanInterface(r2) {
        }
        EBScanPipeline(to_r2) {
        }
      }
    }
  }
}

process_dft_specification
```

C3 Child Block Boundary Scan Insertion

The C3 tile includes two sets of pad buffers and the TAP controller. Separate modules wrap the two sets of pad buffers to insert the boundary scan chains using the standard EmbeddedBoundaryScan flow. The boundary scan chains are used as dedicated wrapper chains of the C3 module, so they are built with the DFT signal added in the dofile below.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_current_design c3_top_pads
set_design_level sub_block

set_dft_specification_requirements -boundary_scan on
set_boundary_scan_port_options \
    -pad_io_ports {clkc rst core_data_in bisr_clock bisr_trigger}
set_boundary_scan_port_options {clkc bisr_clock} -cell_options clock

add_dft_signal bscan_input_isolation_enable \
               output_pad_disable ext_ltest_en ltest_en tck_occ_en
add_dft_signal scan_en shift_capture_clock \
               -source_node {ltest_scan_en shift_capture_clock}

check_design_rules

set spec [create_dft_specification]

# Use EBScan segment as Dedicated wrapper cells during logictest
set_config_value EmbeddedBoundaryScan/max_segment_length_for_logictest \
                 -in $spec 200

# Build Auxiliary input logic for SSN
read_config_data -in_wrapper $spec/EmbeddedBoundaryScan -from_string {
  AuxiliaryInputOutputPorts {
    external_auxiliary_input_ports : core_data_in[4], core_data_in[3],
core_data_in[2], core_data_in[1], core_data_in[0];
  }
}
process_dft_specification

extract_icl -create_ijtag_graybox on

set spec [create_patterns_specification -bscan_sim_views ijtag_graybox]
process_patterns_specification

set_simulation_library_sources -V ../../library/cells/adk.v
run_testbench_simulations
```

C3 Boundary Scan Insertion

Below is the dofile section connecting the two EBScan segments created in the top and bottom pad block to the TAP. A secondary EBScan interface is created for both sides of the tile to collect the boundary chain from either side. Pipelining is added near the ports to help with timing closure. To enable simulating the boundary scan chain at the C3 tile level, use the add_loadboard_loopback_pairs and “set_flat_model_options -emulate_loadboard_loopback_pairs on” commands to close the loop on both secondary EBScan interfaces. Then, create a BSDL file assuming those connections were there.

Note

The BSDL file created for the C3 tile is only for local verification. The tool can extract a true BSDL for the chip level using the chip-level netlist. See “How to Extract BSDL at the Chip Level” below.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
read_verilog ../../library/macros/my_fuse_box.v -interface_only -
exclude_from_file_dictionary
read_design c3_top_pads -view ijtag_graybox
read_design c3_bottom_pads -view ijtag_graybox
set_current_design c3
set_design_level physical_block

set_dft_specification -host_scan_interface_type tap -design_type tile

check_design_rules

set spec [create_dft_spec]

read_config_data -in ${spec}/IjtagNetwork/HostScanInterface(tap)/
Tap(main) -from_string {
  DeviceIDRegister {
    version_code         : 4'h0;
    manufacturer_id_code : 11'h0;
    part_number_code     : 16'h0;
  }
}


read_config_data -in $spec -from_string {
  EmbeddedBoundaryScan {
    HostBScanInterface(tap) {
      Interface {
        ijtag_host_interface : Tap(main)/HostBscan;
      }
      EBScanPipeline(from_right) {
      }
      SecondaryEBScanInterface(right) {
      }
      EBScanPipeline(to_right) {
      }
      EBScanInstance(top_pads) {
      }
      EBScanPipeline(from_left) {
      }
      SecondaryEBScanInterface(left) {
      }
      EBScanPipeline(to_left) {
      }
      EBScanInstance(bottom_pads) {
      }
    }
  }
}

process_dft_specification

extract_icl -create_ijtag_graybox on
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# Close the host bscan scan interfaces and extract fake BSDL file for c3
for early verification
foreach bscan_host {left right} {
  lappend inputs  [get_ports ${bscan_host}_bscan_from_scan_out]
  lappend outputs [get_ports ${bscan_host}_bscan_to_scan_in]
}
add_loadboard_loopback_pairs -inputs $inputs -outputs $outputs
set_flat_model_options -emulate_loadboard_loopback_pairs on

set_dft_specification_requirements -bsdl_extraction on
set_boundary_scan_port_options -pad_io_ports [get_ports -filter
equipped_with_bscan]

check_design_rules

set_context patterns -ijtag -rtl

#create_patterns_specification
set spec [create_patterns_specification -bscan_sim_views ijtag_graybox]

process_patterns_specification

set_simulation_library_sources -Y ../../library/macros/ -extensions {v} -V
../../library/cells/adk.v
run_testbench_simulations -store_simulation_waveforms on
```

How to Extract BSDL at the Chip Level

The following is the dofile to extract the BSDL file at the chip level, and simulate the BSDL and ICL verify patterns:

```
read_verilog ../../design/chip/chip.v_with_dft_port

open_tsdb ../c1/tsdb_outdir
open_tsdb ../c2/tsdb_outdir
open_tsdb ../c3/tsdb_outdir
read_design c1 -view ijtag_graybox
read_design c2 -view ijtag_graybox
read_design c3 -view ijtag_graybox
read_design c3_bottom_pads -view ijtag_graybox
read_design c3_top_pads -view ijtag_graybox

set_current_design chip
set_design_level chip

extract_icl -write_in_tsdb on
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
### Extract the chip level BSDL file
set_dft_specification_requirements -bsdl_extraction on
set_attribute_value [get_port bisr_prog_voltage] -name function \
                    -value power

check_design_rules

### Perform all Signoff simulations

set spec [create_patterns_specification -bscan_sim_views ijtag_graybox]

process_patterns_specification

set_simulation_library_sources -y ../../library/macros -extension v \
                               -v ../../library/cells/adk.v
run_testbench_simulations
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

LogicalGroups

Specifies how to group the boundary-scan cells into logical grouping modules.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan | EmbeddedBoundaryScan {
    LogicalGroups {
      LogicalGroup(group_name) {
        first_port         : port_name ;
        parent_instance    : parent_instance ;
        leaf_instance_name : leaf_instance_name ;
      }
    }
  }
}
```

Description

A wrapper that specifies how to group the boundary-scan cells into logical grouping modules. You can specify more than one LogicalGroup wrapper only when you have specified the pin_order_file property in the BoundaryScan wrapper.

You use the first_port property to identify the first port in the pin order file that belongs to the logical grouping. The association between ports and the LogicalGroup can also be specified in the forth column of the pin order file. If you do not specify any LogicalGroup(*group_name*) wrapper, all boundary scan cells are grouped into a single module, the module is instantiated into the top module, and the leaf instance name is as follows:

```
design_name_design_id_tessent_bscan_logical_group_DEF_inst
```

as if you had specified an empty LogicalGroup(DEF) wrapper.

You use the parent_instance property to specify where you want the logical grouping module to be instantiated. By default, it is instantiated in the top module. You use the leaf_instance_name property to specify the leaf instance name used when instantiating the module if you are not satisfied with the default value. The default leaf_instance_name used is *design_name_design_id*_tessent_bscan_logical_group_*group_name*_inst.

The example below shows how to specify the parent_instance property in order to insert each logical grouping module in the same instance where the pins or nets to be intercepted by the boundary-scan cells are located.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- *group_name*

  A string that uniquely identifies each logical group. The string starts with a letter and can be followed by any number of letters, numbers, and underscores. The group_name is used in constructing the grouping module name as follows:

  `design_name_design_id_tessent_bscan_logical_group_group_name`

- first_port : *port_name* ;

  A property that specifies a port name that exists in the specified pin_order_file. This property is optional when there is only one LogicalGroup wrapper but it is required when there are more then one. Specifying a port_name for the first_port property is equivalent to specifying the group_name in the forth column of that port in the pin_order_file. See the pin_order_file property description in the BoundaryScan wrapper for a description of that file format and usage.

- parent_instance : *parent_instance* ;

  A property that controls where the logical grouping module is instantiated. By default, it is instantiated in the root module but you can push it down into instances. See the following example to understand the requirement that must be met when choosing the parent_instance of the logical grouping module.

  If the specified parent_instance is not unique, it is uniquified before the logical grouping module is inserted in it. You can do bottom up insertion using the EmbeddedBoundaryScan wrapper on sub modules to insert segments of boundary scan chains into repeated modules, and then reuse them as building blocks at the next level up if you want to avoid uniquifying the parent_instance. Specify the "set_design_level sub_block" command and option when inserting boundary scan into those sub modules.

- leaf_instance_name : *leaf_instance_name* ;

  A property that specifies a different leaf name when instantiating the logical grouping module into its parent instance. The default value when unspecified is *design_name_design_id*_tessent_bscan_logical_group_*group_name*_inst.

  If you choose to change it, you do not have to worry about picking a unique leaf name in the given parent_instance. The tool automatically appends the instance_uniquification_suffix to your specified leaf instance name in case of conflicts. See the set_insertion_options command description for a complete description of the -instance_unification_suffix option.

Examples

The following example illustrates the use of the parent_instance property of the LogicalGroup wrapper to meet the rule that the logical grouping module must be instantiated where all the intercepted nets are visible. Figure 13-39 shows the circuit before the insertion of the boundary-scan cells. You can see that the functional connections to and from the pad cells associated with ports P3 and P4 do not exist outside the B1_I instance. In this example, a single logical grouping module is specified so it can only be instantiated inside B1_I as illustrated in Figure 13-40.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
BoundaryScan {
  LogicalGroups {
    LogicalGroup(g1) {
      parent_instance : B1_I ;
    }
  }
}
```

The same example is modified to use a pin_order file thus enabling the splitting of the boundary-scan cells into more than one logical grouping module. The pin order file defines the order from P1 to P4. Two logical grouping modules are specified in which the first one, g1, starts with P1 and goes until P2 and the second one, g2, starts with P3 and goes until P4. The g2 group is still restricted to having its parent_instance property as B1_I because this is the only place where all of the connections to be intercepted are visible. The g1 group, on the other hand, can be instantiated in four locations because all of the connections to be intercepted are visible in those four locations. The parent_instance is shown for the four options, and Figure 13-41 shows the corresponding locations of the logical grouping modules.

```
BoundaryScan {
  pin_order_file      : mychip.pin_order
  LogicalGroups {
    LogicalGroup(g1) {
      first_port      : P1 ;
      parent_instance : B1_I ;       // Option1
      parent_instance : . ;          // Option2
      parent_instance : B2_I ;       // Option3
      parent_instance : B2_I/B3_I ;  // Option4
    }
    LogicalGroup(g2) {
      first_port      : P3 ;
      parent_instance : B1_I ;
    }
  }
}
```

The example also illustrates that you can specify the outputs_per_enable_cell property with a value smaller that the actual fanout count of the functional enable signal. In this case, the functional enable signal fans outs to two output pads. One enable boundary-scan cell is inserted per output port, requiring the tool to split the output enable connectivity to each pad.

Figure 13-42 shows this example with option1 for the parent_instance property. No new ports need to be created. The functional enable signal is observable by two enable bscan cells, and one connection per output pad is made between the logical grouping module and the enable pin of the output pad cells.

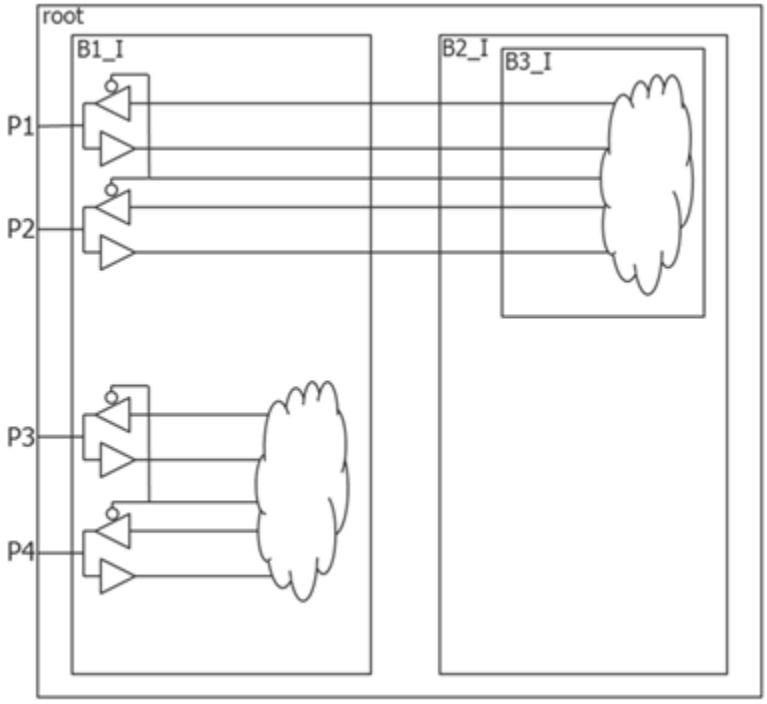
Figure 13-43 shows this example with option3 for the parent_instance property. In this case, the connections between the logical grouping module and the output pads require the creation of new pins as illustrated with the orange net crossing the hierarchy. If you are familiar with Tessent Boundary Scan within ETAssemble, the flexibility to precisely control the fanout of the enable signals was not present in that tool. All of those existing limitations are removed within Tessent Shell, and a net driven by RTL can now be intercepted and split for boundary scan

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

usage. In ETAssemble, you needed the functional enable signal to be identified on a pin which often meant you had to insert buffers in the RTL to enable the Boundary Scan to be placed in the same module as the pads.

```
BoundaryScan {
  pin_order_file      : mychip.pin_order
  OutputPerEnableCell : 1 ;
  LogicalGroups {
    LogicalGroup(g1) {
      first_port      : P1 ;
      parent_instance : B1_I ;       // Option1
      parent_instance : B2_I ;       // Option3
    }
    LogicalGroup(g2) {
      first_port      : P3 ;
      parent_instance : B1_I ;
    }
  }
}
```

Figure 13-39. Example Design With Pads



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-40. Single Logical Group Example

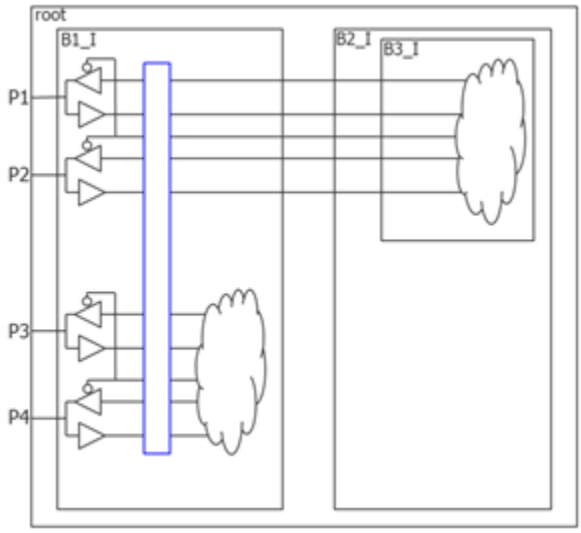
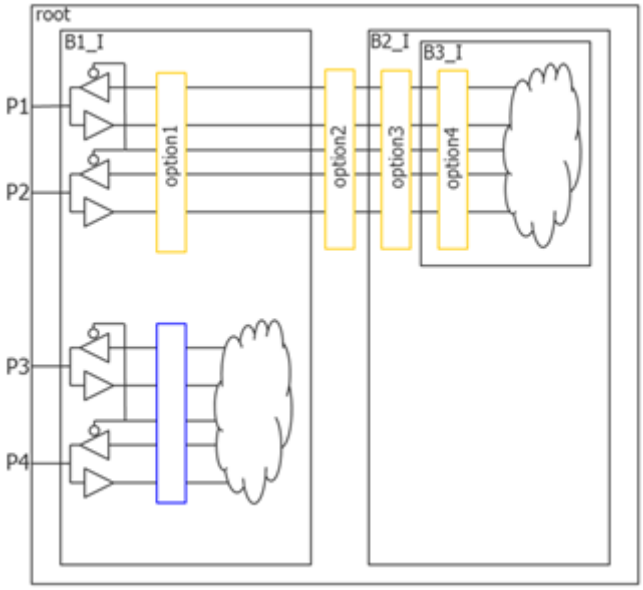


Figure 13-41. Two Logical Groups Example



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-42. Two Logical Groups Example1 with One Output per Enable Cells

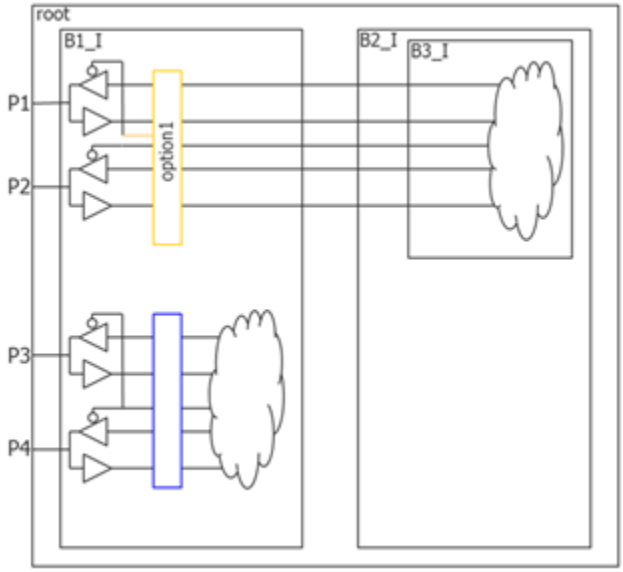
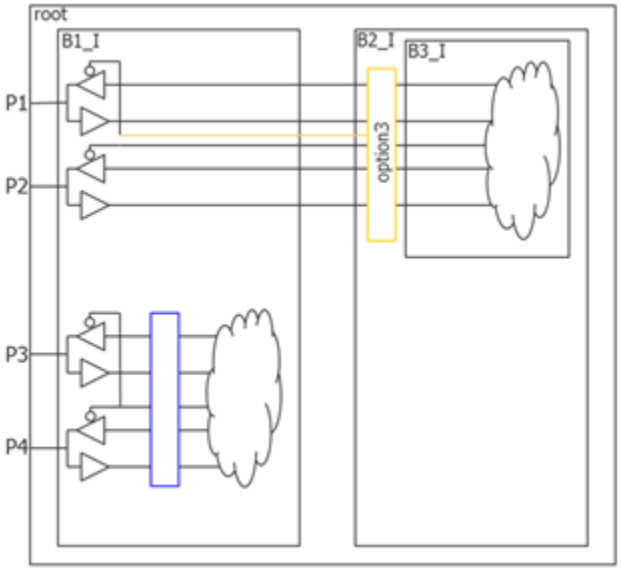


Figure 13-43. Two Logical Groups Example2 with One Output per Enable Cells



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ACModeOptions

Specifies options relevant to the IEEE 1149.6 AC mode of the boundary scan chain.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan {
    ACModeOptions {
      test_receiver_init_clock_value : initialization_event ;
                                       // Legal: logic_low logic_high
                                       // falling_edge rising_edge auto
      pulse_min_duration             : time | auto ;
      train_minimum_count            : count| unused ;
      train_max_duration             : time | unlimited ;
      ACGroup(group_name) {
        insert_before_port           : port_name ;
        insert_after_port            : port_name ;
        port_list                    : port_name, ... ;
      }
    }
  }
  EmbeddedBoundaryScan {
    ACModeOptions {
      test_receiver_init_clock_value : initialization_event ;
                                       // Legal: logic_low logic_high
                                       // falling_edge rising_edge auto
      ACGroup(group_name) {
        insert_before_port           : port_name ;
        insert_after_port            : port_name ;
        port_list                    : port_name, ... ;
      }
    }
  }
}
```

Description

A wrapper that specifies options relevant to the IEEE 1149.6 AC mode of the boundary scan chain. You use this wrapper to define AC enable cells and associate them with a group of ports. You also use this wrapper to specify the event type that initializes the test receiver when the value is not described in the cell library. Finally, you can use this wrapper to specify three properties that are simply forwarded into the BSDL file.

Arguments

- test_receiver_init_clock_value : initialization_event ;

  A property that is useful when the cell library description of the dot6 pad cell uses the legacy pad_init_clock_dot6 attribute instead of one of the new attributes that precisely describe the initialization event: pad_init_posedge_clock_dot6, pad_init_negedge_clock_dot6, pad_init_enable_high_dot6, and pad_init_enable_low_dot6. If the value is “auto” and the dot6 pad cell uses the legacy pad_init_clock_dot6 attribute instead of one of the four new attributes, the tool issues an error; you are required to specify the initialization event when it cannot be inferred from the cell library.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- pulse_min_duration : *time* | auto ;

  An optional property that defines the minimum time to use between the rising and the falling edge during the AC tests. This property is forwarded into the BSDL file using the AIO_EXTEST_Pulse_Execution attribute. When unspecified or specified to "auto", the minimum pulse duration is chosen so as not to be smaller than three times the larger of the HP_time and LP_time BSDL values of each pad cell.

  ```
  attribute AIO_EXTEST_Pulse_Execution of <current_design>: entity is
        "Wait_Duration <pulse_min_duration_in_seconds>" ;
  ```

- train_minimum_count : *count* | unused ;

  An optional property that defines the minimum number of pulses the train test must have. This property is forwarded in the BSDL file using the AIO_EXTEST_Train_Execution attribute. The property defaults to "unused" when unspecified. Specifying a value of one effectively makes the train test equivalent to the pulse test.

  ```
  attribute AIO_EXTEST_Train_Execution of <current_design>: entity is
   "train <train_minimum_count>, maximum_time <train_max_duration>" ;
  ```

- train_max_duration : *time* | unlimited ;

  An optional property that defines a maximum time the train test must take. This property is forwarded in the BSDL file using the AIO_EXTEST_Train_Execution attribute. It defaults to "unlimited" when unspecified.

- *group_name*

  A string that uniquely identifies the ACGroup. This name is not reflected in the BSDL file but is used to name the leaf instance name of the AC group enable cell in the logical grouping module.

  If no AcGroup wrappers are specified, then all AC ports are active during the AC tests. If you specified at least one ACGroup wrapper but you have some AC ports not associated with an ACGroup, they are always active during the AC tests. In this situation, a warning is generated to let you know that not all AC pins were equipped with an AC enable cell. You can define an extra ACGroup wrapper without a port_list property to indicate that you want all AC ports not associated with another ACGroup wrapper to be associated with that one.

- insert_before_port : *port_name* ;

  A property that specifies the position of the AC enable boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the insert_before_port property is mutually exclusive with the specification of the insert_after_port property. If neither the insert_before_port nor insert_after_port properties are specified, the AC enable boundary-scan cell is inserted in front of the boundary-scan cell of the first port that is part of the group. The location of the AC group enable boundary-scan cell is reflected in the BSDL file using the AIO_Pin_Behavior attribute.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- insert_after_port : *port_name* ;

  A property that specifies the position of the AC enable boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the insert_before_port property is mutually exclusive with the specification of the insert_after_port property. If neither the insert_before_port nor insert_after_port properties are specified, the AC enable boundary-scan cell is inserted in front of the boundary-scan cell of the first port that is part of the group. The location of the AC group enable boundary-scan cell is reflected in the BSDL file using the AIO_Pin_Behavior attribute.

- port_list : *port_name_pattern*, ... ;

  A property used to list the port names that belong to the AC group. The port_name_pattern can include the asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and, when specified, is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

  You can have one ACGroup wrapper with an unspecified port_list. This ACGroup collects all AC ports that are not already associated with another ACGroup wrapper.

Examples

The following example defines an ACGroup wrapper with the name "pci" to specify that all ports starting with PCI_ belong to the group. All other AC ports are associated with the "other" group. The "other" ACGroup collects all other AC ports because it does not specify the port_list property.

```
DftSpecification(my_chip,rtl) {
  BoundaryScan {
    ACModeOptions {
      ACGroup(pci) {
        port_list : PCI_* ;
      }
      ACGroup(other) {
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

AuxiliaryInputOutputPorts

Specifies two lists of port name patterns to be used as auxiliary input ports, output ports, or both, during some test modes.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan  {
    AuxiliaryInputOutputPorts {
      auxiliary_input_ports  : port_name_pattern, ... ;
      auxiliary_output_ports : port_name_pattern, ... ;
    }
  }
  EmbeddedBoundaryScan{
    AuxiliaryInputOutputPorts {
      internal_auxiliary_input_ports  : port_name_pattern, ... ;
      internal_auxiliary_output_ports : port_name_pattern, ... ;
      external_auxiliary_input_ports  : port_name_pattern, ... ;
      external_auxiliary_output_ports : port_name_pattern, ... ;
    }
  }
}
```

Description

A wrapper that specifies two lists of port name patterns to be used as auxiliary input ports or output ports during some test modes. You can specify an inout port by adding it to both lists. In the output directions, a 2-to-1 multiplexer (shown in green in Figure 13-46) is added on the test side of the JTAG multiplexer (shown in red in Figure 13-46). The auxiliary output multiplexing is added in the boundary-scan cell to avoid cascading two multiplexers along the functional output path. For auxiliary input logic, an AND gate is used to prevent the aux_in_data signal from toggling in functional mode. The aux_in_en signal is also used to force the output pad driver off when adding AuxIn logic on an inout pad.

The aux_out_enable and aux_in_enable signals are kept separate for each port to provide full flexibility as to which output and input ports are enabled to carry input or output auxiliary data in any given test mode. The ports equipped with auxiliary data input and output logic are useful for supplying channel input and output data to EDT controllers. They are also useful to provide the source of the scan_en, test_clock, and edt_update DFT signals. See the description of the add_dft_signals command for more info. Notice that the aux_out_en and aux_in_en signals have higher priority than the select_jtag_input and select_jtag_output signals. The implication of this is that the aux_out_en and aux_in_en sources must reset off and remain off when using the IEEE 1149.1 test mode of the chip. This requirement is met if you use the DFT signals created by the add_dft_signals to control those pins.

Figure 13-44 shows the typical boundary scan cell inserted on a inout pad. The red multiplexers are those referred to as JTAG multiplexers previously. They are controlled by the select_jtag_output to let the output data and output enable values be controlled by the Boundary Scan cells in IEEE 1149.1 test modes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-44. Typical Boundary Scan Cell Inserted for an Inout Pad

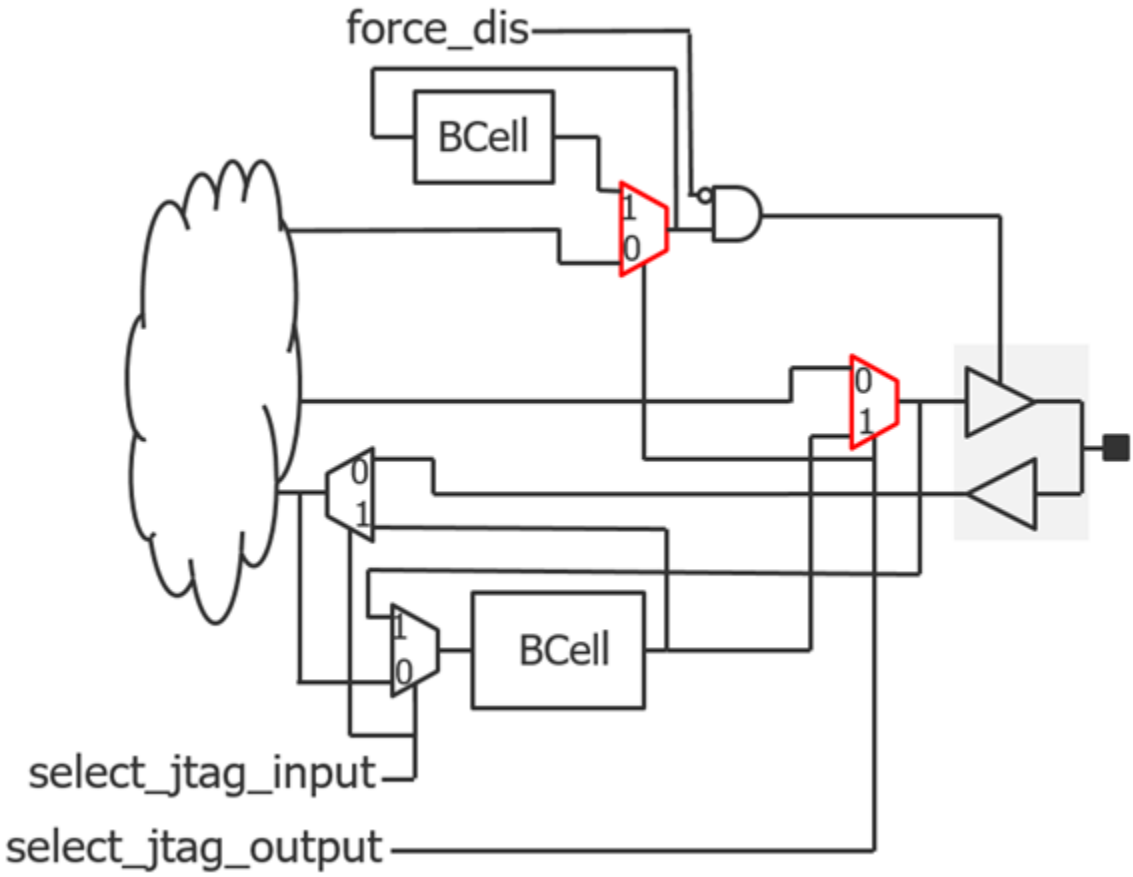


Figure 13-45 shows the boundary scan cell when it is used during ATPG. If you set the max_segment_length_for_logictest property found in the Boundary Scan wrapper to an integer or to unlimited, ATPG scan chains are created with the Boundary Scan chain. The blue path shows how the boundary scan cell is used to isolate the core values from the input ports by forcing select_jtag_input to 1, which is done by forcing the int_ltest_en pin on the boundary scan interface module. For more information about the boundary scan interface module and the logic inserted in it when the boundary scan chain is reused in logic test, see Figure 13-53 on page 4161.

Note

If you are using the DFT signal flow, add the int_ltest_en DFT signal using the add_dft_signals command when you reach the top level and it is connected to the boundary scan interface module. You can then activate the input isolation using the "set_static_dft_signal_values int_ltest_en 1" command and switch when you want to create your ATPG patterns.

The select_jtag_output signal is under ATPG control, enabling both the input and the output of the JTAG multiplexers to be tested. If you do not force the int_ltest_en pin to 1, then the select_jtag_input is also under ATPG control and the input path can also be tested.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-45. Boundary Scan Cell as Observation and Control in Logic Test Modes

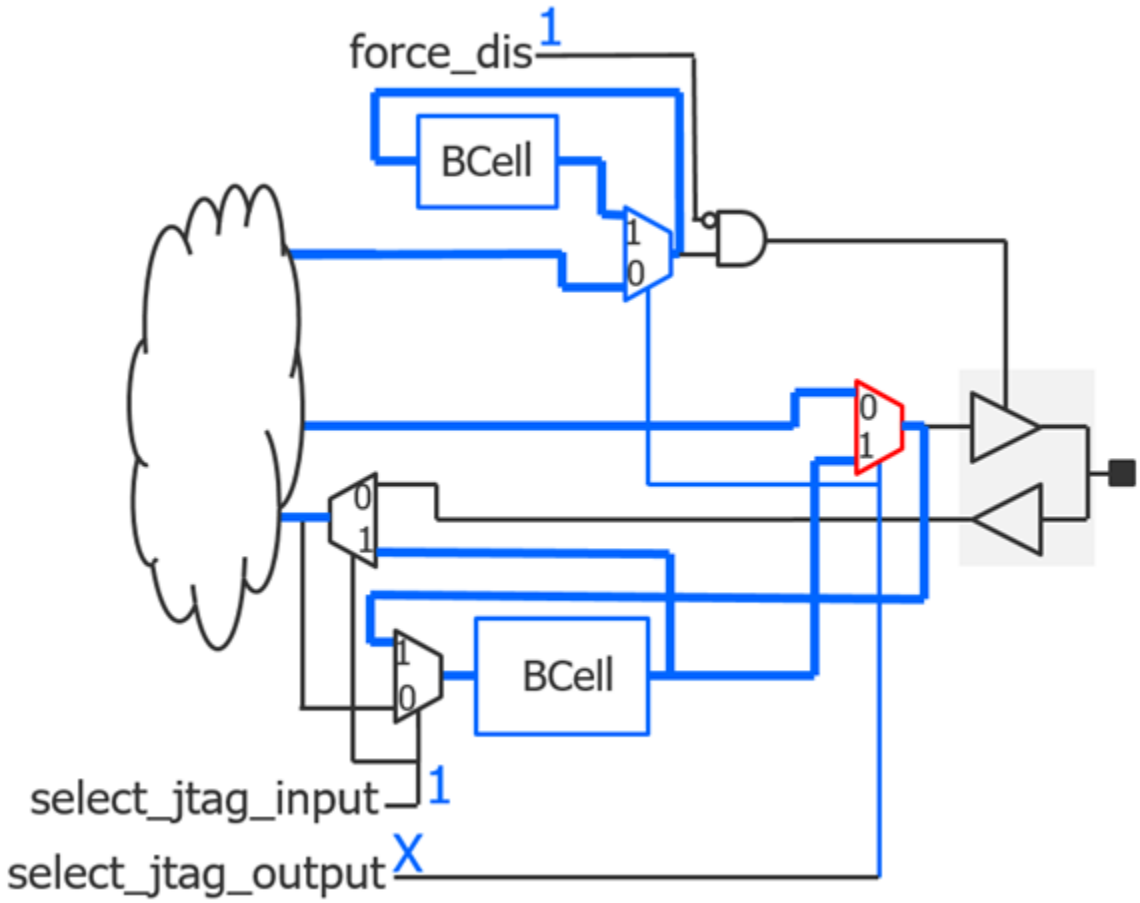


Figure 13-46 shows the same boundary scan cell shown in Figure 13-44 but with auxiliary logic (colored green) added for both the input and output directions. The green multiplexer is inserted on the test side of the JTAG multiplexer (shown in red) rather than in the functional path. The aux_in_en and aux_out_en signals are mutually exclusive, and the boundary scan cell is in functional mode when they are both low.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-46. Adding Auxiliary Input and Output Logic in Bscan Cell

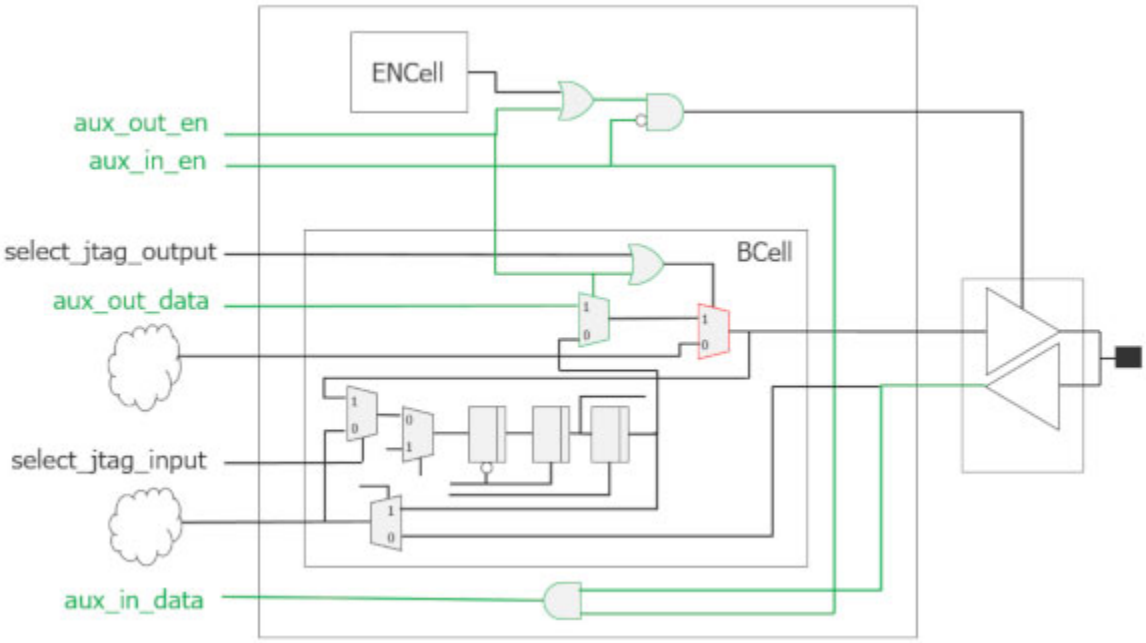


Figure 13-47 shows aux_out_en asserted to 1 to enable the auxiliary output data path. The two multiplexers are switched to select the aux_out_data value, and the enable of the output pad is forced on. When aux_out_en is high, the boundary scan cell is not able to observe the value coming from the core. This is why the add_dft_modal_connections command gates the aux_out_en signals with “scan_en” such that it is active only in shift mode. In capture mode, the aux_out_data is not needed, and instead the boundary scan cell is used to observe the functional core output. The capture mode is shown in Figure 13-48.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-47. Path Used When AuxOut Logic Activated

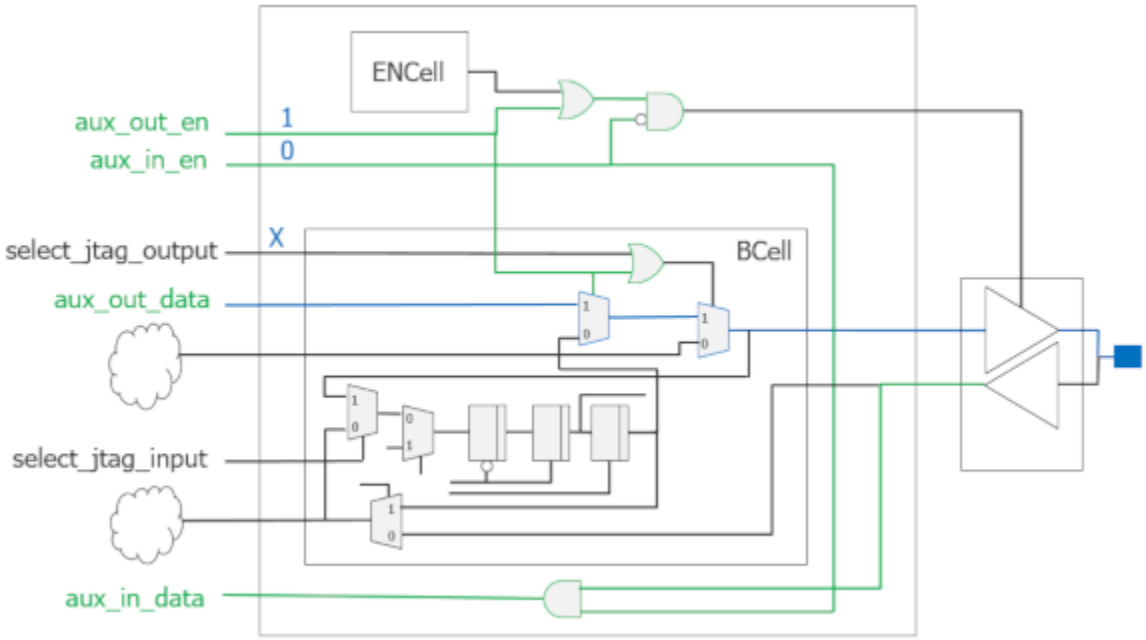


Figure 13-48. Forcing aux_out_en Low in Capture Mode to Observe Core Output

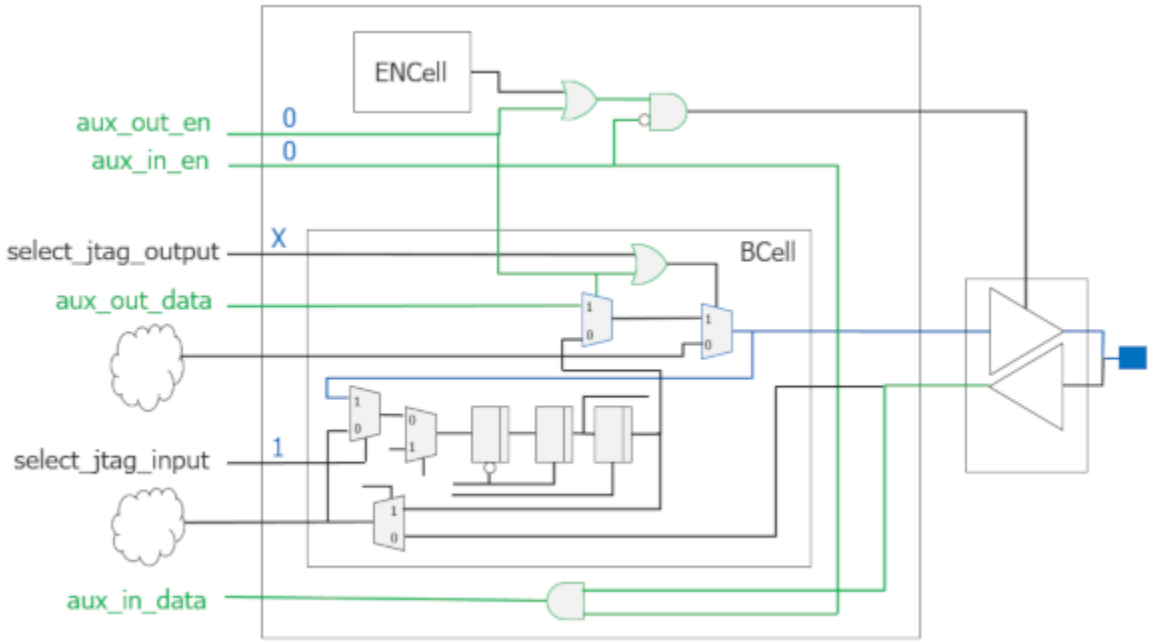


Figure 13-49 shows the I/O pad used as an auxiliary input during ATPG. The boundary scan cell is used to control the signal going to the core, while the input buffer provides the value to the auxiliary in data. This scheme is useful to reuse functional pins for the scan_en, test_clock, and edt_update DFT signals without losing ATPG fault coverage of those inputs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-49. Using AuxIn and Controlling to Core Signal in ATPG

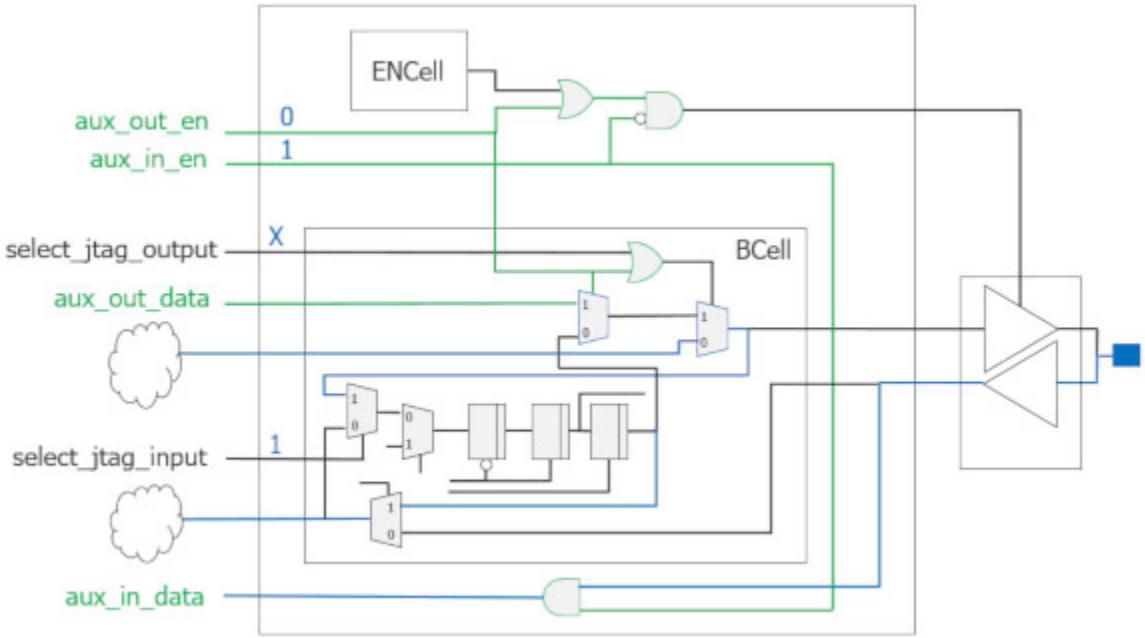
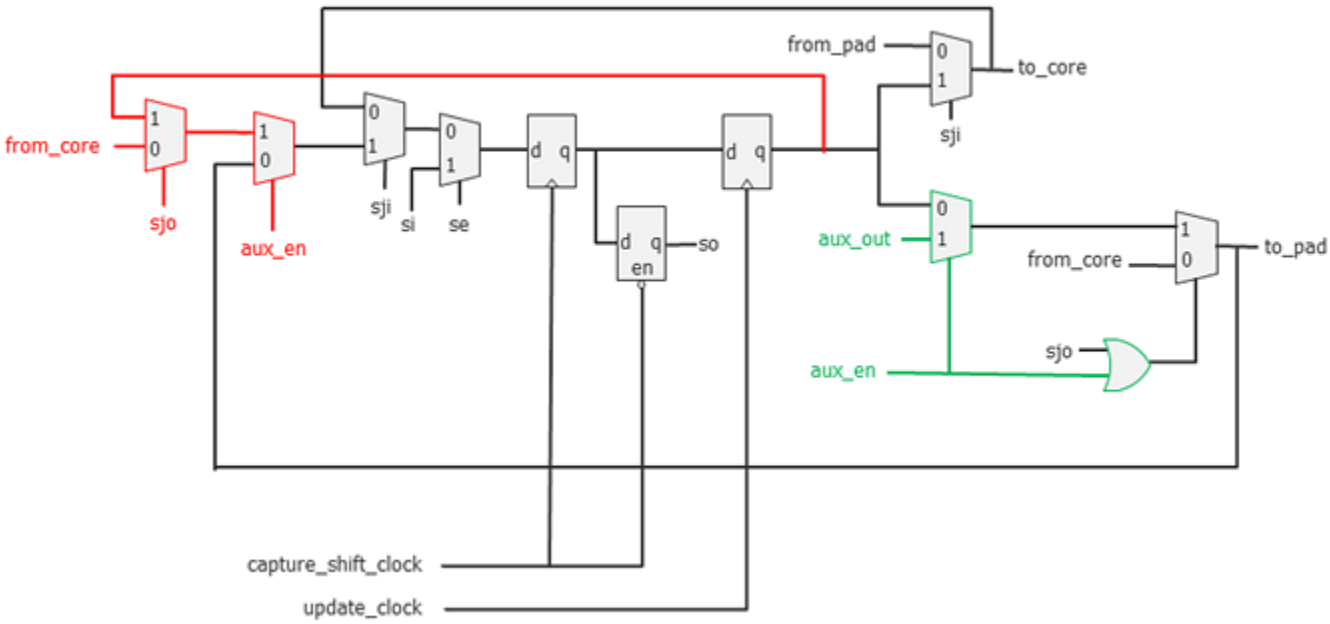


Figure 13-50 shows a bidirectional BoundaryScan cell with an auxiliary output and logic test support. The muxes depicted in red prevent capturing the from_core signal during logic test mode so that the from_core signal can be connected to DFT hardware for logic test.

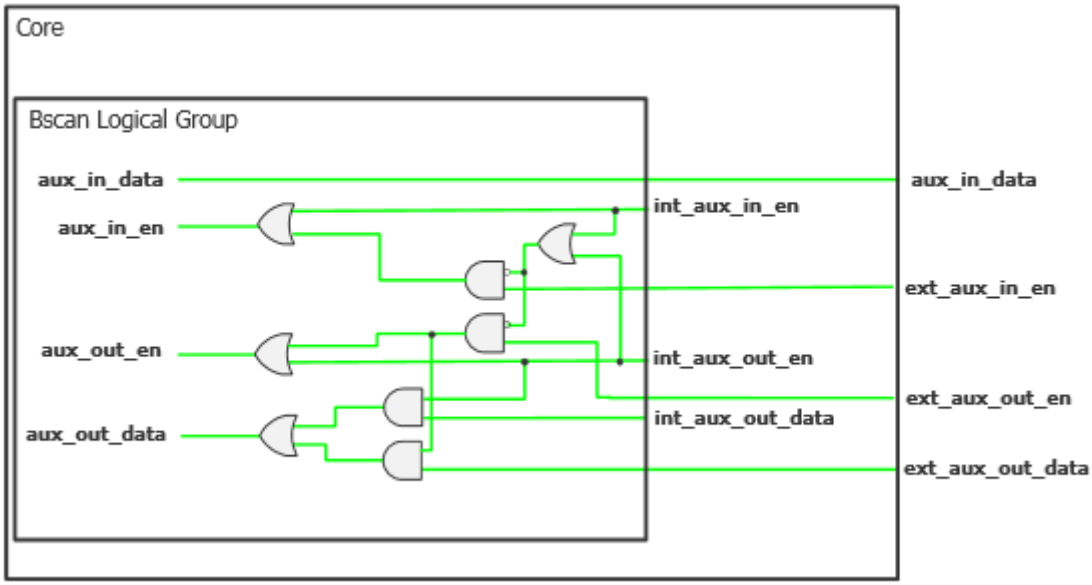
Figure 13-50. Bidirectional Port With AuxOut



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-51 shows the extra logic that is added when the same port is used for both internal and external auxiliary data in the EmbeddedBoundaryScan/AuxiliaryInputOutputPorts wrapper. In the output direction, two data_out input pins are created on the logical grouping module. One of those inputs is used to make auxiliary output connections for elements found within the block and the other input is connected to an input of the block to make auxiliary output connections for elements found outside the block. In the input direction, the same data output port on the logical grouping module is used for providing auxiliary data inputs to elements located inside and outside the block. Both directions have internal and external enable pins. The internal pins are used by the elements inside the blocks and the external pins are used by the elements outside the blocks. For example, if you call add_dft_modal_connections inside the block, it makes connections to the internal pins; and if you call add_dft_modal_connections outside the block, it makes connections to the external pins.

Figure 13-51. Extra Logic Added When a Port is Used Both Internally and Externally



Arguments

- auxiliary_input_ports : *port_name_pattern*, ... ;

  A property that specifies a list of port name patterns to define a group of input or inout ports as auxiliary input ports. The same inout port can be both an input and an output auxiliary port. The auxiliary inputs are left tied on the logic group module to be used by test logic found inside the chip.

  The *port_name_pattern* values can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional, and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- auxiliary_output_ports : *port_name_pattern*, ... ;

  A property that specifies a list of port name patterns that defines a group of output or inout ports as auxiliary output ports. The same inout port can be both an input and an output auxiliary port. The auxiliary output ports are left dangling on the logic group module to be used by test logic found inside the chip.

  The *port_name_pattern* can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

- internal_auxiliary_input_ports : *port_name_pattern*, ... ;

  A property that specifies a list of port name patterns to define a group of input or inout ports as internal auxiliary input ports. The same inout port can be both an input and an output auxiliary port. An internal auxiliary input port differs from an external one in how the auxiliary ports on the boundary scan logical group module are connected. When internal, they are left dangling on the logic group module to be used by test logic found inside the block. When external, the ports are connected to primary input and output ports of the block to be used by test circuit outside the block.

  The *port_name_pattern* values can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

- internal_auxiliary_output_ports : *port_name_pattern*, ... ;

  A property that specifies a list of port name patterns that defines a group of output or inout ports as internal auxiliary output ports. The same inout port can be both an input and an output auxiliary port. An internal auxiliary output port differs from an external one in how the auxiliary ports on the boundary scan logical group module are connected. When internal, they are left tied on the instance of the logic group module to be used by test logic found inside the block. When external, the ports are connected to primary input ports of the block to be used by test circuit outside the block.

  The *port_name_pattern* can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

- external_auxiliary_input_ports : *port_name_pattern*, ... ;

  A property that specifies a list of port name patterns to define a group of input or inout ports as external auxiliary input ports. The same inout port can be both an input and an output auxiliary port. An internal auxiliary input port differs from an external one in how the auxiliary ports on the boundary scan logical group module are connected. When internal, they are left dangling on the instance of the logic group module to be used by test logic found inside the block. When external, the ports are connected to primary output ports of the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

block to be used by test circuit outside the block. The ports on the block follow the naming specified by the Interface/auxiliary_* properties.

The *port_name_pattern* can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

- external_auxiliary_output_ports : *port_name_pattern*, ... ;

  A property that specifies a list of port name patterns that defines a group of output or inout ports as external auxiliary output ports. The same inout port can be both an input and an output auxiliary port. An internal auxiliary output port differs from an external one in how the auxiliary ports on the boundary scan logical group module are connected. When internal, they are left tied on the instance of the logic group module to be used by test logic found inside the block. When external, the ports are connected to primary input ports of the block to be used by test circuit outside the block. The ports on the block follow the naming specified by the Interface/auxiliary_* properties.

  The *port_name_pattern* can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

Examples

The following example defines the GPIO00 to GPIO31 ports as both auxiliary input and output ports. The first 16 are usable by test logic that is both internal and external to the block. The last 16 are only used by test logic that is external to the block. The full flexibility is preserved such that each test mode is able to use <*N*> of them as auxiliary output ports and the rest as auxiliary inputs, where <*N*> can be different for each test mode.

```
DftSpecification(my_chip,rtl) {
  EmbeddedBoundaryScan {
    AuxiliaryInputOutputPorts {
      internal_auxiliary_input_ports  : GPIO%2d[0:15] ;
      internal_auxiliary_output_ports : GPIO%2d[0:15] ;
      external_auxiliary_input_ports  : GPIO%2d[0:31] ;
      external_auxiliary_output_ports : GPIO%2d[0:31] ;
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

EnableGroups

Specifies the creation of enable groups using either the ports in the fanout of a functional enable signal or a specified port list.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan | EmbeddedBoundaryScan {
    EnableGroups {
      EnableGroup(name) {
        capture_core_signal     : on | off ;
        internal_enable_signal : pin_or_net_name ;
        port_list               : port_name_pattern, ... ;
        insert_before_port      : port_name ;
        insert_after_port       : port_name ;
      }
    }
  }
}
```

Description

A wrapper that specifies the creation of enable groups using either the ports in the fanout of a functional enable signal or a specified port list. The properties insert_before_port and insert_after_port are used to specify the location of the enable cells associated with the group. The property capture_core_signal is used to prevent the cell from capturing the core signal during INTEST or during logictest when the boundary scan is used as isolation.

Arguments

- capture_core_signal : on | off ;

  A property that specifies if the enable boundary-scan cells associated with the group are allowed to capture the core signal during INTEST or during the capture pulse when you reuse it for logictest isolation. See the max_segment_length_for_logictest property in the BoundaryScan wrapper for information about the logictest usage. When specified to off, the enable boundary-scan cell captures itself instead of capturing the core signal. This may be useful if you are using LogicBIST and the functional enable is X during the logictest mode.

- internal_enable_signal : *pin_or_net_name* ;

  A property that references an internal signal inside the chip that directly controls the enable pin of a group of output pad cells. Specifying this enable signal is equivalent to listing all the ports controlled by that enable signal in the port_list property. The specification of the internal_enable_pin and port_list properties are mutually exclusive. If the number of output pad cells controlled by the enable signal is larger than the specified outputs_per_enable_cell property, the fanout of the enable cell is split into smaller groups to meet the specified outputs_per_enable_cell value. One enable boundary-scan cell is added per group.

- port_list : *port_name_pattern*, ... ;

  A property that associates a group of output or inout ports with a given enable group. The *port_name_pattern* values can include an asterisk "*" as a wildcard and %#d[lb:rb] as a

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

counter. The # entry is optional and, when specified, is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

The specification of the internal_enable_pin and port_list properties are mutually exclusive. If the number of output pad cells specified in the list is larger than the specified outputs_per_enable_cell value, the list is split into smaller groups to meet the specified outputs_per_enable_cell value. One enable boundary-scan cell is added per group.

You do not need to specify an enable group for all output and inout ports. For the tri-state inout and output ports not explicitly associated to an enable group, one or more additional enable groups is inferred. The inferring is done to match the common functional enable signal sourcing a group of pads. A given group can further be split to satisfy the specified outputs_per_enable_cell value described in the following sections:

- BoundaryScan — outputs_per_enable_cell property
- EmbeddedBoundaryScan — outputs_per_enable_cell property

- insert_before_port : *port_name* ;

  A property that specifies the position of the enable boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the insert_before_port property is mutually exclusive with the specification of the insert_after_port property. If neither the insert_before_port nor insert_after_port properties are specified, the enable boundary-scan cell is inserted before the first boundary cell that is associated with a port controlled by the enable cell.

- insert_after_port : *port_name* ;

  A property that specifies the position of the enable boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the insert_before_port property is mutually exclusive with the specification of the insert_after_port property. If neither the property insert_before_port nor insert_after_port properties are specified, the enable boundary-scan cell is inserted before the first boundary cell that is associated with a port controlled by the enable cell.

Examples

The following example defines three enable groups. The first one is associated with the ports in the fanout of the functional enable signal “core_i/en”. The second one is associated with all ports matching the pattern “OUT*”. The third group is associated with all other remaining ports as it does not specify either the internal_enable_signal nor the port_list property.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(my_chip,rtl) {
  BoundaryScan {
    EnableGroups {
      EnableGroup(g1) {
        internal_enable_signal : core1/en ;
      }
      EnableGroup(g2) {
        port_list : OUT* ;
      }
      EnableGroup(g3) {
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

InternalBScanCells

Specifies the insertion of internal boundary-scan cells that are not associated with ports but instead are used to control and/or observe internal nodes inside the design.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan | EmbeddedBoundaryScan {
    InternalBScanCells {
      InternalBScanCell(name) {
        insert_before_port : port_name ;
        insert_after_port  : port_name ;
        safe_value         : 0 | 1 | X ;
        type               : control | observation | both ;
        connection         : pin_or_net_name ;
        multiplexing       : on | off | auto ;
      }
    }
  }
}
```

Description

A wrapper that specifies the insertion of internal boundary-scan cells that are not associated with ports but instead are used to control and/or observe internal nodes inside the design.

Arguments

- *name*

  A string that uniquely identifies the internal boundary-scan cell in the BSDL file. The string starts with a letter and may be followed by any numbers of letters, numbers, or underscores. The string must be unique across all InternalBScanCell wrappers. The name of internal boundary-scan cells are mapped to the cell position inside the boundary scan chain using an attribute called TESSENT_INTERNAL_CELL_LABELS as shown in this example. The two InternalBScanCell wrappers have the names “my_name1” and “my_name2”.

  ```
  -- BSDL local extensions
  attribute TESSENT_INTERNAL_CELL_LABELS: BSDL_EXTENSION;
  attribute TESSENT_INTERNAL_CELL_LABELS of TOP: entity is
  -- Label  Cell
  "((my_name1 , 36  )," &
  " (my_name2 , 30  ))";
  ```

- insert_before_port : *port_name* ;

  A property that specifies the position of the internal boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the insert_before_port property is mutually exclusive with the specification of the insert_after_port property. If neither the insert_before_port nor insert_after_port properties are specified, the internal boundary-scan cell is inserted at the end of the boundary scan chain near scan_out in the order they were specified. The location

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

of the internal boundary-scan cell is reflected in the BSDL file using the TESSENT_INTERNAL_CELL_LABELS attribute.

- insert_after_port : *port_name* ;

  A property that specifies the position of the internal boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the insert_after_port property is mutually exclusive with the specification of the insert_before_port property. If neither the property insert_before_port nor insert_after_port properties are specified, the internal boundary-scan cell is inserted at the end of the boundary scan chain near scan_out in the order they were specified. The location of the internal boundary-scan cell is reflected in the BSDL file using the TESSENT_INTERNAL_CELL_LABELS attribute.

- safe_value : 0 | 1 | X ;

  A property that specifies a safe value for the internal boundary-scan cell. This value is reflected into the BSDL file. If a non-X safe value is specified, the BoundaryScan signoff and manufacturing pattern set never loads a different value into that register during any of its RunTest specifications.

- type : control | observation | both ;

  A property that specifies if the internal boundary-scan cell is to be used to control and/or observe an internal node.Table 13-7 shows the hardware inserted into the design based on the value of the type and multiplexing property.

- connection : *pin_or_net_name* ;

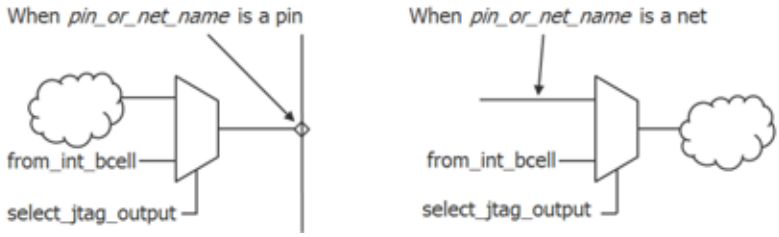
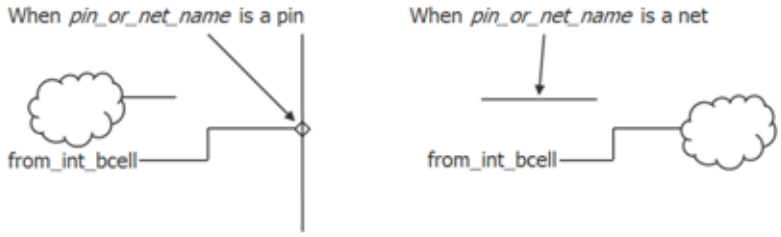
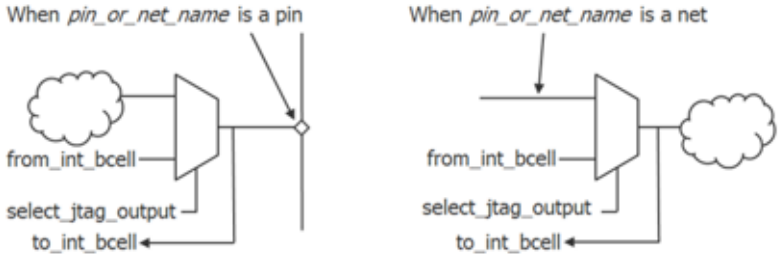
  A property that specifies the name of a pin or a net to control and/or observe. The *parent_instance* of *pin_or_net_name* is auto-uniquified if not unique.

- multiplexing : on | off | auto ;

  A property that is only relevant when the property type is set to “control” or “both”. A multiplexer is inserted as shown in Table 13-7. When the value is “auto”, the multiplexer is inserted only when the has_functional_source attribute of the specified *pin_or_net_name* is set to “true”. See Pin and Net for more information about this attribute.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-7. Connections To and From Internal Boundary Scan Cells

type	description
control	If multiplexing is specified as “on” or “auto” and the has_functional_source attribute for *pin_or_net_name* is set to “true”, the driver is intercepted with a multiplexer located in the *parent_instance* of the *pin_or_net_name*. A net is intercepted at its source while a pin is intercepted at its destination. The internal boundary-scan cell captures its own output.  If multiplexing is specified as “off” or “auto” and the has_functional_source attribute for *pin_or_net_name* is set to “false”, the original driver is disconnected and replaced by the signal from the internal boundary-scan cell. The internal boundary-scan cell captures its own  output.
observation	The specified *pin_or_net_name* is connected to the input of the internal boundary-scan cell.
both	If multiplexing is specified as “on” or “auto” and the has_functional_source attribute for pin_or_net_name is set to “true”, the driver is intercepted with a multiplexer located in the *parent_instance* of the *pin_or_net_name*. A net is intercepted at its source while a pin is intercepted at its destination. The observation net is taken after the multiplexer. The functional source of the pin_or_net_name is observable when select_jtag_output is  low. If multiplexing is specified as “off” or “auto” and the has_functional_source attribute for *pin_or_net_name* is set to “false”, the original driver is disconnected and replaced by the signal from the internal boundary-scan cell. The internal boundary-scan cell captures its own output.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example creates an internal boundary-scan cell that is used to control the u1/pll_reset signal using the boundary scan chain. The specified safe value of 0 ensures this cell is never loaded with a 1 during any of the RunTest in the PatternsSpecification or with any board test patterns created from the BSDL file.

```
DftSpecification(module_name,id) {
  BoundaryScan | EmbeddedBoundaryScan {
    InternalBScanCells {
      InternalBScanCell(mycell) {
        insert_before_port : PortA ;
        safe_value         : 0 ;
        type               : control ;
        connection         : u1/pll_reset ;
        multiplexing       : auto ;
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

InternalBScanSegments

Specifies the position of an existing internal boundary scan segment in the boundary scan chain.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan | EmbeddedBoundaryScan {
    InternalBScanSegments {
      InternalBScanSegment(instance_name) {
        insert_before_port      : port_name ;
        insert_after_port       : port_name ;
        logical_group_name      : logical_group_name ;
      }
    }
  }
}
```

Description

A wrapper that is used to specify the position of an existing internal boundary scan segment in the boundary scan chain. When the BScanSegment contains cells associated with a port of the design, the position of the BScanSegment and its *logical_group_name* is taken from the position and the logical group of the first port associated with the BScanSegment in the pin_order_file. In the case of pure internal boundary scan segments, their position and logical grouping name cannot be extracted from associated ports because they are not associated with ports. The logical group name is not used to create logical grouping modules for existing boundary scan segments because they are already grouped into modules. Instead, they are used to refer to the segments that use the bypass_logical_group inside the BondingConfigurations wrapper.

This wrapper is automatically created when the create_dft_specification command is invoked if a BScanSegment description is found for a module. You must delete this wrapper prior to invoking process_dft_specification if you do not want it to be part of the boundary scan chain. Refer to the "set_design_sources -format bscan_lib" command description for information about automatically finding the BScanSegment files without having to explicitly read them in using the read_core_descriptions command. You specify the module matching options using the set_module_matching_options command.

Arguments

- insert_before_port : *port_name* ;

  A property that specifies the position of the internal boundary segment relative to the position of the boundary-scan cell associated with a port. The specification of the insert_before_port is mutually exclusive with the specification of the insert_after_port property. When neither the insert_before_port nor insert_after_port properties are specified, the boundary scan segment is placed at the end of the boundary scan chain near the scanout. You can only specify this property if the pin_order_file property is used inside the BoundaryScan wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- insert_after_port : *port_name* ;

  A property that specifies the position of the internal boundary segment relative to the position of the boundary-scan cell associated with a port. The specification of the insert_before_port is mutually exclusive with the specification of the insert_after_port property. When neither the insert_before_port nor insert_after_port properties are specified, the boundary scan segment is placed at the end of the boundary scan chain near the scanout. You can only specify this property if the pin_order_file property is used inside the BoundaryScan wrapper.

- logical_group_name : *logical_group_name* ;

  A property used to associate a *logical_group_name* to the existing internal boundary scan segment. This name is not used to create a logical grouping module as the existing boundary scan segment is already grouped into a module. It is used instead to refer to the segment using the bypass_logical_group property inside the BondingConfigurations wrapper.

Examples

The following example defines a design instance as an internal boundary scan segment. It is given a logical group name so that it can be bypassed in some bounding options. A BscanSegment definition must exist for the module associated with instance u1/u2.

```
DftSpecification(module_name,id) {
  BoundaryScan {
    InternalBScanSegments {
      InternalBScanSegment(u1/u2) {
        logical_group_name      : seg1 ;
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BoundaryScan

Specifies the boundary scan chain to build and optionally insert into the design.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan {
    ijtag_host_interface              : id ;
    pin_order_file                    : filename ;
    interface_parent_instance         : parent_instance ;
    create_ac_control                 : auto | on    // default: auto
    outputs_per_enable_cell           : int ;        // default: 16
    max_segment_length_for_logictest : int | unlimited | off ;
    tristate_enable_non_contacted_test_support : on | off ;
    tck_period                        : period ;     // default 100ns
    default_cell_on_bidir_port : combined | separated ;
    handle_logictest_segments_during_scan_insertion : on | off | auto ;
    ImplementationOptions {
    }
    BoundaryScanCellOptions {
    }
    LogicalGroups {
    }
    ACModeOptions {
    }
    AuxiliaryInputOutputPorts {
    }
    EnableGroups {
    }
    InternalBScanCells {
    }
    InternalBScanSegments {
    }
    BondingConfigurations {
    }
    UserInstructions {
    }
  }
}
```

Description

BoundaryScan is a wrapper that specifies the boundary scan chain to build and optionally insert into the design.

This wrapper can only be specified once in a given DftSpecification wrapper. As illustrated in the syntax above, the BoundaryScan wrapper uses the ijtag_host_interface property to specify the host scan interface that drives the boundary scan register. This can be a TAP controller specified in the IjtagNetwork wrapper of the same DftSpecification, or a TAP instance already existing in the design. The TAP controller may have been inserted using a prior execution of the process_dft_specification command, or it may be a third-party TAP controller manually inserted into your design. If you want to use a third-party TAP controller, you must have a valid

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL description for it, and it must have the port functions described in section "Requirements on a TAP to be Usable for BoundaryScan" on page 4154.

The tool identifies the boundary scan register during extraction with a set of annotations to the STIL pattern set. See "Symbolic Variables Specific to Boundary Scan Patterns" in the Tessent IJTAG User's Manual for complete information.

If you have existing boundary scan segments that are described with BoundaryScan wrappers (see "Tessent Core Description" on page 4793), the tool automatically recognizes these and includes them in the boundary scan chain. The legacy *.lvbscan* file format is also natively supported and translated into the current BoundaryScan wrapper syntax when it is loaded into Tessent Shell. Refer to the set_module_matching_options and "set_design_sources -format bscan" commands to learn how to have them automatically searched and loaded. You can also use the read_core_descriptions command to load them explicitly.

Note

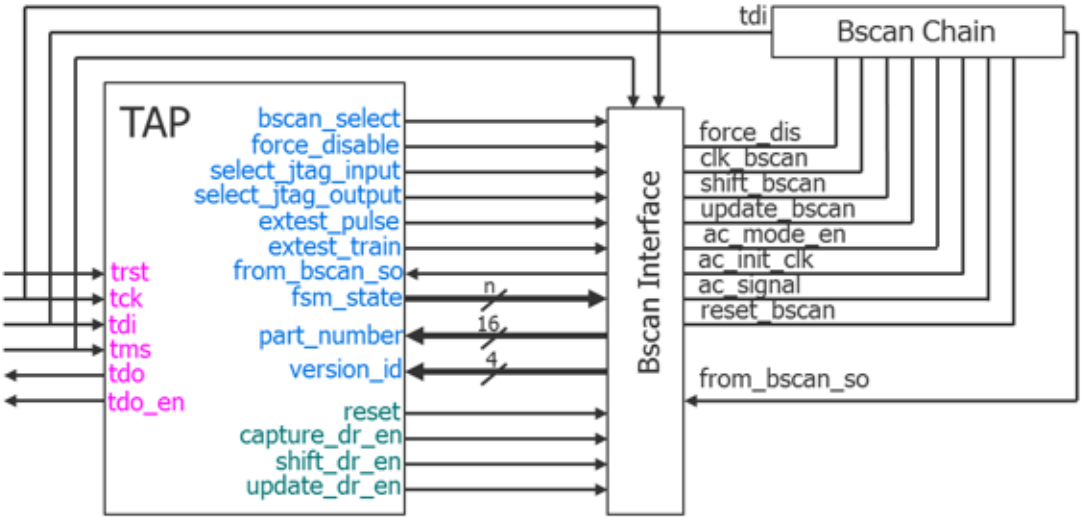
If you want to insert BoundaryScan and the TAP controller, and connect the TAP to internal pins, follow the instructions in Example 2 of the IjtagNetwork wrapper description.

Requirements on a TAP to be Usable for BoundaryScan

You can use any TAP controller as the access point to the boundary scan as long as it is described in ICL and contains a minimum set of port functions. As with any ICL module that is handed off, the port names used in the ICL file must exactly match the port names of the corresponding Verilog or VHDL module. Figure 13-52 illustrates the connections made to and from a TAP controller to implement boundary scan. The blue ports are dedicated to Boundary scan. Most of the ports are required. You only need the extest_pulse and extest_train ports if your chip contains pads that require the IEEE 1149.6 ac modes. You only need the fsm_state port if your chip contains pads that require IEEE 1149.6 ac modes or if the chip has tristate_enable_non_contacted_test_support testing enabled. You only need the part_number_code and version_code ports if you want to use the part_number_code and version_code properties of the BondingConfigurations wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-52. Needed Resources From Third-Party TAP Controller



The remainder of this section lists the ports that must exist on the third-party TAP controller in order for it to be usable by Tessent BoundaryScan, the conditions that make them required, and the required ICL descriptions in the ICL files. You can use the following dofile to create a TAP controller and look at its ICL as an example:

```
set_context dft -rtl
read_config_data -from_string {
  DftSpecification(ex,rtl) {
    IjtagNetwork {
      HostScanInterface(tap) {
        Tap(main) {
          HostBscan {
          }
          HostIjtag(1) {
          }
          DeviceIDRegister {
          }
        }
      }
    }
  }
}
process_dft_specification -no_insertion
```

The ICL of the TAP must include the following attributes: tessent_instruction_reg, tessent_bypass_reg, and tessent_device_id_reg. The value of the attribute is the name of the ScanRegister in ICL that is associated with the instruction, bypass, and dev_id registers. The tessent_device_id_reg attribute is optional and only needed if your TAP supports the device id feature. The ScanRegister referenced by the tessent_instruction_reg attribute must have a RefEnum pointing to an Enum table that defines the listed Opcodes. The IDCODE opcode is optional and only needed if the TAP has a device id register. The EXTEST_PULSE and EXTEST_TRAIN opcodes are only needed if IEEE 1149.6 is used. The tessent_use_in_dft_specification attribute is used to tell the create_dft_specification command

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

to ignore this module and not infer TDRs to control and observes the unconnected DataIn and DataOut ports of your TAP.

```
Attribute tessent_instruction_reg = “instruction" ;
Attribute tessent_bypass_reg = “bypass" ;
Attribute tessent_device_id_reg = “devid” ;
Attribute tessent_use_in_dft_specification = "false";
ScanRegister instruction[4:0] {
  CaptureSource  5'bXXX01 ;
  ResetValue     5'b00000 ;
  ScanInSource   tdi ;
  RefEnum        instruction_opcodes ;
}
Enum instruction_opcodes {
  BYPASS          = 5'bx0000 ;
  CLAMP           = 5'bx0001 ;
  EXTEST          = 5'bx0010 ;
  EXTEST_PULSE    = 5'bx0011 ;
  EXTEST_TRAIN    = 5'bx0100 ;
  IDCODE          = 5'bx0101 ;
  INTEST          = 5'bx0110 ;
  SAMPLE          = 5'bx0111 ;
  PRELOAD         = 5'bx0111 ;
  HIGHZ           = 5'bx1000 ;
}
```

If your TAP has the part_number_code and/or version_code ports, they must be part of the CaptureSource of the ScanRegister that is referenced by the tessent_device_id_reg attribute as shown here. In this example, man_id is a parameter that defines the 11-bit manufacturing id constant.

```
DataInPort version_code[3:0] {
  Attribute connection_rule_option = "allowed_no_source";
}
DataInPort part_number_code[15:0] {
  Attribute connection_rule_option = "allowed_no_source";
}
ScanRegister devid[31:0] {
  ScanInSource tdi ;
  CaptureSource version_code[3:0], part_number_code[15:0], $man_id, 1'b1 ;
}
```

The BoundaryScan host port must be described in a ScanInterface wrapper as shown here. The actual port names in your TAP may differ from these names. The ScanInterface is automatically recognized by the create_dft_specification command if it is named host_bscan; otherwise, you must use the -existing_bscan_host_scan_in option to define the name.

```
ScanInterface host_bscan {
  Port from_bscan_so ;
  Port bscan_select ;
  Port capture_dr_en ;
  Port shift_dr_en ;
  Port update_dr_en ;
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following list presents the port functions that may exist on the third-party TAP, describes when they are needed, and shows how to describe them in the ICL file. The examples illustrate the use of the connection_rule_option attribute to indicate to the ICL extraction process that it must allow the bscan host port to not be connected to a boundary scan chain. The ICL description of the boundary scan interface module must be connected to a TAP. ICL extraction reports errors if the boundary scan interface module is present but incorrectly connected to a TAP. However, having a TAP controller with no boundary scan chain connected to it is allowed.

- bscan_en

  This function is required. It is a signal that is high during the EXTEST, SAMPLE, PRELOAD, INTEST, HIGHZ, EXTEST_TRAIN, and EXTEST_PULSE opcodes. The source is a LogicSignal that properly describes the decoding of this signal from the instruction register bits. It must be described in the ICL file as shown here.

```
ToSelectPort bscan_en   {
  Source bscan_en_int;
  Attribute connection_rule_option = "allowed_no_destination";
}
LogicSignal bscan_en_int {
  IR == EXTEST         ||
  IR == SAMPLE         ||
  IR == PRELOAD        ||
  IR == INTEST         ||
  IR == HIGHZ          ||
  IR == EXTEST_TRAIN   ||
  IR == EXTEST_PULSE;
}
```

- force_disable

  This function is required. It is a signal that is active during the HIGHZ opcode. The source is a LogicSignal that properly describes the decoding of this signal from the instruction register bits. It must be described in the ICL file as shown here.

```
DataOutPort force_disable   {
  Source force_disable_int;
  Attribute connection_rule_option = "allowed_no_destination";
  Attribute tessent_bscan_function  = "force_disable";
}
```

  The value of the attribute tessent_bscan_function is “force_disable_inv” if your port is active low.

- select_jtag_input

  This function is required. It is a signal that is active during the INTEST opcode. The source is a LogicSignal that properly describes the decoding of this signal from the instruction register bits. It must be described in the ICL file as shown here.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DataOutPort select_jtag_input {
  Source select_jtag_input_int;
  Attribute connection_rule_option = "allowed_no_destination";
  Attribute tessent_bscan_function = "select_jtag_input";
}
```

The attribute tessent_bscan_function is “select_jtag_input_inv” if your port is active low.

- select_jtag_output

  This function is required. It is a signal that is active during the EXTEST, HIGHZ, CLAMP, EXTEST_TRAIN and the EXTEST_PULSE opcodes. The source is a LogicSignal that properly describes the decoding of this signal from the instruction register bits. It must be described in the ICL file as shown here.

```
DataOutPort     select_jtag_input {
  Source select_jtag_input_int;
  Attribute connection_rule_option = "allowed_no_destination";
  Attribute tessent_bscan_function = "select_jtag_output";
}
```

The attribute tessent_bscan_function is “select_jtag_output_inv” if your port is active low.

- extest_pulse

  This function is optional and only needed if the chip includes pads requiring the IEEE 1149.6 ac tests. It is a signal that is active during the EXTEST_PULSE opcode. The source is a LogicSignal that properly describes the decoding of this signal from the instruction register bits. It must be described in the ICL file as shown here.

```
DataOutPort     extest_pulse {
  Source extest_pulse_int;
  Attribute connection_rule_option = "allowed_no_destination";
  Attribute tessent_bscan_function = "extest_pulse";
}
```

The attribute tessent_bscan_function is “extest_pulse_inv” if your port is active low.

- extest_train

  This function is optional and only needed if the chip includes pads requiring the IEEE 1149.6 tests. It is a signal that is active during the EXTEST_TRAIN opcode. The source is a LogicSignal that properly describes the decoding of this signal from the instruction register bits. It is must be described in the ICL file as shown here.

```
DataOutPort     extest_train {
  Source extest_pulse_int;
  Attribute connection_rule_option = "allowed_no_destination";
  Attribute tessent_bscan_function = "extest_train";
}
```

The attribute tessent_bscan_function is “extest_train_inv” if your port is active low.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- fsm_state

  This function is optional and only needed if the chip includes pads requiring the IEEE 1149.6 tests or if tristate_enable_non_contacted_test_support : on is specified. It is a bus that represents the TAP finite state machine (fsm_state) encoding. The fsm_state signals must change on the rising of TCK, and they must be described in the ICL file as shown here. The name of the Enum can be any arbitrary name as long as it includes the symbols for all 16 fsm state values.

```
DataOutPort     fsm_state[3:0] {
  Attribute connection_rule_option = "allowed_no_destination" ;
  Attribute function_modifier = "tap_fsm_state" ;
  RefEnum   state_encoding ;
}
Enum state_encoding {
  test_logic_reset  = 4'b1111 ;
  run_test_idle     = 4'b1100 ;
  select_dr         = 4'b0111 ;
  capture_dr        = 4'b0110 ;
  shift_dr          = 4'b0010 ;
  exit1_dr          = 4'b0001 ;
  pause_dr          = 4'b0011 ;
  exit2_dr          = 4'b0000 ;
  update_dr         = 4'b0101 ;
  select_ir         = 4'b0100 ;
  capture_ir        = 4'b1110 ;
  shift_ir          = 4'b1010 ;
  exit1_ir          = 4'b1001 ;
  pause_ir          = 4'b1011 ;
  exit2_ir          = 4'b1000 ;
  update_ir         = 4'b1101 ;
}
```

If you want to insert BoundaryScan and the TAP controller and connect the TAP to internal pins, follow the instructions in "Example 2" on page 3930 in the IjtagNetwork wrapper description.

Boundary Scan Interface Module

As shown in Figure 13-52, the boundary scan interface module resides between the TAP controller and the boundary scan chain. The normal logic included in the interface module is shown in black in Figure 13-53 and includes the clock gaters to create the bscan_update_clock and bscan_shift_capture_clock signals used by some boundary scan cells. See the ImplementationOptions wrapper's description section for a description of boundary scan cells using gated clocks. If you are using IEEE 1149.6 boundary scan cells, the interface has more logic that is used to create the IEEE 1149.6 signals but this logic is not shown in Figure 13-53.

The blue logic shown in Figure 13-53 is the logic added in the interface module when the boundary scan chain is re-used during the logic test modes. This feature is enabled by setting the max_segment_length_for_logictest property to an integer or to "unlimited".

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Multiplexers are inserted to segment the boundary scan chains into one or more logic test scan chains as shown in Figure 13-54. The name of the scan_in and scan_out pins created by this feature are documented in the Instrument Dictionary.

The blue module shown in bold in Figure 13-53 contains a scan chain of six flip-flops between ltest_si and ltest_so used to provide ATPG control to the six output ports of the module. This enables the achievement of high APTG fault coverage of the boundary scan logic. The same module also includes a mini-OCC as shown in Figure 13-20 of the Sib wrapper description section.

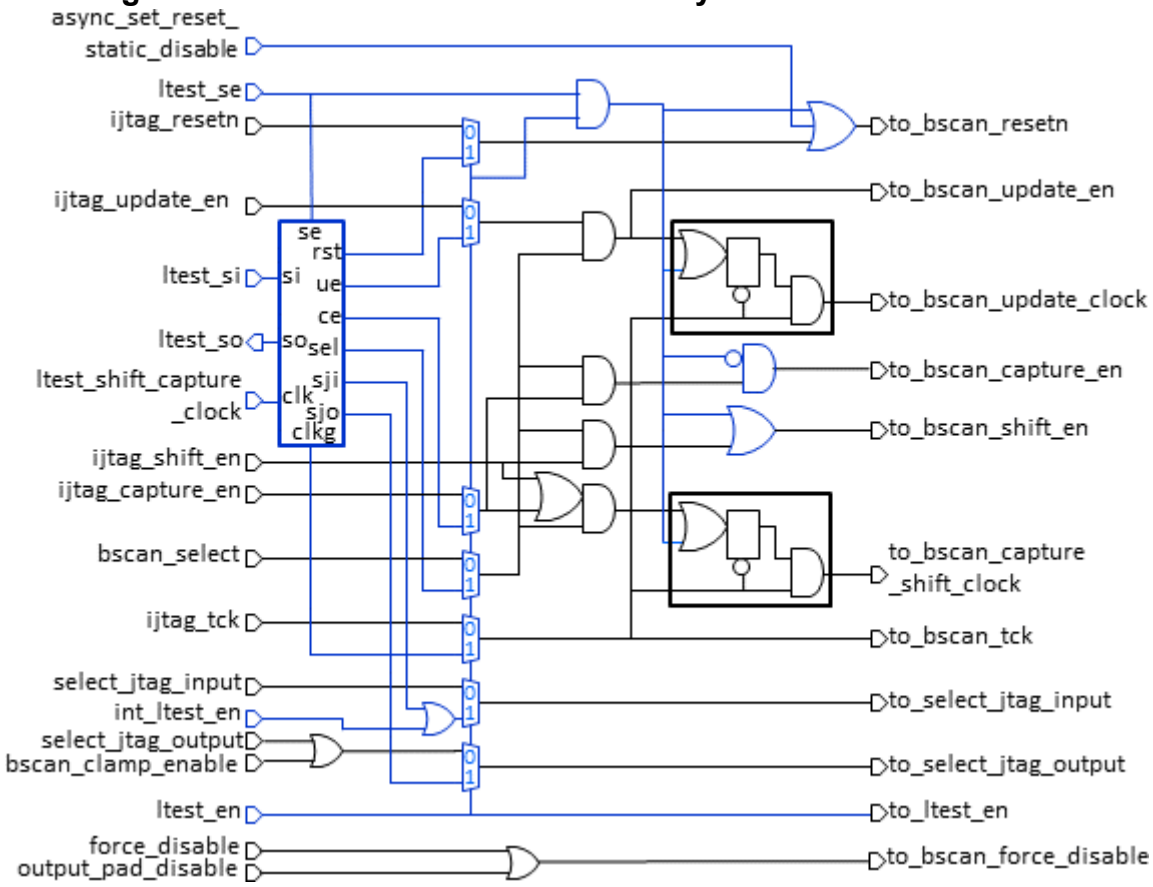
The int_ltest_en input is ORed with the ATPG controlled select_jtag_input signal before being sent to the boundary scan cells. This enables use of the boundary scan cells to isolate the input of the chip from the core input values in order to support low count pin test for multi-site testing. See Figure 13-48 on page 4139 and Figure 13-49 on page 4140 in the AuxiliaryInputOutputPorts section for more details on this feature. You use the "set_static_dft_signal_values int_ltest_en 1" to set int_ltest_en in your test_setup.

The flip-flops in the interface module are tested only during BoundaryScan test. Those in the segmented logical groups (Figure 13-54) are also tested during BoundaryScan test, as well as during logic test if re-use is enabled.

An additional output port on the Boundary Scan Interface Module is to_bscan_pad_sel (not shown). This signal can be used to enable boundary scan mode in a pad. It can be created in the BoundaryScan interface block on each block level, and is not passed down in the hierarchy. It is high during EXTEST, CLAMP, PRELOAD, SAMPLE, EXTEST_PULSE and EXTEST_TRAIN. It is low during BYPASS, IDCODE and all instructions that are unrelated to BoundaryScan.

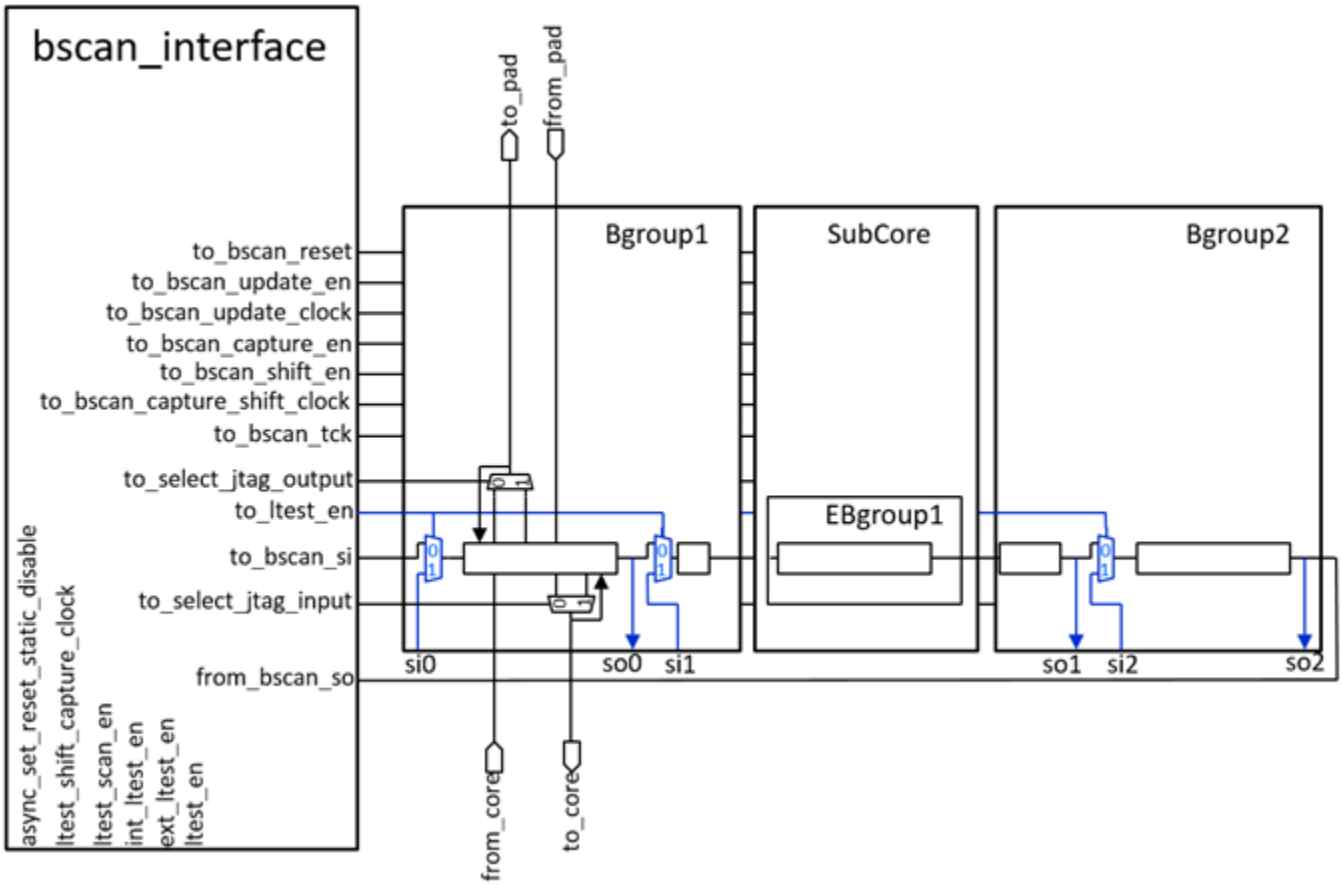
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-53. Schematic of the Boundary Scan Interface Module



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-54. Segmenting the Boundary Scan Chains into Logic Test Chains



Instrument Dictionary

When you have run the process_dft_spec commands, and the BoundaryScan wrapper or the EmbeddedBoundaryScan wrapper was present in the DftSpecification, an instrument dictionary is created to describe what was done on each port. You can access this dictionary using this command:

format_dictionary [get_instrument_dictionary mentor::jtag_bscan::DftSpecification]

Note

The ports section of the instrument dictionary does not list ports that have no pad cells.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The format of the dictionary is the following:

```
ports {
  <port_name> {
    bscan_cell_presence added | existing | none
    bcell_pad_connection_functions {
      from_pad            pin_name
      to_core             pin_name
      from_core           pin_name
      to_core             pin_name
      init_data_dot6      pin_name
      init_data_inv_dot6  pin_name
      test_data_dot6      pin_name
      test_data_inv_dot6  pin_name
      from_sji_mux        pin_name
      to_sji_mux          pin_name
      from_sjo_mux        pin_name
      to_sjo_mux          pin_name
    }
    // class is one of input, output, or inout
    // sub-classes are no_capture_core_signal, sample_only
    // no_sji_mux, no_sjo_mux, clock, dot6
    added_bscan_cell_options {list of class and sub-classes}
    output_during_force_disable 0 | 1 | X | Z
    auxiliary_output_pin pin_name
    auxiliary_output_enable_pin pin_name
    auxiliary_input_pin pin_name
    auxiliary_input_enable_pin pin_name
    ac_cell on | off
    bsdl_pin_name pin_name
    bonding_configurations {list of bonding options}
    enable_group_id ID
    extracted_pad_functions {list of functions and locations}
    output_trace_instance_list {list of pad instances}
    output_trace_mode_list {list of output direction modes}
    input_trace_instance_list {list of pad instances}
    input_trace_mode_list {list of input direction modes}
  }
}
boundary_scan_interface {
  force_disable           pin_name
  select_jtag_input       pin_name
  select_jtag_output      pin_name
  capture_shift_clock     pin_name
  capture_shift_clock_inv pin_name
  bscan_clock             pin_name
  update_clock            pin_name
  select                  pin_name
  reset                   pin_name
  capture_en              pin_name
  shift_en                pin_name
  update_en               pin_name
  scan_in                 pin_name
  scan_out                pin_name
  scan_out_launch_edge    posedge | negedge
  ac_init_clock0          pin_name
  ac_init_clock1          pin_name
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
    ac_signal                pin_name
    ac_mode_en               pin_name
  }
  logic_test_scan_chains {
    chain# {
      length                 integer
      scan_in                port_pin_name
      scan_out               port_pin_name
      scan_in_capture_edge   posedge | negedge
      scan_out_launch_edge   posedge | negedge
    }
  }
  bscan_occ {
    present                       on | off
    static_clock_control          external
    capture_window_size           integer
    interface {
      clock_sequence              {pin_list}
      static_clock_control_mode   pin_name
      capture_en                  pin_name
      scan_en                     pin_name
    }
    internal_nodes {
      clock_gate_output          pin_name
    }
    parent_instance              instance_name
    instance_name                leaf_instance_name
  }
```

The ports section provides information about the boundary scan cell associated with each port, and where those ports get connected on the pad cell. If a port was equipped with Auxiliary muxing logic as requested in the AuxiliaryInputOutputPorts wrapper, you can find the associated pin on the Logical grouping modules in the dictionary. The “boundary_scan_interface” section gives you the pins on the right hand side of the boundary scan interface module, illustrated in Figure 13-52, for all pin functions needed by the boundary scan segments. Finally, the logic_test_scan_chains section is created if you used the max_segment_length_for_logictest property documented below. This section describes all logictest chains created along with their length, scan_in, and scan_out port, and the edge timing for scan_in and scan_out with respect to the shift clock.

See the example below on how to retrieve the dictionary and manipulate the chain data.

Note

The mini OCC is added to the BoundaryScan interface when max_segment_length_for_logictest is an integer or unlimited. For more information about the mini OCC, refer to the Sib command description.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ijtag_host_interface : *id* ;

  A required property that specifies the ScanInterface that is hosting the boundary scan register. The format of the *id* is either “Tap(id)” or “HostScanInterface(id)”. You use Tap(*id*) when the boundary scan host is a Tap controller specified in the IjtagNetwork wrapper of the current DftSpecification wrapper. You use HostScanInterface(*id*) when the host of the boundary scan is a TAP that already exists in the design. The IjtagNetwork must have a HostScanInterface(*id*) wrapper with design_instance and scan_interface properties that identify the TAP and the bscan host interface as shown in “Example 2” on page 3958 in the HostScanInterface/Interface section.

- pin_order_file : *filename* ;

  An optional property that specifies the name of a file that provides an ordered list of ports that the tool uses to order the boundary scan cells. The *filename* can be an absolute or relative path.

  If you have read in a DEF file using the read_def command, the tool extracts the order from the physical placement of the pad cells to minimize the routing of the boundary scan chain. The tool orders the ports starting on the TDO port, followed by the closest neighbor, then its closest neighbor.

  If no LEF/DEF information is loaded, the tool orders the ports according to the declaration order in the design.

  Figure 13-55 shows a pin order file format example.

Figure 13-55. Example pin_order_file

```
// PortName    PinName     OptionList    LogicalGroups
// -------     ---------   ----------    ------------
    IN1           1           -            left_0
    IN2           2           -              -
    GND1[2]       3         GND              -
    GND1[1]       3A        GND            left_1
    GND1[0]       X1        GND              -
    IN3           4           -              -
    IN4           5           -            top
    OUT1          6           -              -
    OUT2          7           -            right_0
    VDD1          Y1        PWR              -
    TN            9         CE1            bottom
    SCANIN        10        NJTAG            -
    NC1           -         NC             right_0
    TCKP          11        TCK            right_0
    TMSP          12        TMS            right_0
    TRSTP         13        TRST           right_0
    TDIP          15        TDI            right_0
    TDOP          16        TDO            right_0
```

It consists of a four-column table where the boundary scan ordering is taken from top to bottom.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The first column lists the port name as it exists in the chip netlist.
- The second column lists the pin name the port maps to in its package. This information is only forwarded to the BSDL file and only needs to be accurate by the time the BSDL is used in the board environment. If you specify a "-" or nothing in this column, an integer is automatically assigned to this port in the BSDL file. As chips today often have multiple bonding configurations and packages, this is normally sufficient. The pin name typically needs to be modified at some time after the design has been completed. You can specify nothing in the second column if you do not have anything to specify in the third or fourth columns. You must use "-" if you want to let the tool pick an integer for the pin name, but you have entries to specify in the third or fourth columns.
- The third column is used to attach options to specific ports. It is equivalent to using the port_name_pattern property inside the BoundaryScanCellOptions wrapper. When you specify more than one option for a given port, the options must be separated with a comma. If you have no option, you can specify "-". You can also specify nothing in the third column as long as you do not have an entry in the fourth column (LogicalGroups).
- The fourth column is used to group the boundary scan cells into logical modules. It is equivalent to using the LogicalGroups wrapper inside the BoundaryScan wrapper. A "-" or no value in the fourth column means that the given port belongs to the logical grouping of the port above it.

You can supply a pin_order_file that specifies the port list ordering and nothing else in the second, third, and fourth columns. The tool automatically assigns the pin names. The automatically chosen names contain one or two letters for the direction and an arbitrary number. You can modify these later when you know the final pin names on the package.

Tip

You can specify the options and logical grouping more easily in the BoundaryScan wrapper because wildcards are supported.

- interface_parent_instance : *parent_instance* ;

  An optional property that identifies the design module in which the tool places the boundary scan interface. The default placement is at the top level if this property is not specified.

- create_ac_control : auto | on ;

  An optional property that instructs the tool to create an interface block to generate the ac_mode_en, ac_signal, ac_init_clock1, and ac_init_clock0 signals. When the value of this property is set to on, the tool creates a TAP controller and a dot6 control block. Both are instantiated in the dft_block instance, and the dot6 control block is connected to the TAP controller.

  The default is "auto."

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- outputs_per_enable_cell : *int* ;

  An optional property that instructs the tool to limit the number of output pins controlled by a given output enable cell. If the enable pin of the output pad cell is tied on or off, the tool is free to associate any output pad cell to any given enable cell in order to meet the outputs_per_enable_cell constraints. If, on the other hand, a group of output pad cells have their enable pins controlled by a single functional net, the tool inserts the appropriate number of enable boundary-scan cells, and it automatically separates the fanout of the enable signal such that the fanout of each enable cell is smaller than or equal to the specified outputs_per_enable_cell value. The splitting of the functional enable fanout is illustrated in the example of the LogicalGroups wrapper in Figure 13-42 and Figure 13-43.

- max_segment_length_for_logictest : *int* | unlimited | off ;

  An optional property that, when set to "unlimited" or an integer value, instructs the tool to segment the boundary scan chains into smaller segments to be reused by scan chain testing, EDT, or LogicBIST. If "unlimited" is specified, the boundary scan chain is accessible as one long scan chain.

  Note

  To use the boundary scan cells to isolate the core from the primary input ports, add the int_ltest_en DFT signal using the add_dft_signals command. The signal is connected to the boundary scan interface module and forces the to_select_jtag_input port to 1 when asserted.

  If you have inout ports, or if you want to save power during the LogicBIST modes, add the output_pad_disable DFT signal to allow turning off the tri-state output buffers. The signal is connected to the boundary scan interface module and forces the to_force_disable port to one when asserted. The output pads used for channel outputs are turned on during shift mode and tri-stated while in capture mode. This avoids bus conflict during ATPG and avoid needlessly using power in the output pad buffers. The pad buffers is tested by the boundary scan test patterns.

  If an integer is specified, the boundary scan chain is partitioned into segments of the specified size. Note that segments can end up having a different length because Tessent Shell is not able split the boundary scan chain at some positions, such as within a given EmbeddedBoundaryScan segment. You should choose a number that does not exceed the length of the other scan chains in the design because the boundary scan segments otherwise increase your test time.

  The default value of "off" indicates that the boundary scan chain is not reused during logic test.

- tristate_enable_non_contacted_test_support : on | off ;

  An optional property that, when set to "on," generates the hardware needed to test whether the pad driver of a bidirectional non-contacted pad can be disabled by the global force_disable signal and the local output enable signal. Pattern generation is controlled by specifying the tristate_enable_non_contacted test for RunTest.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The generated hardware is controlled by the two global DFT control signals tristate_enable_nc_test and tristate_enable_nc_test_type, which can be shown with the report_dft_signal_names command. It is recommended to specify these control signals with add_dft_signals before creating the DFT specification so the IJTAG network has TDRs to control the generated hardware.

The default is "off."

Note

This feature is not available in the embedded boundary scan implementation.

- tck_period : *period* ;

  An optional property that instructs the tool to use a different default period of the test clock. The default value of the Tessent BoundaryScan period is 100ns.

- default_cell_on_bidir_port : combined | separated;

  An optional property that instructs the tool to use either a single boundary scan register (combined) or two independent unidirectional boundary scan registers (separated) for bidirectional ports.

- handle_logictest_segments_during_scan_insertion : on | off | auto ;

  An optional property to be used with the max_segment_length_for_logictest property. The handle_logictest_segments_during _scan_insertion property defines the type of logic test segments to be created. When this property is set to "on", the logic test segments are described per logical group with a tcd_scan and a ctl file and can be used during scan insertion. When this property is set to "off", the created logic test segments are described in the instrument directory and are used during EDT DFT insertion. The default is "auto", which resolves to "on" when there are lower-level embedded BoundaryScan segments with logic test support and "off" otherwise.

  At the sub-block or physical block level, describe the segments for scan insertion to obtain logic test support. If you want to use the BoundaryScan chains as wrapper chains inside the block, we recommend that you set the max_segment_length_for_logictest property to a number or to "unlimited" at the block level, and to "on" later at the chip level.

  When this property is set to int or unlimited, a *.ctl* file is created for each logical group module. This file describes the scan chains created from embedded Boundary Scan segments and may be used during scan chain stitching in third-party tools.

  Note

  The Core/BoundaryScan/logic_test_segments_present property specifies whether an embedded BoundaryScan chain has logic test support.

Examples

The following example defines a BoundaryScan wrapper. The ijtag_host_interface property is used to specify that the boundary scan chain is to be connected to a TAP controller defined in the IjtagNetwork wrapper. The file *./mychip.pin_order* is referenced using the pin_order_file

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

property to order the boundary scan cell so as to minimize routing. See the set_boundary_scan_port_options -pin_order_file option to have the create_dft_specification command create a DftSpecfication that points to an existing pin order file. See the read_def command to see how to load a DEF file and have the pin order file automatically created following the placement of the ports in your design.

In this example, the pin order file only contains the column that lists the port names. The PinName column is omitted and the tool auto-assigns integers to the pin names in the BSDL file. The Option column is also not used and, instead, the Options are specified using the BoundaryScanCellOptions wrapper. The outputs_per_enable_cell property is specified with a value of 8, which means that one enable boundary scan cell is inserted for every eight ports. A group of 22 pads in this design share a common functional enable signal. The tool inserts three enable cells for this group, each controlling the enable pin of 7 or 8 output pad buffers. The max_segment_length_for_logictest property is used to specify that the tool is to provide access to the boundary scan chain for logictest in chain segments no longer than 200 flip-flops long.

```
DftSpecification(mychip,rtl) {
  BoundaryScan {
    ijtag_host_interface             : Tap(main) ;
    pin_order_file                   : ./mychip.pin_order ;
    outputs_per_enable_cell          : 8 ;
    max_segment_length_for_logictest : 200 ;
  }
}
```

Because the above example used the “max_segment_length_for_logictest” property, the instrument dictionary “mentor::jtag_bscan::DftSpecification” contains the “logic_test_scan_chains” section which can be accessed this way:

```
set logic_test_enable [get_instr_dict \
         mentor::jtag_bscan::DftSpecification logic_test_scan_chains]
foreach chain [dict keys $logic_test_enable chain*] {
  dict with logic_test_enable $chain {
    puts "length = $length"
    puts "scan_in = $scan_in"
    puts "scan_out = $scan_out"
    puts "scan_in_capture_edge = $scan_in_capture_edge"
    puts "scan_out_launch_edge = $scan_out_launch_edge"
  }
}
```

The dictionary can also be exported to a file and re-used in a separate tool invocation when using the -variable_name switch and the redirection option “>”. The generated file can then later be sourced into a dofile to recover a variable called “bscan_ltest_chains,” which holds the value of the dictionary.

```
format_dictionary [
  get_instrument_dictionary mentor::jtag_bscan::DftSpecification \
                            logic_test_scan_chains \
] -variable_name bscan_ltest_chains > bscan_ltest_chains.dict
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ImplementationOptions

Specifies implementation options when building the boundary-scan cells.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan | EmbeddedBoundaryScan {
    ImplementationOptions {
      clocking           : tck | gated_tck | gated_tck_inv ; // *DefSpec
      update_stage       : flop | latch | auto ;             // *DefSpec
      scan_path_retiming : flop | latch ;                    // *DefSpec
    }
  }
}
```

Description

A wrapper that specifies implementation options when building the boundary-scan cells. The options consist of the clocking, and the use of latches or flip-flops as an update stage and for scan path retiming.

Arguments

- clocking: tck | gated_tck | gated_tck_inv ;

  A property that specifies how the clocking of the boundary-scan cell should be constructed. When specifying tck, the boundary-scan cells are clocked with the free-running TCK clock and the capture and shift is enabled using the capture_en and shift_en functions. This option requires one extra multiplexer per boundary-scan cell as compared to the gated_tck or gated_tck_inv implementations. When specifying gated_tck, a gated low version of TCK is used and the gating lets the TCK go through during the capture and shift enable pulse. When specifying gated_tck_inv, a gated high version of TCK is used and the gating lets the inverted TCK go through during the capture and shift enable pulse. All three clocking specifications behaves identically in IEEE 1149.1 mode. Using TCK versus the other two gated TCK options is a trade-off between area and the number and sizes of clock trees. Note that in all three cases, the boundary-scan cells are always built such that the scanin of each cell is strobed on the rising edge of TCK and the scanout is launched off the negedge of TCK to make sure boundary-scan cells always work without having to balance TCK to all boundary scan cells which makes the ground bounce issue worst.

  The gated_tck_inv option is a legacy clocking mode and is the one that is compatible with the existing LogicVision burst mode architecture. For more information, see the "BurstMode Logic BIST Architecture" section in the *LV Flow User's Manual*.

  Table 13-4 illustrates the boundary scan cell implementations based on the clocking, update_stage and scan_path_retiming values.

- update_stage: flop | latch | auto ;

  A property that specifies if the update stage is to be implemented using a latch or a flop. Table 13-4 illustrates the boundary scan cell implementations based on the clocking, update_stage, and scan_path_retiming values.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When this property is set to auto, a flop is used if the value of the max_segment_length_for_logictest property is not “off” or if the value of the clocking property is “tck”. Using a latch causes false E5 DRC violations and loss of coverage if X-bounding is used. (If X-bounding is not used, fault coverage is not affected.)

- scan_path_retiming : flop | latch ;

  A property that specifies if the scan path retiming element is to be implemented using a latch or a flop.

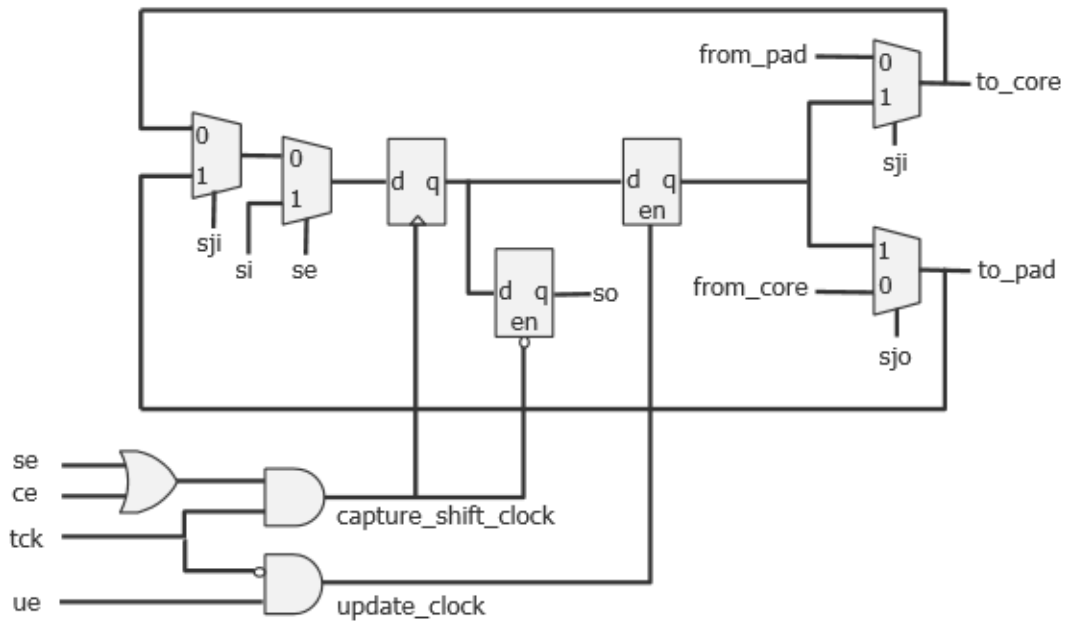
  Table 13-4 illustrates the boundary scan cell implementations based on the clocking, update_stage, and scan_path_retiming values. Refer to Table 13-6 to see the implementation of an input and sample_only boundary scan cell using the default implementation options. The following function abbreviations are used in the table figures:

  - se — shift enable
  - ce — capture enable
  - ue — update enable
  - si — scan input
  - so — scan output
  - sji — select jtag input
  - sjo — select jtag output
  - tck — test clock

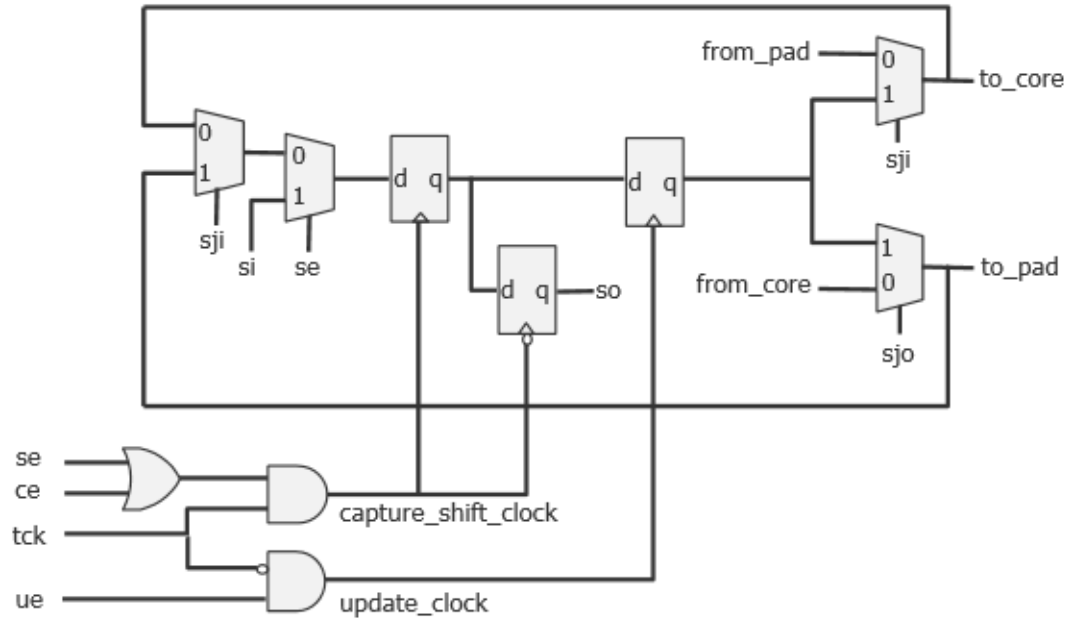
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-8. Illustration of Different Boundary Scan Implementations

1) This example shows how the boundary-scan cells are constructed when clocking is set to gated_tck, and update_stage and scan_path_retiming are both set to latch.



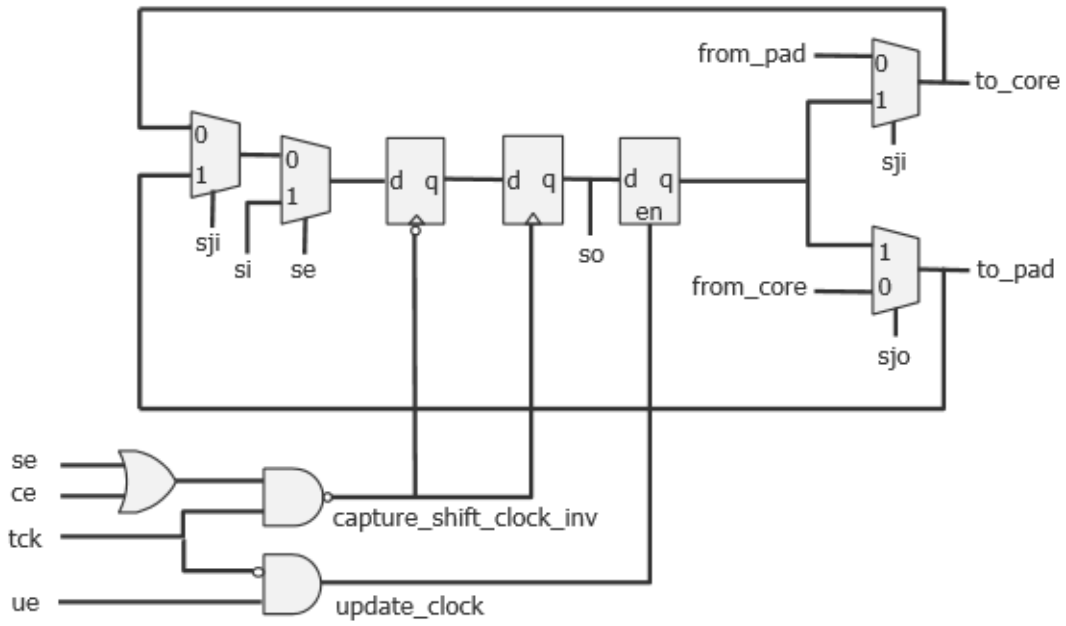
2) This example shows how the boundary-scan cells are constructed when the clocking is set to gated_tck and update_stage and scan_path_retiming are both set to flop. Compared to the prior example, this shows how changing update_stage and scan_path_retiming from latch to flop simply converts the latch into a flop.



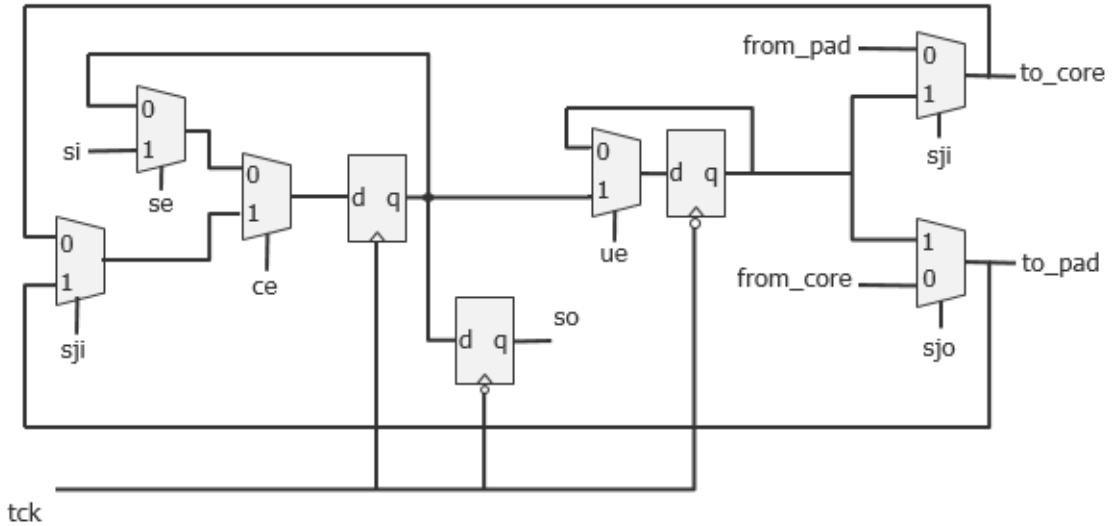
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-8. Illustration of Different Boundary Scan Implementations (cont.)

3) This example shows how the boundary-scan cells are constructed when clocking is set to gated_tck_inv, and update_stage and scan_path_retiming are both set to latch. This is a legacy mode that makes the boundary-scan cell compatible with the LogicVision burst mode architecture. Note how the implementation for gated_tck and gated_tck_inv are almost the same; only the active-low retiming flop is placed before the data flop.



4) This example shows how the boundary-scan cells are constructed when clocking is set to tck, and both update_stage and scan_path_retiming are set to flop. Notice the two extra multiplexers that are needed in each cell.



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example sets the clocking property to gated_tck_inv and the update_stage to latch. The scan_path_retiming is unspecified so it defaults to latch.

```
DftSpecification(mychip,rtl) {
  BoundaryScan {
    ImplementationOptions {
      clocking     : gate_tck_inv ;
      update_stage : latch ;
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BoundaryScanCellOptions

Assigns options to boundary scan ports.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan | EmbeddedBoundaryScan {
    BoundaryScanCellOptions {
      port_name_pattern : option, ... ; // repeatable
    }
  }
}
```

Description

This wrapper is used to assign options to boundary scan ports. This wrapper replaces the Overrides wrapper in the *.etassemble* file.

Arguments

- *port_pattern_name* : *option*, ... ;

  A syntax used to assign one or more options to a set of ports matched by the specified *port_pattern_name*. The *port_pattern_name* can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and, when specified, is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches "ABC00", "ABC01", and "ABC02".

  Table 13-5 lists the option names and their meanings. The option can be specified in the BoundaryScanCellOptions wrapper or in the third column of the *pin_order_file*.

Table 13-9. BoundaryScanCellOptions list

Option name	Meaning
add_dot6_from_pad_cell	Used to identify AC input and inout ports where the from_pad data signal is intercepted with an additional boundary scan cell. The boundary scan cells sampling the AC pad Test Receivers is added as normal.
analog	Used to identify input, output, and inout ports that are analog. These pins do not have a boundary scan cell and are reported as linkage pins in the BSDL file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-9. BoundaryScanCellOptions list (cont.)

Option name	Meaning
compliance_enable0	Used to identify input and inout ports that are active low compliance enable pins. These ports are reflected as such in the BSDL file to reflect the fact that these pins must be driven low for the chip to behave in accordance with the BSDL description. The compliance enable zero pins are automatically asserted low in any pattern that makes use of the TAP controller. This option is automatically set when you use the -enable_low_compliance_enables switch of the create_dft_specification command. If the compliance enable logic is already present in your design, you cannot use the -enable_low_compliance_enables switch of the create_dft_specification command. Instead, you follow the steps illustrated in Example 2 of the IjtagNetwork wrapper section.
compliance_enable1	Used to identify input and inout ports that are active high compliance enable pins. Those ports are reflected as such in the BSDL file to reflect the fact that these pins must be driven high for the chip to behave in accordance with the BSDL description. The compliance enable one ports are automatically asserted high in any pattern that makes use of the TAP controller. This option is automatically set when you use the -enable_high_compliance_enables switch of the create_dft_specification command. If the compliance enable logic is already present in your design, you cannot use the -enable_high_compliance_enables switch of the create_dft_specification command. Instead, you follow the steps illustrated in Example 2 of the IjtagNetwork wrapper section.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-9. BoundaryScanCellOptions list (cont.)

Option name	Meaning
clock	Identifies input and inout ports that are clocks. Specifying this value alters the RTL and the BSDL definition of the generated boundary-scan cells associated with the port. • If the port is of type input, the boundary-scan cell has a BSDL function of "clock". No JTAG multiplexer is inserted along the functional input path, just as if the sample_only option was specified. • If the port is of type inout, the boundary-scan cell has a BSDL function of "bidir" to cover both the input and output directions. There is no JTAG multiplexer inserted along the functional input path, just as if the sample_only option was specified. The boundary-scan cell is modified to capture itself instead of the from_core signal in the output direction, just as if the no_capture_core_signal option was specified. If the output direction is not used, you can use the "input_only" option in addition to the "clock" option. In this case, the BSDL function is "clock".
combined_bidir_cells	For bidirectional ports, specifies that the tool add a single boundary scan TS_BC_7 data cell to handle both input and output signal flow. This is the default. If you need an alternative method that specifies two separate boundary scan cells for a port, use the separated_bidir_cells option instead.
dont_touch	Used to identify input, output, and inout ports that are to be fully excluded from boundary scan. It is similar to using the no_bscan_cell option except that for output and inout ports, the output enable pin of the output pad buffer is not intercepted with the force_disable signal such that the port cannot go tri-state during the HIGHZ instruction. These ports are reported as linkage pins in the BSDL file.
ground	Used to identify input and inout ports that are ground pins in the package part. These ports are reported as linkage in the BSDL file. The ports associated with the ground option may or may not exist in the design. If they do not exist in the design file, the port name pattern cannot include the asterisk wildcard symbol "*" but you can use the %d[range] syntax to define a group of numbered ports as ground ports. The ports on which you have set the function attribute to ground prior to running the check_design_rules command are automatically identified as ground pins in the created DftSpecification wrapper when you run the create_dft_specification command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-9. BoundaryScanCellOptions list (cont.)

Option name	Meaning
input_only	Used to identify inout ports that are to be treated as input-only ports during boundary-scan insertion, even if the actual port direction is inout in the netlist and the port is connected to a bidirectional pad cell. The generated BSDL file documents the port as input, and the port is only equipped with an input-type boundary-scan cell. **Note:** The input_only option works in conjunction with the clock or sample_only option if you enclose both options in braces. For example: "{input_only sample_only}".
no_bscan_cell	Used to identify input, output, and inout ports that are not to be equipped with a boundary-scan cell. It is similar to using the dont_touch option except that for output and inout ports, the output enable pin of the output pad buffer is still intercepted with the force_disable signal such that the port still goes tri-state during the HIGHZ instruction. These ports are reported as linkage pins in the BSDL file.
no_connect	Used to identify input, output, and inout ports that are not connected to package pins. Those ports are treated as dont_touch pins and they are not reported at all in the BSDL file.
no_capture_core_signal	Used to identify output and inout ports that are to be equipped with boundary-scan cells that do not capture the functional core signal but instead hold their values during the capture cycle of the INTEST instruction. This option is useful when you are reusing the boundary scan chain as isolation during logictest and you want to avoid the boundary-scan cells from capturing unknowns when the from_core signal is not controllable during logictest. Avoiding unknowns is a requirement if you are using LogicBIST.
output_only	Used to identify inout ports that are to be treated as output-only ports during boundary-scan insertion even if the actual port direction is inout in the netlist and the port is connected to a bidirectional pad cell. The generated BSDL file documents the port as output, and the port is only equipped with an output-type boundary-scan cell.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-9. BoundaryScanCellOptions list (cont.)

Option name	Meaning
power	Used to identify input and inout ports that are power pins in the package part. These ports are reported as linkage in the BSDL file. The ports associated with the power option may or may not exist in the design. If they do not exist in the design file, the port name pattern cannot include the asterisk wildcard symbol "*" but you can use the %d[range] syntax to define a group of numbered ports as ground ports. The ports on which you have set the function attribute to power prior to running the check_design_rules command are automatically identified as power pins in the created DftSpecification wrapper when you run the create_dft_specification command.
sample_only	Identifies input and inout ports that are not to have a JTAG multiplexer inserted along the functional input path. The fanout of these ports cannot be intercepted by the boundary-scan cell value during the INTEST instruction. Also, if you reuse the boundary scan as isolation during logictest, these ports are not isolated. This option is inferred for input or inout ports having the clock option. You typically also want to use it for Async reset input ports.
separated_bidir_cells	For bidirectional ports, specifies that the tool add two boundary scan BC_2 data cells, one to handle input signal flow and the other to handle output. The default is combined_bidir_cells, which specifies a single boundary scan cell for a port.

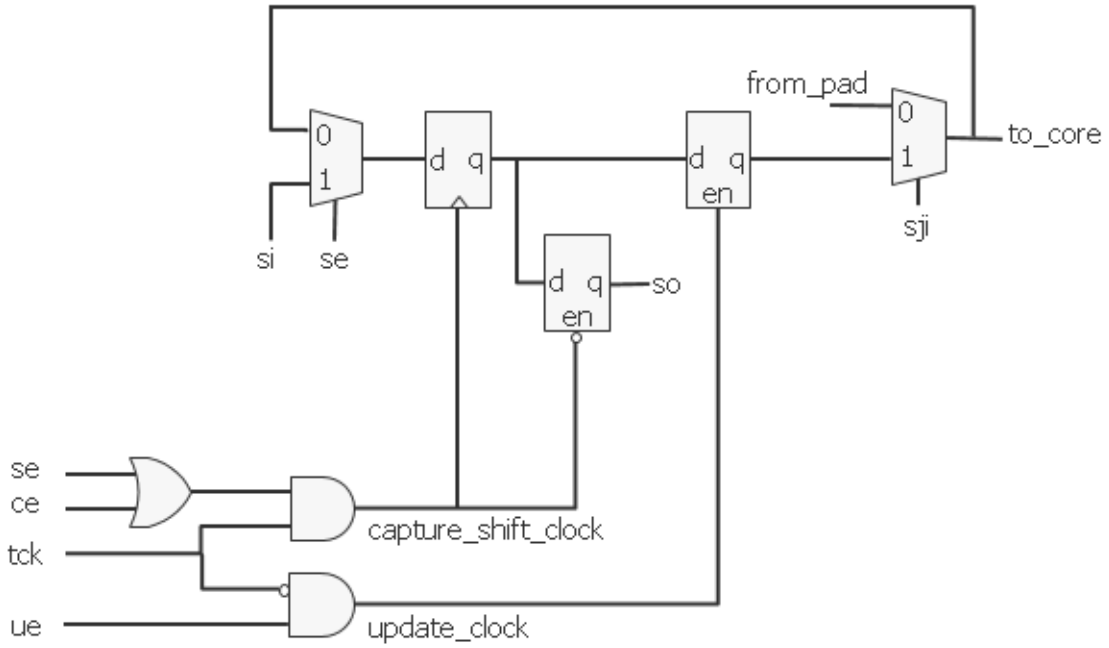
The differences between a normal input boundary scan cell and one assigned the sample_only option are illustrated in Table 13-6 below. Refer to Table 13-4 for information on how an inout cell is constructed. The following function abbreviations are used in the table figures:

- se — shift enable
- ce — capture enable
- ue — update enable
- si — scan input
- so — scan output
- sji — select jtag input
- sjo — select jtag output
- tck — test clock

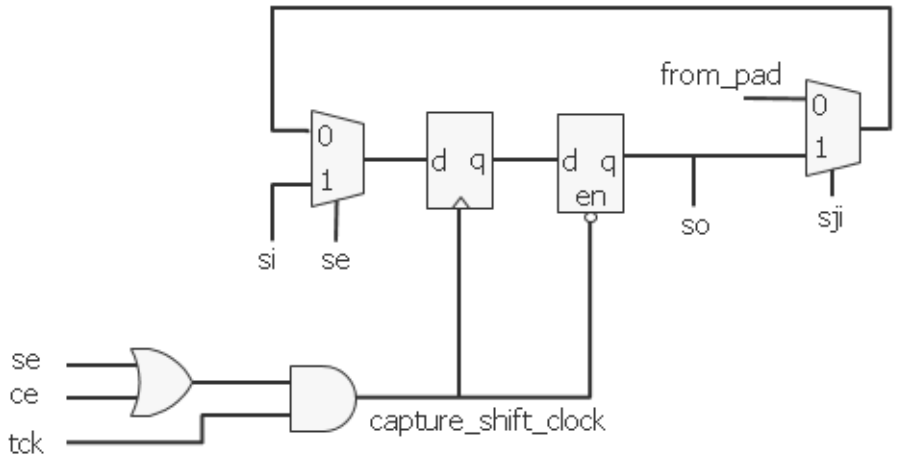
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-10. Illustration of the “sample_only” Boundary Scan Cell Option

1) This example shows a normal input boundary scan cell. The multiplexer shown on the right intercepts the from_pad connection of the pad cell. The boundary scan cell can drive a value to the core in INTEST mode.



2) The example below shows a sample_only input boundary scan cell. The multiplexer shown on the right no longer intercepts the from_pad connection of the pad cell, but simply connects to it, preserving the direct connection between the pad and core. Note that compared to the normal input boundary scan cell, the update latch has been removed and the retiming latch is now connected to the multiplexer on the right.



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example uses the BoundaryScanCellOptions wrapper to identify VDD and VSS ports in the BSDL file. These ports do not exist in the design. The pattern clk* matches any ports in the design starting with the string “clk” as clock ports. They do not have a JTAG mux inserted along their functional input path, and they are identified as Clock ports in the BSDL file.

```
DftSpecification(mychip,rtl) {
  BoundaryScan {
    BoundaryScanCellOptions {
      VDD%d[0:32] : power ;
      VSS%d[0:56] : ground ;
      clk* : clock;
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

LogicalGroups

Specifies how to group the boundary-scan cells into logical grouping modules.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan | EmbeddedBoundaryScan {
    LogicalGroups {
      LogicalGroup(group_name) {
        first_port         : port_name ;
        parent_instance    : parent_instance ;
        leaf_instance_name : leaf_instance_name ;
      }
    }
  }
}
```

Description

A wrapper that specifies how to group the boundary-scan cells into logical grouping modules. You can specify more than one LogicalGroup wrapper only when you have specified the pin_order_file property in the BoundaryScan wrapper.

You use the first_port property to identify the first port in the pin order file that belongs to the logical grouping. The association between ports and the LogicalGroup can also be specified in the forth column of the pin order file. If you do not specify any LogicalGroup(*group_name*) wrapper, all boundary scan cells are grouped into a single module, the module is instantiated into the top module, and the leaf instance name is as follows:

```
design_name_design_id_tessent_bscan_logical_group_DEF_inst
```

as if you had specified an empty LogicalGroup(DEF) wrapper.

You use the parent_instance property to specify where you want the logical grouping module to be instantiated. By default, it is instantiated in the top module. You use the leaf_instance_name property to specify the leaf instance name used when instantiating the module if you are not satisfied with the default value. The default leaf_instance_name used is *design_name_design_id*_tessent_bscan_logical_group_*group_name*_inst.

The example below shows how to specify the parent_instance property in order to insert each logical grouping module in the same instance where the pins or nets to be intercepted by the boundary-scan cells are located.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- *group_name*

  A string that uniquely identifies each logical group. The string starts with a letter and can be followed by any number of letters, numbers, and underscores. The group_name is used in constructing the grouping module name as follows:

  `design_name_design_id_tessent_bscan_logical_group_group_name`

- first_port : *port_name* ;

  A property that specifies a port name that exists in the specified pin_order_file. This property is optional when there is only one LogicalGroup wrapper but it is required when there are more then one. Specifying a port_name for the first_port property is equivalent to specifying the group_name in the forth column of that port in the pin_order_file. See the pin_order_file property description in the BoundaryScan wrapper for a description of that file format and usage.

- parent_instance : *parent_instance* ;

  A property that controls where the logical grouping module is instantiated. By default, it is instantiated in the root module but you can push it down into instances. See the following example to understand the requirement that must be met when choosing the parent_instance of the logical grouping module.

  If the specified parent_instance is not unique, it is uniquified before the logical grouping module is inserted in it. You can do bottom up insertion using the EmbeddedBoundaryScan wrapper on sub modules to insert segments of boundary scan chains into repeated modules, and then reuse them as building blocks at the next level up if you want to avoid uniquifying the parent_instance. Specify the "set_design_level sub_block" command and option when inserting boundary scan into those sub modules.

- leaf_instance_name : *leaf_instance_name* ;

  A property that specifies a different leaf name when instantiating the logical grouping module into its parent instance. The default value when unspecified is *design_name_design_id*_tessent_bscan_logical_group_*group_name*_inst.

  If you choose to change it, you do not have to worry about picking a unique leaf name in the given parent_instance. The tool automatically appends the instance_uniquification_suffix to your specified leaf instance name in case of conflicts. See the set_insertion_options command description for a complete description of the -instance_unification_suffix option.

Examples

The following example illustrates the use of the parent_instance property of the LogicalGroup wrapper to meet the rule that the logical grouping module must be instantiated where all the intercepted nets are visible. Figure 13-39 shows the circuit before the insertion of the boundary-scan cells. You can see that the functional connections to and from the pad cells associated with ports P3 and P4 do not exist outside the B1_I instance. In this example, a single logical grouping module is specified so it can only be instantiated inside B1_I as illustrated in Figure 13-40.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
BoundaryScan {
  LogicalGroups {
    LogicalGroup(g1) {
      parent_instance : B1_I ;
    }
  }
}
```

The same example is modified to use a pin_order file thus enabling the splitting of the boundary-scan cells into more than one logical grouping module. The pin order file defines the order from P1 to P4. Two logical grouping modules are specified in which the first one, g1, starts with P1 and goes until P2 and the second one, g2, starts with P3 and goes until P4. The g2 group is still restricted to having its parent_instance property as B1_I because this is the only place where all of the connections to be intercepted are visible. The g1 group, on the other hand, can be instantiated in four locations because all of the connections to be intercepted are visible in those four locations. The parent_instance is shown for the four options, and Figure 13-41 shows the corresponding locations of the logical grouping modules.

```
BoundaryScan {
  pin_order_file      : mychip.pin_order
  LogicalGroups {
    LogicalGroup(g1) {
      first_port      : P1 ;
      parent_instance : B1_I ;       // Option1
      parent_instance : . ;          // Option2
      parent_instance : B2_I ;       // Option3
      parent_instance : B2_I/B3_I ;  // Option4
    }
    LogicalGroup(g2) {
      first_port      : P3 ;
      parent_instance : B1_I ;
    }
  }
}
```

The example also illustrates that you can specify the outputs_per_enable_cell property with a value smaller that the actual fanout count of the functional enable signal. In this case, the functional enable signal fans outs to two output pads. One enable boundary-scan cell is inserted per output port, requiring the tool to split the output enable connectivity to each pad.

Figure 13-42 shows this example with option1 for the parent_instance property. No new ports need to be created. The functional enable signal is observable by two enable bscan cells, and one connection per output pad is made between the logical grouping module and the enable pin of the output pad cells.

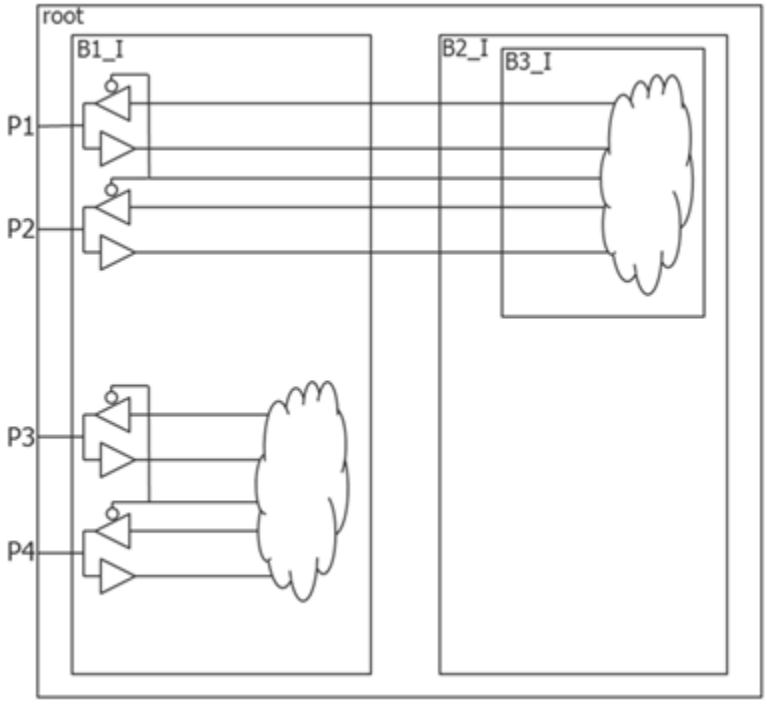
Figure 13-43 shows this example with option3 for the parent_instance property. In this case, the connections between the logical grouping module and the output pads require the creation of new pins as illustrated with the orange net crossing the hierarchy. If you are familiar with Tessent Boundary Scan within ETAssemble, the flexibility to precisely control the fanout of the enable signals was not present in that tool. All of those existing limitations are removed within Tessent Shell, and a net driven by RTL can now be intercepted and split for boundary scan

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

usage. In ETAssemble, you needed the functional enable signal to be identified on a pin which often meant you had to insert buffers in the RTL to enable the Boundary Scan to be placed in the same module as the pads.

```
BoundaryScan {
  pin_order_file      : mychip.pin_order
  OutputPerEnableCell : 1 ;
  LogicalGroups {
    LogicalGroup(g1) {
      first_port      : P1 ;
      parent_instance : B1_I ;       // Option1
      parent_instance : B2_I ;       // Option3
    }
    LogicalGroup(g2) {
      first_port      : P3 ;
      parent_instance : B1_I ;
    }
  }
}
```

Figure 13-56. Example Design With Pads



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-57. Single Logical Group Example

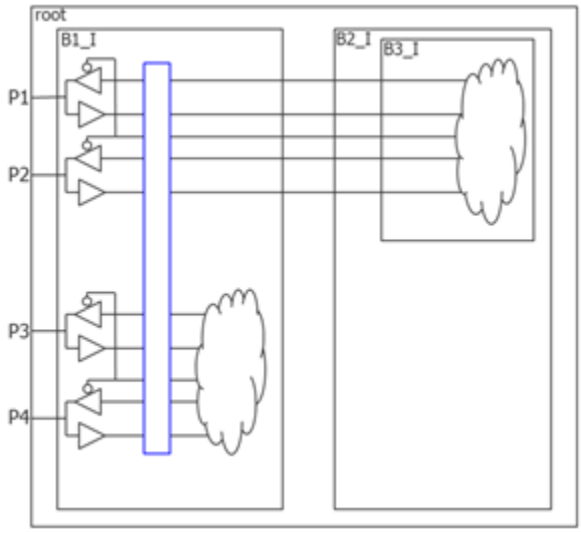
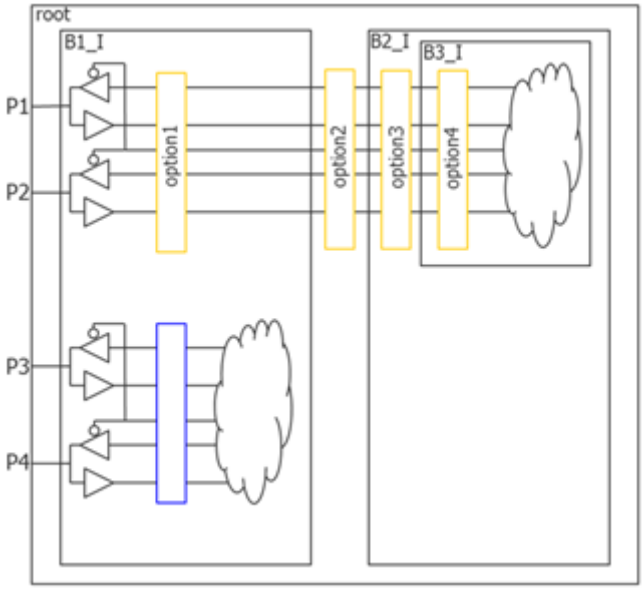


Figure 13-58. Two Logical Groups Example



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-59. Two Logical Groups Example1 with One Output per Enable Cells

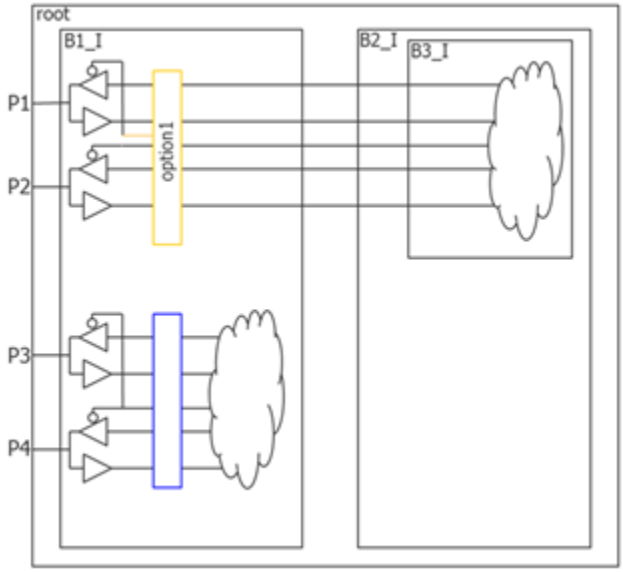
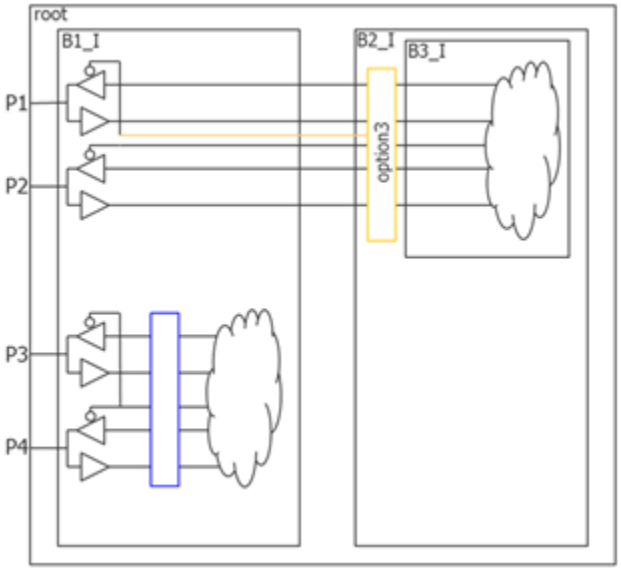


Figure 13-60. Two Logical Groups Example2 with One Output per Enable Cells



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ACModeOptions

Specifies options relevant to the IEEE 1149.6 AC mode of the boundary scan chain.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan {
    ACModeOptions {
      test_receiver_init_clock_value : initialization_event ;
                                       // Legal: logic_low logic_high
                                       // falling_edge rising_edge auto
      pulse_min_duration             : time | auto ;
      train_minimum_count            : count| unused ;
      train_max_duration             : time | unlimited ;
      ACGroup(group_name) {
        insert_before_port           : port_name ;
        insert_after_port            : port_name ;
        port_list                    : port_name, ... ;
      }
    }
  }
  EmbeddedBoundaryScan {
    ACModeOptions {
      test_receiver_init_clock_value : initialization_event ;
                                       // Legal: logic_low logic_high
                                       // falling_edge rising_edge auto
      ACGroup(group_name) {
        insert_before_port           : port_name ;
        insert_after_port            : port_name ;
        port_list                    : port_name, ... ;
      }
    }
  }
}
```

Description

A wrapper that specifies options relevant to the IEEE 1149.6 AC mode of the boundary scan chain. You use this wrapper to define AC enable cells and associate them with a group of ports. You also use this wrapper to specify the event type that initializes the test receiver when the value is not described in the cell library. Finally, you can use this wrapper to specify three properties that are simply forwarded into the BSDL file.

Arguments

- test_receiver_init_clock_value : initialization_event ;

  A property that is useful when the cell library description of the dot6 pad cell uses the legacy pad_init_clock_dot6 attribute instead of one of the new attributes that precisely describe the initialization event: pad_init_posedge_clock_dot6, pad_init_negedge_clock_dot6, pad_init_enable_high_dot6, and pad_init_enable_low_dot6. If the value is “auto” and the dot6 pad cell uses the legacy pad_init_clock_dot6 attribute instead of one of the four new attributes, the tool issues an error; you are required to specify the initialization event when it cannot be inferred from the cell library.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- pulse_min_duration : *time* | auto ;

  An optional property that defines the minimum time to use between the rising and the falling edge during the AC tests. This property is forwarded into the BSDL file using the AIO_EXTEST_Pulse_Execution attribute. When unspecified or specified to “auto”, the minimum pulse duration is chosen so as not to be smaller than three times the larger of the HP_time and LP_time BSDL values of each pad cell.

  ```
  attribute AIO_EXTEST_Pulse_Execution of <current_design>: entity is
        "Wait_Duration <pulse_min_duration_in_seconds>" ;
  ```

- train_minimum_count : *count* | unused ;

  An optional property that defines the minimum number of pulses the train test must have. This property is forwarded in the BSDL file using the AIO_EXTEST_Train_Execution attribute. The property defaults to “unused” when unspecified. Specifying a value of one effectively makes the train test equivalent to the pulse test.

  ```
  attribute AIO_EXTEST_Train_Execution of <current_design>: entity is
   "train <train_minimum_count>, maximum_time <train_max_duration>" ;
  ```

- train_max_duration : *time* | unlimited ;

  An optional property that defines a maximum time the train test must take. This property is forwarded in the BSDL file using the AIO_EXTEST_Train_Execution attribute. It defaults to “unlimited” when unspecified.

- *group_name*

  A string that uniquely identifies the ACGroup. This name is not reflected in the BSDL file but is used to name the leaf instance name of the AC group enable cell in the logical grouping module.

  If no AcGroup wrappers are specified, then all AC ports are active during the AC tests. If you specified at least one ACGroup wrapper but you have some AC ports not associated with an ACGroup, they are always active during the AC tests. In this situation, a warning is generated to let you know that not all AC pins were equipped with an AC enable cell. You can define an extra ACGroup wrapper without a port_list property to indicate that you want all AC ports not associated with another ACGroup wrapper to be associated with that one.

- insert_before_port : *port_name* ;

  A property that specifies the position of the AC enable boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the insert_before_port property is mutually exclusive with the specification of the insert_after_port property. If neither the insert_before_port nor insert_after_port properties are specified, the AC enable boundary-scan cell is inserted in front of the boundary-scan cell of the first port that is part of the group. The location of the AC group enable boundary-scan cell is reflected in the BSDL file using the AIO_Pin_Behavior attribute.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- insert_after_port : *port_name* ;

  A property that specifies the position of the AC enable boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the insert_before_port property is mutually exclusive with the specification of the insert_after_port property. If neither the insert_before_port nor insert_after_port properties are specified, the AC enable boundary-scan cell is inserted in front of the boundary-scan cell of the first port that is part of the group. The location of the AC group enable boundary-scan cell is reflected in the BSDL file using the AIO_Pin_Behavior attribute.

- port_list : *port_name_pattern*, ... ;

  A property used to list the port names that belong to the AC group. The port_name_pattern can include the asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and, when specified, is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

  You can have one ACGroup wrapper with an unspecified port_list. This ACGroup collects all AC ports that are not already associated with another ACGroup wrapper.

Examples

The following example defines an ACGroup wrapper with the name "pci" to specify that all ports starting with PCI_ belong to the group. All other AC ports are associated with the "other" group. The "other" ACGroup collects all other AC ports because it does not specify the port_list property.

```
DftSpecification(my_chip,rtl) {
  BoundaryScan {
    ACModeOptions {
      ACGroup(pci) {
        port_list : PCI_* ;
      }
      ACGroup(other) {
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

AuxiliaryInputOutputPorts

Specifies two lists of port name patterns to be used as auxiliary input ports, output ports, or both, during some test modes.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan  {
    AuxiliaryInputOutputPorts {
      auxiliary_input_ports  : port_name_pattern, ... ;
      auxiliary_output_ports : port_name_pattern, ... ;
    }
  }
  EmbeddedBoundaryScan{
    AuxiliaryInputOutputPorts {
      internal_auxiliary_input_ports  : port_name_pattern, ... ;
      internal_auxiliary_output_ports : port_name_pattern, ... ;
      external_auxiliary_input_ports  : port_name_pattern, ... ;
      external_auxiliary_output_ports : port_name_pattern, ... ;
    }
  }
}
```

Description

A wrapper that specifies two lists of port name patterns to be used as auxiliary input ports or output ports during some test modes. You can specify an inout port by adding it to both lists. In the output directions, a 2-to-1 multiplexer (shown in green in Figure 13-46) is added on the test side of the JTAG multiplexer (shown in red in Figure 13-46). The auxiliary output multiplexing is added in the boundary-scan cell to avoid cascading two multiplexers along the functional output path. For auxiliary input logic, an AND gate is used to prevent the aux_in_data signal from toggling in functional mode. The aux_in_en signal is also used to force the output pad driver off when adding AuxIn logic on an inout pad.

The aux_out_enable and aux_in_enable signals are kept separate for each port to provide full flexibility as to which output and input ports are enabled to carry input or output auxiliary data in any given test mode. The ports equipped with auxiliary data input and output logic are useful for supplying channel input and output data to EDT controllers. They are also useful to provide the source of the scan_en, test_clock, and edt_update DFT signals. See the description of the add_dft_signals command for more info. Notice that the aux_out_en and aux_in_en signals have higher priority than the select_jtag_input and select_jtag_output signals. The implication of this is that the aux_out_en and aux_in_en sources must reset off and remain off when using the IEEE 1149.1 test mode of the chip. This requirement is met if you use the DFT signals created by the add_dft_signals to control those pins.

Figure 13-44 shows the typical boundary scan cell inserted on a inout pad. The red multiplexers are those referred to as JTAG multiplexers previously. They are controlled by the select_jtag_output to let the output data and output enable values be controlled by the Boundary Scan cells in IEEE 1149.1 test modes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-61. Typical Boundary Scan Cell Inserted for an Inout Pad

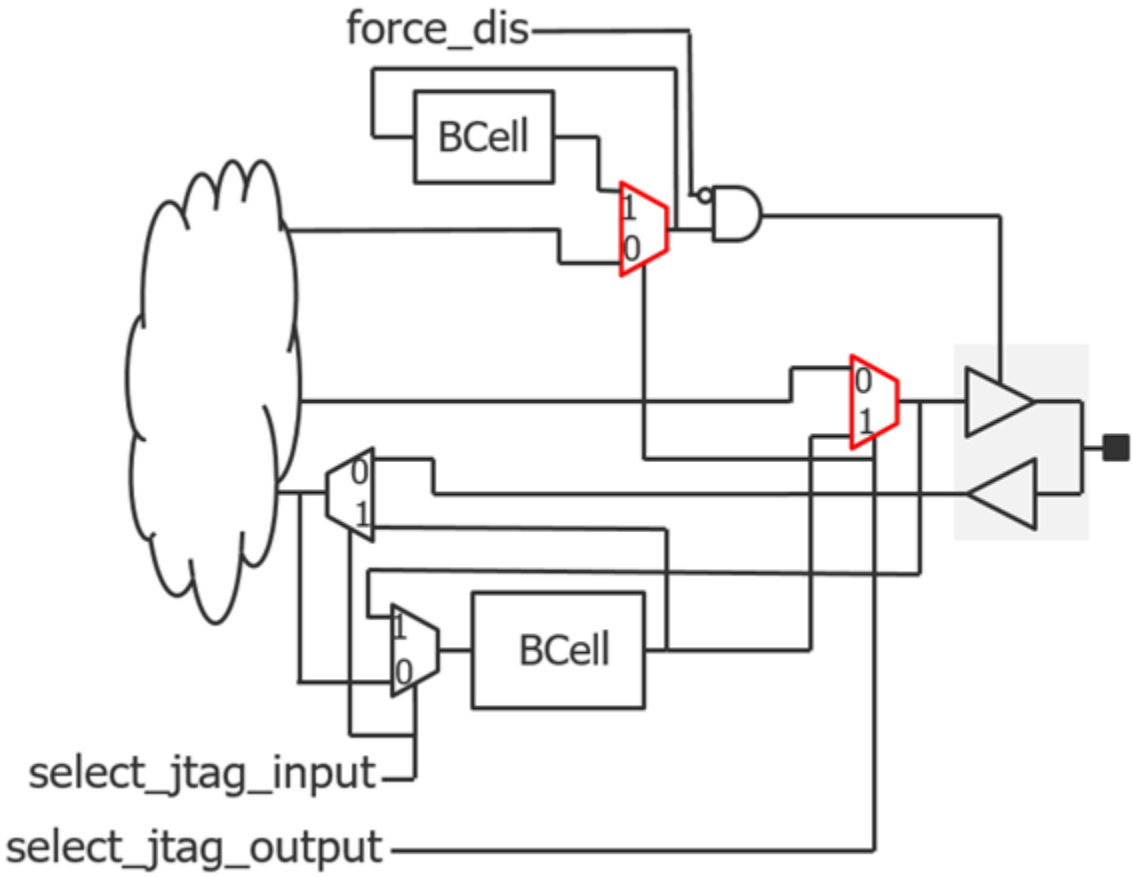


Figure 13-45 shows the boundary scan cell when it is used during ATPG. If you set the max_segment_length_for_logictest property found in the Boundary Scan wrapper to an integer or to unlimited, ATPG scan chains are created with the Boundary Scan chain. The blue path shows how the boundary scan cell is used to isolate the core values from the input ports by forcing select_jtag_input to 1, which is done by forcing the int_ltest_en pin on the boundary scan interface module. For more information about the boundary scan interface module and the logic inserted in it when the boundary scan chain is reused in logic test, see Figure 13-53 on page 4161.

Note

If you are using the DFT signal flow, add the int_ltest_en DFT signal using the add_dft_signals command when you reach the top level and it is connected to the boundary scan interface module. You can then activate the input isolation using the "set_static_dft_signal_values int_ltest_en 1" command and switch when you want to create your ATPG patterns.

The select_jtag_output signal is under ATPG control, enabling both the input and the output of the JTAG multiplexers to be tested. If you do not force the int_ltest_en pin to 1, then the select_jtag_input is also under ATPG control and the input path can also be tested.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-62. Boundary Scan Cell as Observation and Control in Logic Test Modes

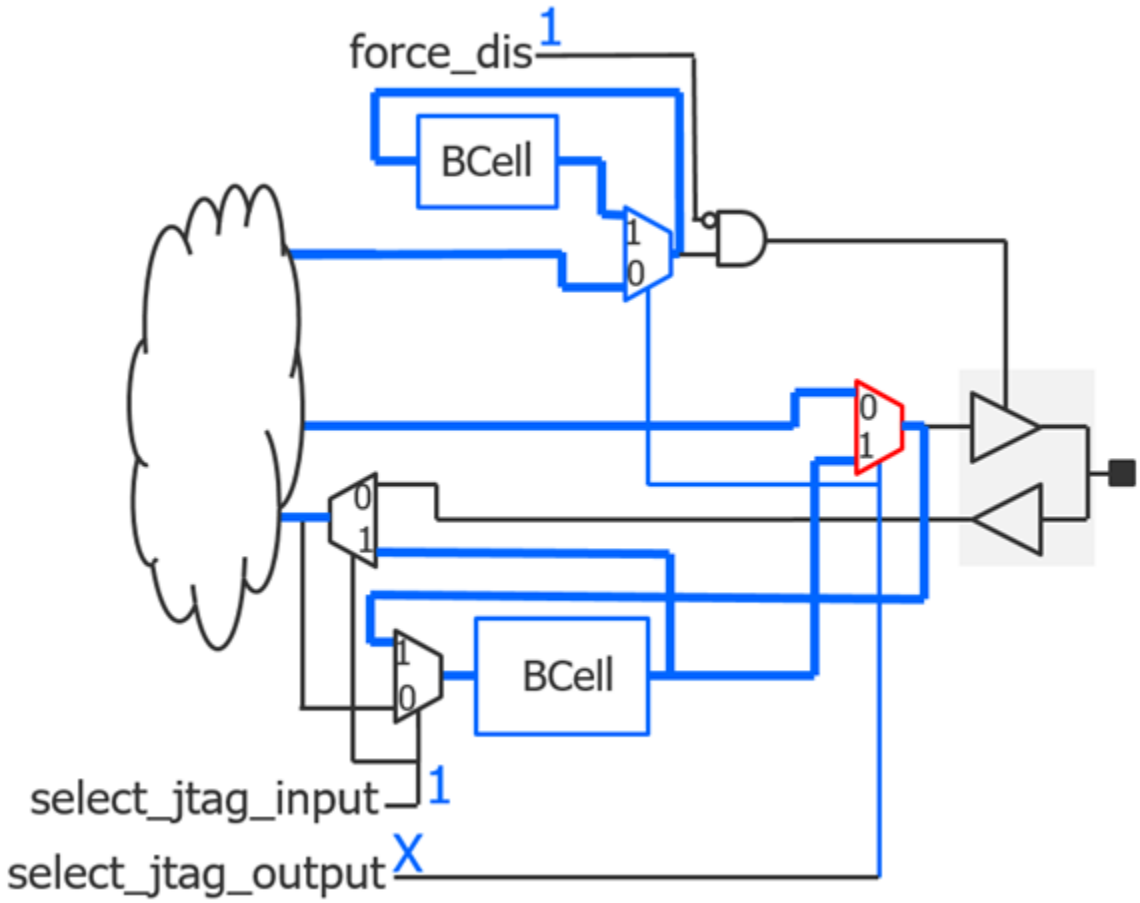


Figure 13-46 shows the same boundary scan cell shown in Figure 13-44 but with auxiliary logic (colored green) added for both the input and output directions. The green multiplexer is inserted on the test side of the JTAG multiplexer (shown in red) rather than in the functional path. The aux_in_en and aux_out_en signals are mutually exclusive, and the boundary scan cell is in functional mode when they are both low.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-63. Adding Auxiliary Input and Output Logic in Bscan Cell

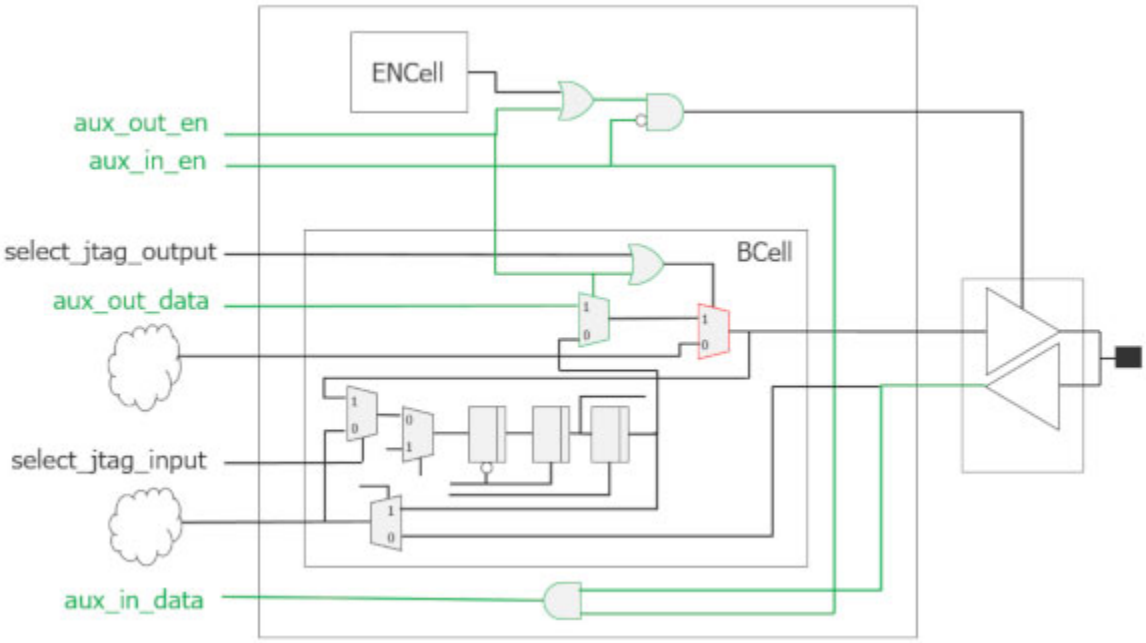


Figure 13-47 shows aux_out_en asserted to 1 to enable the auxiliary output data path. The two multiplexers are switched to select the aux_out_data value, and the enable of the output pad is forced on. When aux_out_en is high, the boundary scan cell is not able to observe the value coming from the core. This is why the add_dft_modal_connections command gates the aux_out_en signals with “scan_en” such that it is active only in shift mode. In capture mode, the aux_out_data is not needed, and instead the boundary scan cell is used to observe the functional core output. The capture mode is shown in Figure 13-48.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-64. Path Used When AuxOut Logic Activated

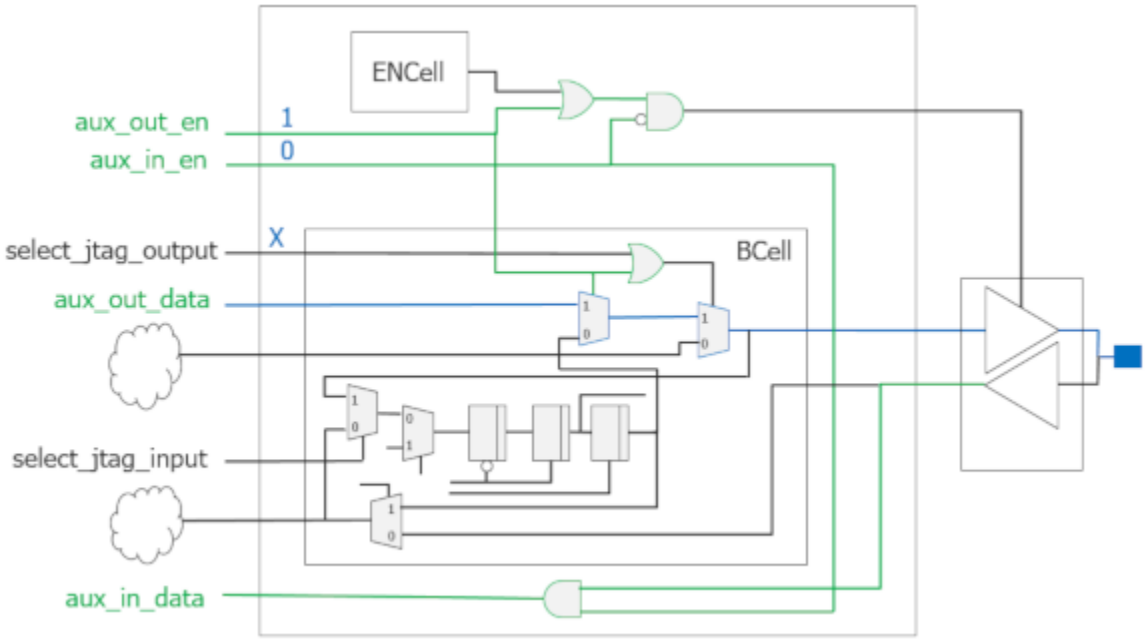


Figure 13-65. Forcing aux_out_en Low in Capture Mode to Observe Core Output

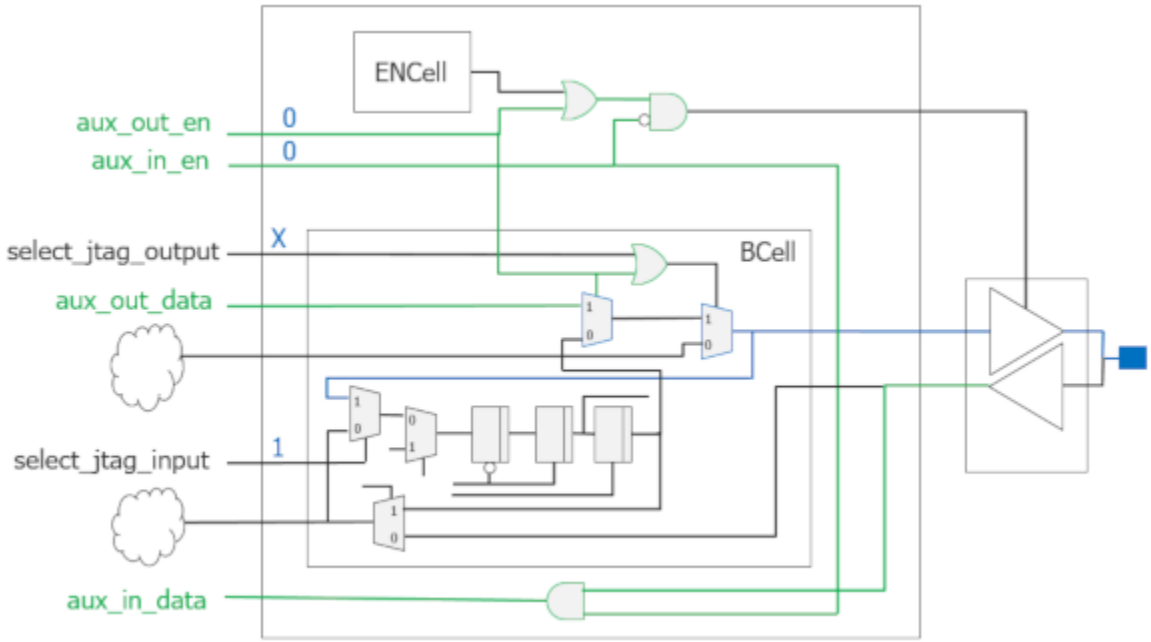


Figure 13-49 shows the I/O pad used as an auxiliary input during ATPG. The boundary scan cell is used to control the signal going to the core, while the input buffer provides the value to the auxiliary in data. This scheme is useful to reuse functional pins for the scan_en, test_clock, and edt_update DFT signals without losing ATPG fault coverage of those inputs.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-66. Using AuxIn and Controlling to Core Signal in ATPG

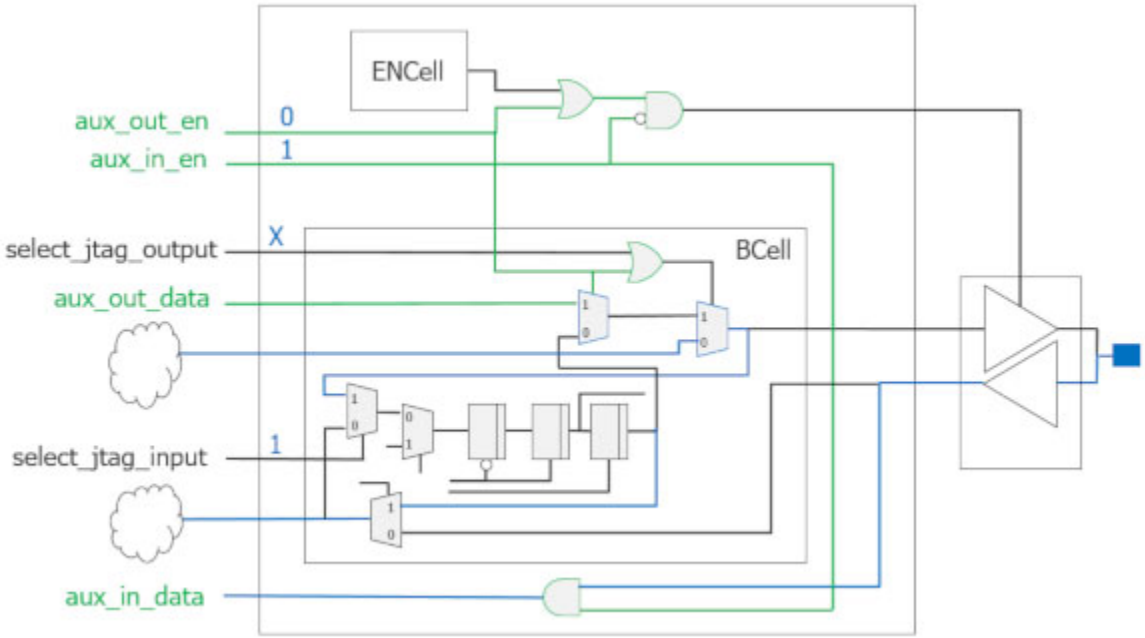
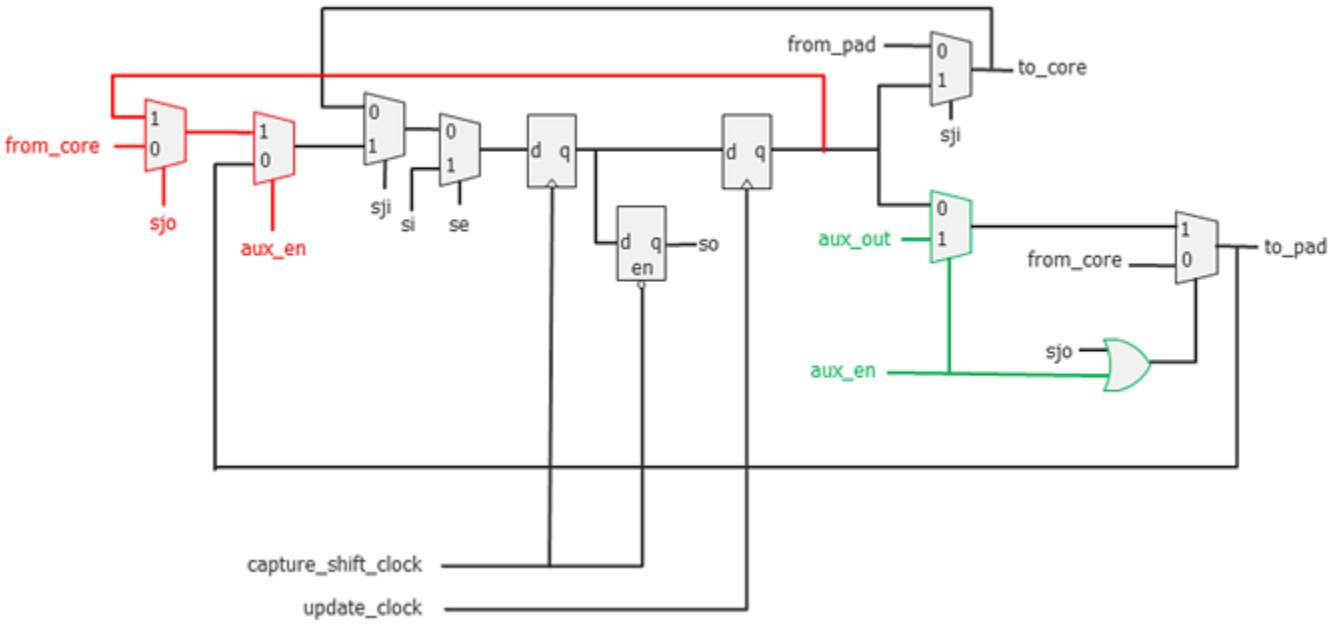


Figure 13-50 shows a bidirectional BoundaryScan cell with an auxiliary output and logic test support. The muxes depicted in red prevent capturing the from_core signal during logic test mode so that the from_core signal can be connected to DFT hardware for logic test.

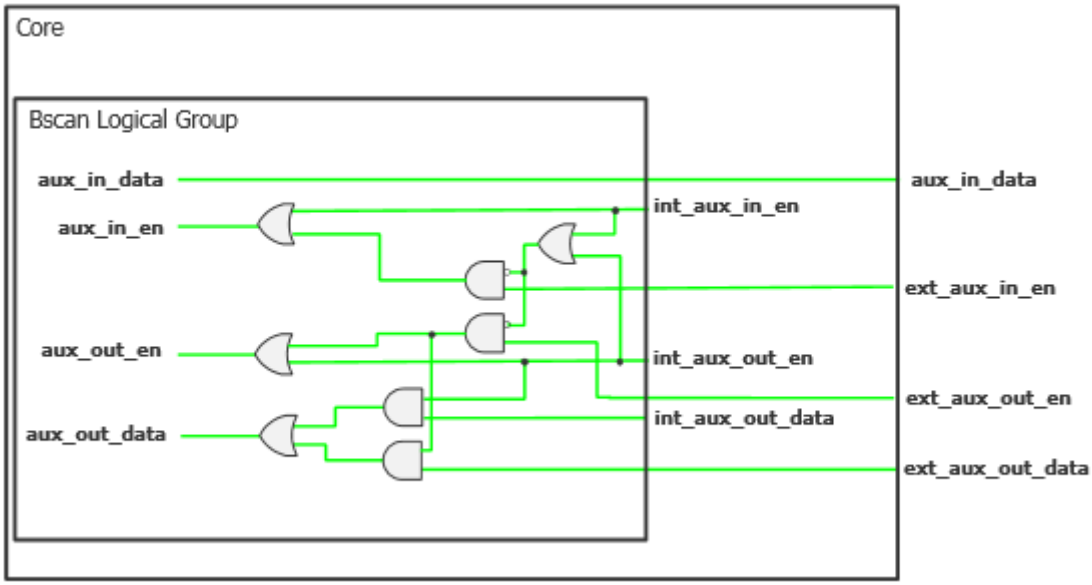
Figure 13-67. Bidirectional Port With AuxOut



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-51 shows the extra logic that is added when the same port is used for both internal and external auxiliary data in the EmbeddedBoundaryScan/AuxiliaryInputOutputPorts wrapper. In the output direction, two data_out input pins are created on the logical grouping module. One of those inputs is used to make auxiliary output connections for elements found within the block and the other input is connected to an input of the block to make auxiliary output connections for elements found outside the block. In the input direction, the same data output port on the logical grouping module is used for providing auxiliary data inputs to elements located inside and outside the block. Both directions have internal and external enable pins. The internal pins are used by the elements inside the blocks and the external pins are used by the elements outside the blocks. For example, if you call add_dft_modal_connections inside the block, it makes connections to the internal pins; and if you call add_dft_modal_connections outside the block, it makes connections to the external pins.

Figure 13-68. Extra Logic Added When a Port is Used Both Internally and Externally



Arguments

- auxiliary_input_ports : *port_name_pattern*, ... ;

  A property that specifies a list of port name patterns to define a group of input or inout ports as auxiliary input ports. The same inout port can be both an input and an output auxiliary port. The auxiliary inputs are left tied on the logic group module to be used by test logic found inside the chip.

  The *port_name_pattern* values can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional, and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- auxiliary_output_ports : *port_name_pattern*, ... ;

  A property that specifies a list of port name patterns that defines a group of output or inout ports as auxiliary output ports. The same inout port can be both an input and an output auxiliary port. The auxiliary output ports are left dangling on the logic group module to be used by test logic found inside the chip.

  The *port_name_pattern* can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

- internal_auxiliary_input_ports : *port_name_pattern*, ... ;

  A property that specifies a list of port name patterns to define a group of input or inout ports as internal auxiliary input ports. The same inout port can be both an input and an output auxiliary port. An internal auxiliary input port differs from an external one in how the auxiliary ports on the boundary scan logical group module are connected. When internal, they are left dangling on the logic group module to be used by test logic found inside the block. When external, the ports are connected to primary input and output ports of the block to be used by test circuit outside the block.

  The *port_name_pattern* values can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

- internal_auxiliary_output_ports : *port_name_pattern*, ... ;

  A property that specifies a list of port name patterns that defines a group of output or inout ports as internal auxiliary output ports. The same inout port can be both an input and an output auxiliary port. An internal auxiliary output port differs from an external one in how the auxiliary ports on the boundary scan logical group module are connected. When internal, they are left tied on the instance of the logic group module to be used by test logic found inside the block. When external, the ports are connected to primary input ports of the block to be used by test circuit outside the block.

  The *port_name_pattern* can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

- external_auxiliary_input_ports : *port_name_pattern*, ... ;

  A property that specifies a list of port name patterns to define a group of input or inout ports as external auxiliary input ports. The same inout port can be both an input and an output auxiliary port. An internal auxiliary input port differs from an external one in how the auxiliary ports on the boundary scan logical group module are connected. When internal, they are left dangling on the instance of the logic group module to be used by test logic found inside the block. When external, the ports are connected to primary output ports of the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

block to be used by test circuit outside the block. The ports on the block follow the naming specified by the Interface/auxiliary_* properties.

The *port_name_pattern* can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

- external_auxiliary_output_ports : *port_name_pattern*, ... ;

  A property that specifies a list of port name patterns that defines a group of output or inout ports as external auxiliary output ports. The same inout port can be both an input and an output auxiliary port. An internal auxiliary output port differs from an external one in how the auxiliary ports on the boundary scan logical group module are connected. When internal, they are left tied on the instance of the logic group module to be used by test logic found inside the block. When external, the ports are connected to primary input ports of the block to be used by test circuit outside the block. The ports on the block follow the naming specified by the Interface/auxiliary_* properties.

  The *port_name_pattern* can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

Examples

The following example defines the GPIO00 to GPIO31 ports as both auxiliary input and output ports. The first 16 are usable by test logic that is both internal and external to the block. The last 16 are only used by test logic that is external to the block. The full flexibility is preserved such that each test mode is able to use <*N*> of them as auxiliary output ports and the rest as auxiliary inputs, where <*N*> can be different for each test mode.

```
DftSpecification(my_chip,rtl) {
  EmbeddedBoundaryScan {
    AuxiliaryInputOutputPorts {
      internal_auxiliary_input_ports  : GPIO%2d[0:15] ;
      internal_auxiliary_output_ports : GPIO%2d[0:15] ;
      external_auxiliary_input_ports  : GPIO%2d[0:31] ;
      external_auxiliary_output_ports : GPIO%2d[0:31] ;
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

EnableGroups

Specifies the creation of enable groups using either the ports in the fanout of a functional enable signal or a specified port list.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan | EmbeddedBoundaryScan {
    EnableGroups {
      EnableGroup(name) {
        capture_core_signal     : on | off ;
        internal_enable_signal : pin_or_net_name ;
        port_list               : port_name_pattern, ... ;
        insert_before_port      : port_name ;
        insert_after_port       : port_name ;
      }
    }
  }
}
```

Description

A wrapper that specifies the creation of enable groups using either the ports in the fanout of a functional enable signal or a specified port list. The properties insert_before_port and insert_after_port are used to specify the location of the enable cells associated with the group. The property capture_core_signal is used to prevent the cell from capturing the core signal during INTEST or during logictest when the boundary scan is used as isolation.

Arguments

- capture_core_signal : on | off ;

  A property that specifies if the enable boundary-scan cells associated with the group are allowed to capture the core signal during INTEST or during the capture pulse when you reuse it for logictest isolation. See the max_segment_length_for_logictest property in the BoundaryScan wrapper for information about the logictest usage. When specified to off, the enable boundary-scan cell captures itself instead of capturing the core signal. This may be useful if you are using LogicBIST and the functional enable is X during the logictest mode.

- internal_enable_signal : *pin_or_net_name* ;

  A property that references an internal signal inside the chip that directly controls the enable pin of a group of output pad cells. Specifying this enable signal is equivalent to listing all the ports controlled by that enable signal in the port_list property. The specification of the internal_enable_pin and port_list properties are mutually exclusive. If the number of output pad cells controlled by the enable signal is larger than the specified outputs_per_enable_cell property, the fanout of the enable cell is split into smaller groups to meet the specified outputs_per_enable_cell value. One enable boundary-scan cell is added per group.

- port_list : *port_name_pattern*, ... ;

  A property that associates a group of output or inout ports with a given enable group. The *port_name_pattern* values can include an asterisk "*" as a wildcard and %#d[lb:rb] as a

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

counter. The # entry is optional and, when specified, is an integer specifying the minimum number of digits the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

The specification of the internal_enable_pin and port_list properties are mutually exclusive. If the number of output pad cells specified in the list is larger than the specified outputs_per_enable_cell value, the list is split into smaller groups to meet the specified outputs_per_enable_cell value. One enable boundary-scan cell is added per group.

You do not need to specify an enable group for all output and inout ports. For the tri-state inout and output ports not explicitly associated to an enable group, one or more additional enable groups is inferred. The inferring is done to match the common functional enable signal sourcing a group of pads. A given group can further be split to satisfy the specified outputs_per_enable_cell value described in the following sections:

- BoundaryScan — outputs_per_enable_cell property
- EmbeddedBoundaryScan — outputs_per_enable_cell property

- insert_before_port : *port_name* ;

  A property that specifies the position of the enable boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the insert_before_port property is mutually exclusive with the specification of the insert_after_port property. If neither the insert_before_port nor insert_after_port properties are specified, the enable boundary-scan cell is inserted before the first boundary cell that is associated with a port controlled by the enable cell.

- insert_after_port : *port_name* ;

  A property that specifies the position of the enable boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the insert_before_port property is mutually exclusive with the specification of the insert_after_port property. If neither the property insert_before_port nor insert_after_port properties are specified, the enable boundary-scan cell is inserted before the first boundary cell that is associated with a port controlled by the enable cell.

Examples

The following example defines three enable groups. The first one is associated with the ports in the fanout of the functional enable signal “core_i/en”. The second one is associated with all ports matching the pattern “OUT*”. The third group is associated with all other remaining ports as it does not specify either the internal_enable_signal nor the port_list property.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(my_chip,rtl) {
  BoundaryScan {
    EnableGroups {
      EnableGroup(g1) {
        internal_enable_signal : core1/en ;
      }
      EnableGroup(g2) {
        port_list : OUT* ;
      }
      EnableGroup(g3) {
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

InternalBScanCells

Specifies the insertion of internal boundary-scan cells that are not associated with ports but instead are used to control and/or observe internal nodes inside the design.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan | EmbeddedBoundaryScan {
    InternalBScanCells {
      InternalBScanCell(name) {
        insert_before_port : port_name ;
        insert_after_port  : port_name ;
        safe_value         : 0 | 1 | X ;
        type               : control | observation | both ;
        connection         : pin_or_net_name ;
        multiplexing       : on | off | auto ;
      }
    }
  }
}
```

Description

A wrapper that specifies the insertion of internal boundary-scan cells that are not associated with ports but instead are used to control and/or observe internal nodes inside the design.

Arguments

- *name*

  A string that uniquely identifies the internal boundary-scan cell in the BSDL file. The string starts with a letter and may be followed by any numbers of letters, numbers, or underscores. The string must be unique across all InternalBScanCell wrappers. The name of internal boundary-scan cells are mapped to the cell position inside the boundary scan chain using an attribute called TESSENT_INTERNAL_CELL_LABELS as shown in this example. The two InternalBScanCell wrappers have the names “my_name1” and “my_name2”.

  ```
  -- BSDL local extensions
  attribute TESSENT_INTERNAL_CELL_LABELS: BSDL_EXTENSION;
  attribute TESSENT_INTERNAL_CELL_LABELS of TOP: entity is
  -- Label  Cell
  "((my_name1 , 36  )," &
  " (my_name2 , 30  ))";
  ```

- insert_before_port : *port_name* ;

  A property that specifies the position of the internal boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the insert_before_port property is mutually exclusive with the specification of the insert_after_port property. If neither the insert_before_port nor insert_after_port properties are specified, the internal boundary-scan cell is inserted at the end of the boundary scan chain near scan_out in the order they were specified. The location

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

of the internal boundary-scan cell is reflected in the BSDL file using the TESSENT_INTERNAL_CELL_LABELS attribute.

- insert_after_port : *port_name* ;

  A property that specifies the position of the internal boundary-scan cell. This property can only be used when the pin_order_file property inside the BoundaryScan wrapper is specified. The specification of the insert_after_port property is mutually exclusive with the specification of the insert_before_port property. If neither the property insert_before_port nor insert_after_port properties are specified, the internal boundary-scan cell is inserted at the end of the boundary scan chain near scan_out in the order they were specified. The location of the internal boundary-scan cell is reflected in the BSDL file using the TESSENT_INTERNAL_CELL_LABELS attribute.

- safe_value : 0 | 1 | X ;

  A property that specifies a safe value for the internal boundary-scan cell. This value is reflected into the BSDL file. If a non-X safe value is specified, the BoundaryScan signoff and manufacturing pattern set never loads a different value into that register during any of its RunTest specifications.

- type : control | observation | both ;

  A property that specifies if the internal boundary-scan cell is to be used to control and/or observe an internal node.Table 13-7 shows the hardware inserted into the design based on the value of the type and multiplexing property.

- connection : *pin_or_net_name* ;

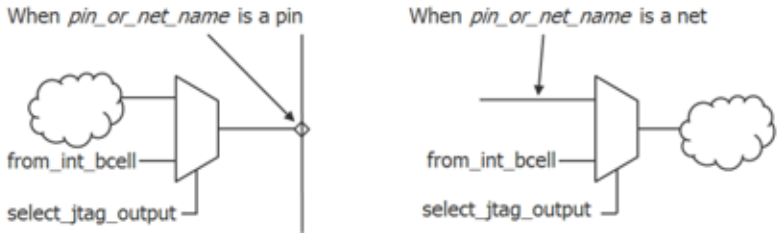
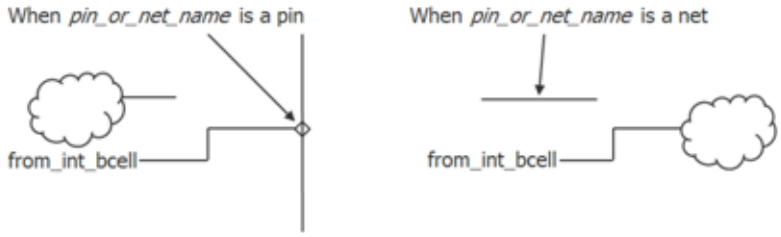
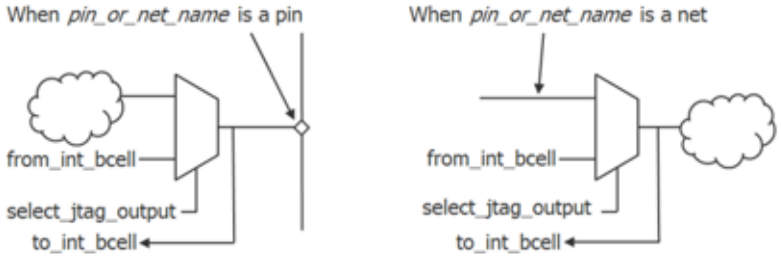
  A property that specifies the name of a pin or a net to control and/or observe. The *parent_instance* of *pin_or_net_name* is auto-uniquified if not unique.

- multiplexing : on | off | auto ;

  A property that is only relevant when the property type is set to “control” or “both”. A multiplexer is inserted as shown in Table 13-7. When the value is “auto”, the multiplexer is inserted only when the has_functional_source attribute of the specified *pin_or_net_name* is set to “true”. See Pin and Net for more information about this attribute.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-11. Connections To and From Internal Boundary Scan Cells

type	description
control	If multiplexing is specified as “on” or “auto” and the has_functional_source attribute for *pin_or_net_name* is set to “true”, the driver is intercepted with a multiplexer located in the *parent_instance* of the *pin_or_net_name*. A net is intercepted at its source while a pin is intercepted at its destination. The internal boundary-scan cell captures its own output.  If multiplexing is specified as “off” or “auto” and the has_functional_source attribute for *pin_or_net_name* is set to “false”, the original driver is disconnected and replaced by the signal from the internal boundary-scan cell. The internal boundary-scan cell captures its own  output.
observation	The specified *pin_or_net_name* is connected to the input of the internal boundary-scan cell.
both	If multiplexing is specified as “on” or “auto” and the has_functional_source attribute for pin_or_net_name is set to “true”, the driver is intercepted with a multiplexer located in the *parent_instance* of the *pin_or_net_name*. A net is intercepted at its source while a pin is intercepted at its destination. The observation net is taken after the multiplexer. The functional source of the pin_or_net_name is observable when select_jtag_output is  low. If multiplexing is specified as “off” or “auto” and the has_functional_source attribute for *pin_or_net_name* is set to “false”, the original driver is disconnected and replaced by the signal from the internal boundary-scan cell. The internal boundary-scan cell captures its own output.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example creates an internal boundary-scan cell that is used to control the u1/pll_reset signal using the boundary scan chain. The specified safe value of 0 ensures this cell is never loaded with a 1 during any of the RunTest in the PatternsSpecification or with any board test patterns created from the BSDL file.

```
DftSpecification(module_name,id) {
  BoundaryScan | EmbeddedBoundaryScan {
    InternalBScanCells {
      InternalBScanCell(mycell) {
        insert_before_port : PortA ;
        safe_value         : 0 ;
        type               : control ;
        connection         : u1/pll_reset ;
        multiplexing       : auto ;
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

InternalBScanSegments

Specifies the position of an existing internal boundary scan segment in the boundary scan chain.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan | EmbeddedBoundaryScan {
    InternalBScanSegments {
      InternalBScanSegment(instance_name) {
        insert_before_port      : port_name ;
        insert_after_port       : port_name ;
        logical_group_name      : logical_group_name ;
      }
    }
  }
}
```

Description

A wrapper that is used to specify the position of an existing internal boundary scan segment in the boundary scan chain. When the BScanSegment contains cells associated with a port of the design, the position of the BScanSegment and its *logical_group_name* is taken from the position and the logical group of the first port associated with the BScanSegment in the pin_order_file. In the case of pure internal boundary scan segments, their position and logical grouping name cannot be extracted from associated ports because they are not associated with ports. The logical group name is not used to create logical grouping modules for existing boundary scan segments because they are already grouped into modules. Instead, they are used to refer to the segments that use the bypass_logical_group inside the BondingConfigurations wrapper.

This wrapper is automatically created when the create_dft_specification command is invoked if a BScanSegment description is found for a module. You must delete this wrapper prior to invoking process_dft_specification if you do not want it to be part of the boundary scan chain. Refer to the "set_design_sources -format bscan_lib" command description for information about automatically finding the BScanSegment files without having to explicitly read them in using the read_core_descriptions command. You specify the module matching options using the set_module_matching_options command.

Arguments

- insert_before_port : *port_name* ;

  A property that specifies the position of the internal boundary segment relative to the position of the boundary-scan cell associated with a port. The specification of the insert_before_port is mutually exclusive with the specification of the insert_after_port property. When neither the insert_before_port nor insert_after_port properties are specified, the boundary scan segment is placed at the end of the boundary scan chain near the scanout. You can only specify this property if the pin_order_file property is used inside the BoundaryScan wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- insert_after_port : *port_name* ;

  A property that specifies the position of the internal boundary segment relative to the position of the boundary-scan cell associated with a port. The specification of the insert_before_port is mutually exclusive with the specification of the insert_after_port property. When neither the insert_before_port nor insert_after_port properties are specified, the boundary scan segment is placed at the end of the boundary scan chain near the scanout. You can only specify this property if the pin_order_file property is used inside the BoundaryScan wrapper.

- logical_group_name : *logical_group_name* ;

  A property used to associate a *logical_group_name* to the existing internal boundary scan segment. This name is not used to create a logical grouping module as the existing boundary scan segment is already grouped into a module. It is used instead to refer to the segment using the bypass_logical_group property inside the BondingConfigurations wrapper.

Examples

The following example defines a design instance as an internal boundary scan segment. It is given a logical group name so that it can be bypassed in some bounding options. A BscanSegment definition must exist for the module associated with instance u1/u2.

```
DftSpecification(module_name,id) {
  BoundaryScan {
    InternalBScanSegments {
      InternalBScanSegment(u1/u2) {
        logical_group_name      : seg1 ;
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BondingConfigurations

Specifies different boundary scan chain configurations based on bonding variations of a single die into multiple packages.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan {
    BondingConfigurations {
      BondingConfiguration(name) {
        enable_signal           : pin_or_net_name ;
        part_number_code        : binary ; // default: 16'h0
        version_code            : binary ; // default: 4'h0
        unused_ports            : port_name_pattern, ...  ;
        bypassed_logical_groups : logical_group_name, ... ;
        active_logical_groups   : logical_group_name, ... ;
        TcdBscanSegment(instance_name) {
          segment_selection : segment_selection_name;
        }
      }
    }
  }
}
```

Description

A wrapper that specifies different boundary scan chain configurations based on bonding variations of a single die into multiple packages. You need to have a one-hot set of decoded signals in which each signal is high for one bonding configuration and low for all the others. In a given bonding configuration, you can chose not to connect some port on the die to package pins. In this case, the boundary scan cells are described as internal boundary scan cells in the BSDL file associated with that bonding configuration unless you chose to group those cells in a logical grouping module and specify the module in the list of logical group modules to bypass. You typically need to bypass a segment of the boundary scan cell when the segment is completely powered down in the given package.

You can also specify a different part_number_code and version_code for each of the bonding configurations. If you are using a third-party TAP controller, the controller must capture the part_number and version codes from ports at the boundary of the TAP controller. You must define those ports as DataInPorts, as described in the "Requirements on a TAP to be Usable for BoundaryScan" section, and they must be part of the CaptureSource of the device_id register.

Arguments

- *name*

  A string that uniquely identifies each bonding configuration. This string is used in the name of the BSDL file associated with each bonding configuration.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- enable_signal : *pin_or_net_name* ;

  A mandatory property that points to a signal inside the design that is 1 when the given bonding configuration is used and 0 for all other bonding configurations.

  The *pin_or_net_name* can be a DftSignal(*name*) that you previously registered with the register_static_dft_signal_names command as a global DFT control signal and added with the add_dft_signals command.

  You must ensure the enable_signal has the correct value during simulation. The current enable_signal should be 1 and all others 0. If it is a DftSignal, you can use the "create_patterns_specification -separate_tap_and_bscan_patterns" command to create an extra pattern for the BoundaryScan register tests, and the Patterns/DftControlSettings property to set the value. For a pin or net, you should set the value during simulation.

  Caution

  If the source of the BondingConfiguration enable_signal is dynamic (such as a DftSignal), a BondingConfiguration-specific device ID (part_number_code and version_code properties) is not allowed. It is not permitted to drive a BondingConfiguration that affects the device ID with a signal that is dependent on IJTAG logic or RESET. BondingConfiguration can alter the device ID only if you use a static enable signal—tied to a value or fused.

  If you specify an enable_signal, the tool returns the following warning:

  ```
  //  Warning: /DftSpecification(car,gate)/BoundaryScan/
  BondingConfigurations/BondingConfiguration(default)/enable_signal
  //  You don't need to specify an enable signal. This entry will be
  //  ignored.
  ```

  You only need an enable_signal if you define bypassed_logical_groups, active_logical_groups, a part_number_code, or a version_code. A bonding configuration that has at least one of those needs an enable_signal.

- part_number_code : *binary* ;

  An optional property that defines the part_number_code to associate with the given bonding configuration. The generation of the part number code per bonding configuration is done inside the boundary scan interface module as shown in Figure 13-52. If you are using a third-party TAP to host the boundary scan chain, it must have a part_number_code DataInPort and that port must be part of the CaptureSource of the device_id register as described in the "Requirements on a TAP to be Usable for BoundaryScan" section.

  If this property is not specified, its value is taken from the specified part_number_code property in the Tap wrapper referenced by the ijtag_host_interface property of the BoundaryScan wrapper or from the tied value on the part_number_code DataInPort when using a TAP that already exists in the design.

- version_code : *binary* ;

  An optional property that defines which version id code to associate with the given bonding configuration. The generation of the version id code per bonding configuration is done

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

inside the boundary scan interface module as shown in Figure 13-52. If you are using a third-party TAP to host the boundary scan chain, it must have a version_code DataInPort and that port must be part of the CaptureSource of the device_id register as described in the "Requirements on a TAP to be Usable for BoundaryScan" section.

If this property is not specified, its value is taken from the specified version_code property in the Tap wrapper referenced by the ijtag_host_interface property of the BoundaryScan wrapper or from the tied value on the version_code DataInPort when using a TAP that already exist in the design.

- unused_ports : *port_name_pattern*, ... ;

  A property that specifies a list of ports that are not contacted in the given bonding configuration but whose boundary scan cells remain part of the boundary scan chain. Those cells are described as internal boundary scan cells in the BSDL. If the boundary scan cells are powered down in the given bonding configuration, you must instead group the boundary scan cells into one or many logical groups and specify that they are to be bypassed by listing them in the bypassed_logical_groups property or not listing them in the active_logical_groups property.

  The port_name_pattern can include an asterisk "*" as a wildcard and %#d[lb:rb] as a counter. The # entry is optional and when specified is an integer specifying the minimum number of digit the number must have. For example, ABC%2d[0:2] matches ABC00, ABC01, and ABC02.

- bypassed_logical_groups : *logical_group_name*, ... ;

  A property used to list one or more logical grouping modules that are to be bypassed in the current bonding configurations. You use this option if you want to optimize the length of the boundary scan chain for each bonding configuration, or if the segment is completely powered down in the current bonding configuration. You can choose to declare the boundary scan cells associated with the non-contacted ports as internal boundary scan cells using the unused_ports property, but this is only possible if the boundary scan chain segment remains alive and powered in the current bonding configuration.

  This option is mutually exclusive with the active_logical_groups property.

- active_logical_groups : *logical_group_name*, ... ;

  A property used to list one or more logical grouping modules that are to be active in the current bonding configurations. You use this option if you want to optimize the length of the boundary scan chain for each bonding configuration, or if the segments specified here are the only ones powered up in the current bonding configuration. You can choose to declare the boundary scan cells associated with the non-contacted ports as internal boundary scan cells using the unused_ports property, but this is only possible if the boundary scan chain segment remains alive and powered in the current bonding configuration.

  This option is mutually exclusive with the bypassed_logical_groups property.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- TcdBscanSegment(*instance_name*) { segment_selection: *segment_selection_name* }

  An instance name and segment selection name pair that specifies bypassed segments for lower-level modules with embedded boundary scan. You must specify this for every BondingConfiguration if there are any lower-level embedded boundary scan segments that can be bypassed.

Examples

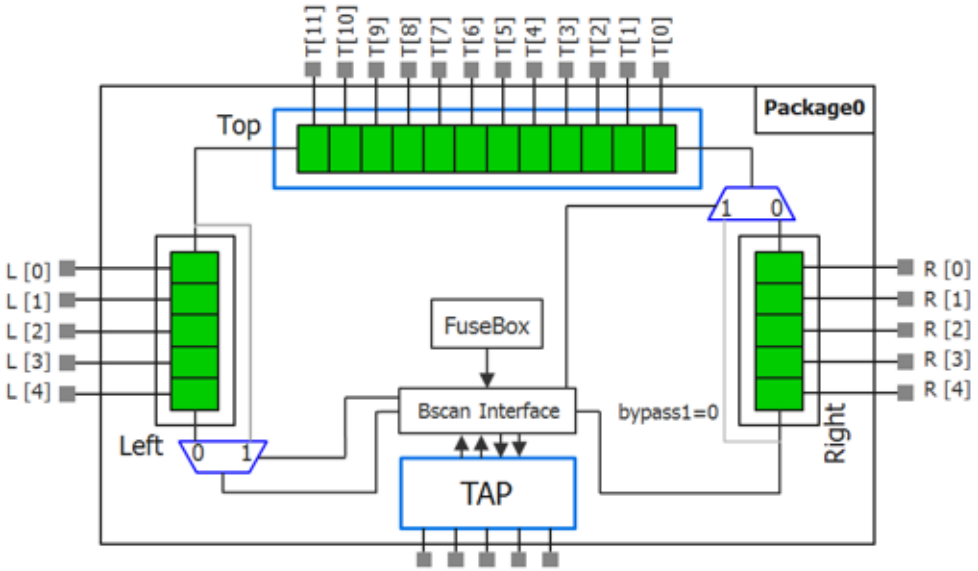
The following example shows three different bonding configurations.

```
DftSpecification(my_chip,rtl) {
  BoundaryScan {
    ijtag_host_interface : HostScanInterface(existing_tap) ;
    BondingConfigurations {
      BondingConfiguration (package0) {
        enable_signal           : core/bounding_decode/pkg0 ;
      }
      BondingConfiguration (package1) {
        enable_signal           : core/bounding_decode/pkg1 ;
        unused_ports            : L[*],T1,T3,T5,T7,T9,T11 ;
        part_number_code        : 16'h0001 ;
        bypassed_logical_groups : Left ;
      }
      BondingConfiguration (package2) {
        enable_signal           : core/bounding_decode/pkg2 ;
        unused_ports            : T6,T7,T8,T9,T10,T11 ;
        part_number_code        : 16'h0002 ;
        bypassed_logical_groups : Right ;
      }
    }
  }
}
```

The bonding configuration “package0” has no unused pins and no bypassed logical groups. The boundary scan register contains all boundary scan cells in the design as shown in Figure 13-69. No boundary scan cells associated with unused ports are converted to internal boundary scan cells. The BSDL file may contain internal boundary scan cells if some were specified with the InternalBScanCells and InternalBScanSegments wrappers. Those cells are internal in all configurations unless they are bypassed. The part number code and the version code are not specified for this bonding configuration. Because this example uses a TAP already present in the design, the default values are taken from the tied values present on the part_number_code and version_code DataInPorts on the TAP where it is instantiated in the design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

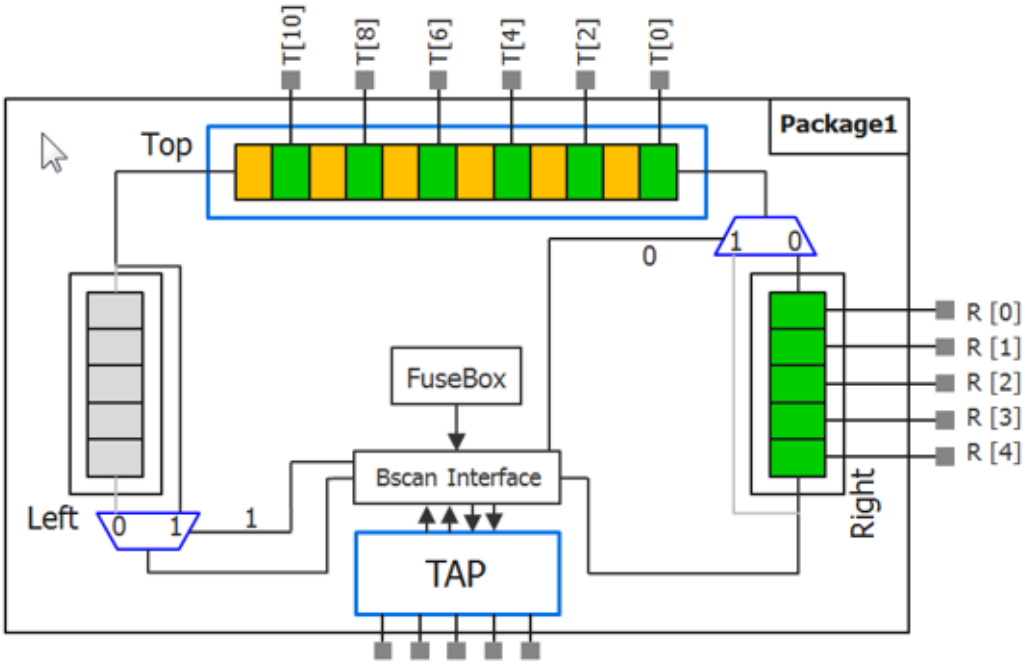
Figure 13-69. package0 Bonding Configuration Example



The bonding configuration “package1”, shown in Figure 13-70, has several unused pins and a bypassed logical group called “Left”. The bypassed cells are shown in gray and the cells converted to internal cells are shown in orange. A different device_id code is specified for the configuration. Notice that the ports matching L[*] were specified with the unused_ports property. Those ports are already associated with a logical group that is declared as bypassed. This is a redundant specification. The cells associated with the L[*] ports are not declared as internal boundary scan cells in the BSDL file because they are bypassed. The cell associated with the ports T1,T3,T5,T7,T9,T11 are declared as internal boundary scan cells as they are not bypassed.

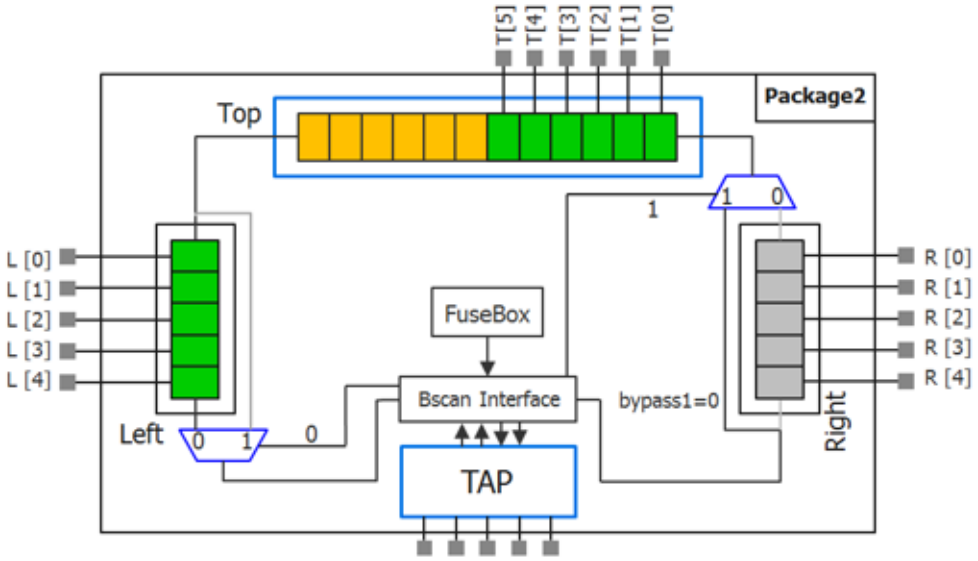
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-70. package1 Bonding Configuration Example



The bonding configuration “package2”, shown in Figure 13-71, has several unused pins and one bypassed logical group called “Right”. In this example, a set of fuses is used to program the bonding configuration. The one-hot encoded enable signals could have also been decoded from ports that get tied high and low in each respective package. When you want to simulate the BoundaryScan patterns using the various bonding configurations, you need to make sure the right bonding configuration is enabled. See the description of the bonding_configuration property inside the BoundaryScan wrapper of the PatternsSpecification wrapper.

Figure 13-71. package2 Bonding Configuration Example



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

UserInstructions

Specifies a user instruction and an optional user register that is to be added in the BSDL file.

Usage

```
DftSpecification(module_name,id) {
  BoundaryScan {
    UserInstructions {
      UserInstruction(instr) {       // Repeatable
        tap_host_scan_interface_id  : Tap(id)/HostIjtag(id) ;
        instruction_codes           : binary, ... ;
        bsdl_visibility             : public | private ;
        user_register               : user_reg[int] ;
        capture_value               : sized_binary_string ;
      }
    }
  }
}
```

Description

An optional wrapper that is used to specify a user instruction and an optional user register that is to be added in the BSDL file. The UserInstruction wrapper *instr* name is listed in the INSTRUCTION_OPCODE section of the BSDL file. If the *instr* is specified as being private, it is also listed in the INSTRUCTION_PRIVATE section.

If a user register associated with *instr* is specified, it is listed in the REGISTER_ACCESS section of the BSDL file. UserInstruction wrappers that specify user_register properties must have unique *user_reg* names or use the same *user_reg* name and *int* size.

The opcode(s) for the user instruction can be defined directly using the instruction_codes property, or they can be referenced from those defined in the Tap/HostIjtag wrapper for the current DftSpecification/IjtagNetwork using the tap_host_scan_interface_id property. Only one of these methods can be used for each UserInstruction wrapper.

Arguments

- UserInstruction(*instr*)

  A repeatable wrapper used to specify the user instruction name. The UserInstruction wrapper *instr* name is listed in the INSTRUCTION_OPCODE section of the BSDL file and must be a valid BSDL name.

- UserInstruction(*instr*)/tap_host_scan_interface_id : Tap(*id*)/HostIjtag(*id*) ;

  A property that specifies the user instruction opcode(s) are to be referenced from those defined in the Tap/HostIjtag wrapper for the current DftSpecification/IjtagNetwork. The Tap/HostIjtag wrapper must exist and the Tap(*id*) must be the TAP that hosts the boundary scan hardware. This property is mutually exclusive with the instruction_codes property.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- UserInstruction(*instr*)/instruction_codes : *binary*, ... ;

  A property that specifies the binary opcode(s) associated with the *instr* user instruction. This property is mutually exclusive with the tap_host_scan_interface_id property.

- UserInstruction(*instr*)/bsdl_visibility : public | private ;

  A property that specifies whether the instruction name *instr* is to only be used by the manufacturer, or to be available publicly. The default setting is public. If set to private, the *instr* is listed in the INSTRUCTION_PRIVATE section of the BSDL file.

- UserInstruction(*instr*)/user_register : *user_reg*[*int*] ;

  An optional property that specifies a user register that is associated with the user instruction *instr*. The parameter *user_reg* specifies the name of the user register and must be a valid BSDL name.

  The parameter *int* specifies the size of the user register, and must be specified if the size cannot be determined from the tap_host_scan_interface_id or if the specified register is not one of BOUNDARY, BYPASS or DEVICE_ID. When tap_host_scan_interface_id is specified together with user_register, the parameter *int* is checked against the selected part of the IJTAG network. If the selected part has a fixed length, it needs to match *int*. If the selected part controls a configurable scan path, or if the length cannot be determined, the user needs to specify the length as *int* in the user_register property. The parameter *int* is optional if the length can be determined from the selected IJTAG network.

  The specified *user_reg* is listed in the REGISTER_ACCESS section of the BSDL file. UserInstruction wrappers that specify user_register properties must have unique *user_reg* names or use the same *user_reg* name and *int* size.

  If user_register is not specified, then bsdl_visibility must be set to private.

- UserInstruction(*instr*)/capture_value : *sized_binary_string*;

  If specified, this property is added to the REGISTER_ACCESS section of the BSDL file. Example:

  ```
  attribute REGISTER_ACCESS of TESTPRDCT:entity is
  "AAAA[12] (BBB CAPTURES 000011110000)";
  ```

  The value is a sized binary string consisting of 1s, 0s, and Xs; for example “8’b0000X1X1” and must match the size of the register.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

LogicBist

A wrapper that specifies the information used by the process_dft_specification command to build the Hybrid TK/LBIST IP and insert the IP into the design.

Usage

```
DftSpecification (module_name,id) {
  LogicBist {
    ijtag_host_interface : name ; // default: tied_scan_interface
    Controller(id) {
    }
    NcpIndexDecoder {
    }
  }
}
```

Description

A wrapper that specifies the creation and the optional insertion of Hybrid TK/LBIST controllers in a design.

Arguments

- ijtag_host_interface : *name* ;

  This property is used to specify the host scan interface that the Hybrid TK/LBIST controllers connect to. You can specify the host name as specified by the HostScanInterface parameter within the IjtagNetwork wrapper with two exceptions: “None" is unsupported. Instead, use the tied_scan_interface parameter, which means that all IJTAG signals are tied to 0 and you have to properly reconnect those pins. You can use a post insertion script or do it manually after process_dft_specification.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Controller

This wrapper is used to add and specify properties for the Hybrid TK/LBIST controller generated with process_dft_specification.

Usage

```
DftSpecification (module_name,id) {
  LogicBist {
    Controller(id) {
      parent_instance            : name ;
      leaf_instance_name         : name ;
      burn_in                    : on | off ;
      extest_lbist               : on | off | auto ;
      ext_mode_edt_present       : on | off | auto ;
      associated_edt             : edt_spec ... // repeatable
      use_ssn_bus_clock_as_test_clock_bypass : on | off | auto ;
      self_test                  : on | off ;
      pre_post_shift_dead_cycles : int ;  // default: 8
      scan_signals_bypass        : on | off ; // default: on
      DesignInstance (instance_name) {
        associated_edt           : edt_spec ... // repeatable
      }
      ControllerChain {
      }
      SingleChainForDiagnosis {
      }
      ShiftCycles {
      }
      CaptureCycles {
      }
      PatternCount {
      }
      ChainTestPatternCount {
      }
      WarmupPatternCount {
      }
      AsyncSetResetPatternCount {
      }
      NcpOptions {
      }
      PostScanActivityControl {
      }
      Interface {
      }
      Connections {
      }
      SetLoadUnloadTimingOptions {
      }
    }
  }
}
```

Description

Use this wrapper to describe the shared Hybrid TK/LBIST controller instance. When using this wrapper, you must specify the EDT wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

There is no way to specify multiple LBIST controllers and mapping of Hybrid TK/LBIST blocks to the LogicBIST controller. All EDT blocks must be specified in the same insertion pass. Not all EDT controllers need to be hybrid EDT controllers. You use the property DftSpecification/EDT/Controller/LogicBistOptions/present to specify the EDT controllers that use the hybrid IP.

Arguments

- parent_instance : *name* ;

  This property is used to specify the parent instance. This is the name of the instance inside which you instantiate your hybrid IP. By default, the tool chooses the lowest common ancestor of all related EDT blocks.

- leaf_instance_name : *name* ;

  This optional property specifies the LogicBIST controller instance name. If not specified, the name is "<design_name>_<dft_spec_id>_tessent_lbist_inst", where <dft_spec_id> is the ID specified for the DftSpecification wrapper.

- burn_in : on | off ;

  This optional property specifies whether to enable burn-in test mode for the LogicBIST IP. The default is off. Refer to "Burn-In Test Mode" in the *Hybrid TK/LBIST Flow User's Manual* for details.

- extest_lbist : on | off | auto ;

  This optional property specifies whether the LBIST controller combines the lbist_en and ext_lbist_en DFT signals with an OR gate to enable reuse of the internal OCC and EDT of the core in the LBIST of its parent physical region. The default is auto.

  - auto: If you add the ext_lbist_en DFT signal using the add_dft_signals command, auto resolves to on. Otherwise, it resolves to off.
  - on: The LBIST Controller includes hardware to enable control of the internal instruments of the core in the LBIST of their parent physical regions. To enable control over embedded EDTs for the external mode of the core, the ext_mode_edt_present property must be set to "on".
  - off: The LBIST controller does not include hardware to enable control of the OCC or EDT in the LBIST of their parent physical regions.

  When extest_lbist is set to on, and the ext_lbist_en signal does not exist, the tool automatically infers the ext_lbist_en signal on the port when the current design level is sub-block or physical block.

  This option is available only in the TK/LBIST independent insertion flow.

- ext_mode_edt_present : on | off | auto

  This optional property specifies whether the LBIST controller includes hardware to enable the use of EDTs for the external mode of the core in the LBIST of its parent physical region. The default is auto. If the extest_lbist property is set to "on" and any hybrid EDT from the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

current physical region is assigned to external scan mode by its EDT/Controller/Connections/mode_enables property, then auto resolves to “on”; otherwise, it resolves to “off”.

This option is available only in the TK/LBIST independent insertion flow.

- associated_edt : edt_spec ...

  This property specifies hybrid EDTs from the current pass that are controlled by the LogicBist Controller. edt_spec is a repeatable list of EDT specifications that can include:

  - EDT controller IDs generated in the same pass.

  If the list is unspecified, it is automatically inferred with the following method:

  - If the ext_mode_edt_present property is set to “on”, then all EDTs from the current pass are added to the list, excluding the one for which the mode_enables property specifies the DFT signal for the external mode.
  - If the ext_mode_edt_present property is set to “off”, then all EDTs from the current pass are added to the list.

  This property is valid only in the TK/LBIST independent insertion flow.

- use_ssn_bus_clock_as_test_clock_bypass : on | off | auto

  This optional property specifies whether the generated LBIST controller’s ICL model supports usage of the SSN bus clock as test clock. The default is auto. This resolves to on if the corresponding property in SSN/DataPath/ScanHost wrapper is set to on or the defined test_clock DFT signal is connected with the ssn_bus_clock input of SSH. Otherwise, auto resolves to off.

- self_test : on | off ;

  This property specifies whether to enable Self-Test mode for the LogicBIST IP. The default is off. Refer to “Self-Test Mode” in the *Hybrid TK/LBIST Flow User’s Manual* for details.

- pre_post_shift_dead_cycles : *int* ;

  This optional property specifies how many dead cycles, at the shift clock period, are applied before and after the capture clocks are pulsed. The specified integer can be between 2 and 8, with the default being 8. Reducing the number of pause cycles reduce the test time but it may also make it harder to achieve timing closure of the test logic. To ensure that the shift and capture signals are staggered an integer value of 1 is not supported. Refer to “Programmable Shift and Capture Pause Cycles” in the *Hybrid TK/LBIST Flow User’s Manual* for details.

  Note

  This property cannot be specified when independent_insertion_flow is enabled. See “Independent Hybrid TK/LBIST Insertion Flow” in the Hybrid TK/LBIST Flow User’s Manual for more details.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- scan_signals_bypass

  This property enables you to control whether the LogicBist controller contains bypass muxes to intercept the scan control DFT signals scan_en, edt_update, and shift_capture_clock. This is a high-level instruction specifying whether the signals should be intercepted or only internally generated. The default is on.

- DesignInstance (*instance_name*) {}

  An optional wrapper that specifies a design instance inserted during a prior insertion pass in the current or child design levels. You can specify this wrapper more than once. Each instance is validated to be a block having LBIST-ready EDT controllers and the associated TCD file is loaded so that the information needed to create the LBIST IP is available. The id of the wrapper is an instance_name for child design levels or "." for the current design.

- DesignInstance (*instance_name*)/associated_edt : edt_spec ...

  This property specifies hybrid EDTs inserted during a previous insertion pass in the current or child design levels that are controlled by the LogicBist Controller. The edt_spec is a repeatable list of EDT specifications that can include:

  - EDT instance path names relative to the inst_name specified in DesignInstance().
  - EDT module names.

  If this property is unspecified, then the tool infers it in the following way:

  - If the DesignInstance wrapper specifies the current design level and the ext_mode_edt_present property is set to "off", then all EDTs from this design level are added to the list.
  - If the DesignInstance wrapper specifies the current design level and the ext_mode_edt_present property is set to "on", then all EDTs from this design level are added to the list, excluding the one for which the mode_enables property specifies the DFT signal for the external mode.
  - If the DesignInstance wrapper specifies a child design level, and it has an LBIST controller with the ext_mode_edt_present property set to "on", then all EDTs from this design level are added to the list for which the mode_enables property indicates the DFT signal for the external mode.
  - If the DesignInstance wrapper indicates a child design level and it does not have any LBIST controllers (lbist-ready blocks), then all hybrid EDTs from this design level are added to the list.

  This property is valid only in the TK/LBIST independent insertion flow.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ControllerChain

A wrapper that configures the Hybrid TK/LBIST Controller Chain Mode (CCM).

Usage

```
DftSpecification (module_name,id) {
  LogicBist {
    Controller(id) {
      ControllerChain {
        present                 : on | off ;
        clock                   : enum ; // legal: tck | edt_clock
        segment_per_instrument  : on | off ;
        max_segment_length      : int | unlimited; // int >= 32
        async_set_reset_gating  : always_on | load_unload_only ;
      }
    }
  }
}
```

Description

A wrapper that configures the CCM logic, which generates ATPG patterns that target the Hybrid TK/LBIST logic. See "Controller Chain Mode" in the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- present : on | off ;

  This property specifies whether to enable or disable the CCM. The default is off. When turned on, the tool creates RTL scan chain segments within the IP, so that it can calculate test coverage for the LogicBIST controller and Hybrid TK/LBIST blocks.

- clock : *enum* ;

  This property specifies whether the tool uses the tck or edt_clock clock during CCM. The default is edt_clock.

- segment_per_instrument : on | off ;

  An optional switch and literal pair that specifies whether, during IP insertion, the tool automatically connects controller scan chain segments within the IP blocks into one scan chain, or leaves them as segments. The default is on (leave as segments).

  This option cannot be set to off when an integer value is set for the max_segment _length property.

  This option must be set to on when the independent_insertion_flow property is enabled. You must segment the CCM chains if LBIST hosts the IJTAG subnetwork with no hybrid EDT placements.

- max_segment_length : int | unlimited ;

  This property specifies the maximum controller scan chain segment length in Controller Chain Mode. The minimum segment length is 32. The default is unlimited, meaning a single

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

scan chain segment for each instrument. When the total number of scan cells is fewer than the specified length, the tool creates a single chain.

- async_set_reset_gating : always_on | load_unload_only ;

  This property specifies whether the LBIST asynchronous set/reset is disabled during CCM. When set to always_on, the LBIST RTL gates the lbist_reset with ccm_en to disable the hardware default asynchronous set/reset during CCM.

  The default is load_unload_only.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SingleChainForDiagnosis

A wrapper used to configure the hybrid TK/LBIST single chain mode logic.

Usage

```
DftSpecification (module_name,id) {
  LogicBist {
    Controller(id) {
      SingleChainForDiagnosis {
        present                          : on | off ;
        parent_instance                  : name ;
        leaf_instance_name               : name ;
        skip_edt_blocks                  : on | off ;
      }
    }
  }
}
```

Description

You use this wrapper to configure the hybrid TK/LBIST single chain mode logic. See "Single Chain Mode Logic" in the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- present : on | off ;

  This property determines the "auto" value for the EDT/Controller/BypassChains/single_bypass_chain property. When the single_bypass_chain property is set to "auto," it takes the value of the present property.

  When the present property is "on" (the default), the EDT/Controller/BypassChains/single_bypass_chain property must be set to either "on" or "auto". Tessent Shell generates an error if the single_bypass_chain property is off while the present property is on.

  Note

  Unlike many other options, if this property is set to "off", then the parent_instance and leaf_instance_name properties are still meaningful. This is because the module controlling LBIST single chain for diagnosis is also responsible for any bypass logic of hybrid EDT blocks. Consequently, it gets instantiated even if the diagnostic logic for LBIST is disabled. Those properties still determine the name of the module.

- parent_instance : *name* ;

  This property is used to specify the parent instance under which the single chain for diagnosis logic is instantiated. When unspecified, this property inherits its value from the ../parent_instance property.

- leaf_instance_name : *name* ;

  This optional property specifies the single chain mode logic instance name. If not specified, the name is "<design_name>_<dft_spec_id>_tessent_single_chain_mode_logic_inst", where <dft_spec_id> is the ID specified for the DftSpecification wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- skip_edt_blocks : on | off ;

  This property specifies whether to insert a SIB in the module controlling LBIST single chain for diagnosis for each EDT block that allows optionally skipping all scan chains in a given EDT block during single chain unload. The default is on. For details, refer to "Single Chain Mode Diagnosis" in the *Hybrid TK/LBIST Flow User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ShiftCycles

A wrapper the configures the shift cycles for the hybrid TK/LBIST controller.

Usage

```
DftSpecification (module_name,id) {
  LogicBist {
    Controller(id) {
      ShiftCycles {
        counter_resolution : byte | bit ;
        max               : int ;
        hardware_default : int ;                    // default: max
      }
    }
  }
}
```

Description

A wrapper the configures the shift cycles for the hybrid TK/LBIST controller.

Arguments

- counter_resolution : byte | bit ;

  This parameter specifies whether the shift counter allows shift lengths that operate in multiples of 8 or 1. The default, “bit,” causes the tool to apply the number of shifts equal to the length of the longest scan chain. When you specify “byte,” the tool rounds up the length of the longest scan chain to the next multiple of 8.

  Note

  The value “byte”cannot be specified when independent_insertion_flow is enabled. See “Independent Hybrid TK/LBIST Insertion Flow” in the *Hybrid TK/LBIST Flow User’s Manual* for complete information.

- max : *int* ;

  This property specifies the maximum number of shift cycles for the controller. This is equivalent to the set_lbist_controller_options command’s “-max_shift_cycle” switch pair.

- hardware_default : *int* ;

  This property specifies the default maximum number of shift cycles for the controller. The default is max.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

CaptureCycles

A wrapper that configures the capture cycles for the hybrid TK/LBIST controller.

Usage

```
DftSpecification (module_name,id) {
  LogicBist {
    Controller(id) {
      CaptureCycles {
        max               : int ;
        hardware_default : int ; // default: max
      }
    }
  }
}
```

Description

A wrapper that configures the capture cycles for the hybrid TK/LBIST controller.

Arguments

- max : *int* ;

  This property specifies the maximum number of capture cycles for the controller. This is equivalent to the set_lbist_controller_options command's "-max_capture_cycles" switch pair.

- hardware_default : *int* ;

  This property specifies the default maximum number of capture cycles for the controller. The default is max.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

PatternCount

A wrapper that configures hybrid TK/LBIST pattern counter.

Usage

```
DftSpecification (module_name,id) {
  LogicBist {
    Controller(id) {
      PatternCount {
        max               : int ;
        hardware_default : int ; // default: max
      }
    }
  }
}
```

Description

A wrapper that configures hybrid TK/LBIST pattern counter register.

Arguments

- max : *int* ;

  This property specifies the maximum number of patterns for the controller. This is equivalent to the set_lbist_controller_options command's "-max_pattern_count" switch pair.

- hardware_default : *int* ;

  This property specifies the default maximum number of patterns for the controller. The default is max.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ChainTestPatternCount

Specifies the limits on the number of chain test patterns to apply.

Usage

```
DftSpecification (module_name,id) {
  LogicBist {
    Controller(id) {
      ChainTestPatternCount {
        max : int ; // default: 0
        hardware_default : int ; // default: 0
      }
    }
  }
}
```

Description

Use this optional wrapper to specify the maximum limit and a hardware default value for the number of chain test patterns to apply. This wrapper enables you to run a fixed number of chain test patterns after the warm-up patterns, and before the scan patterns. This wrapper creates new hardware in the generated LBIST controller, which is the chain_test_pattern_cnt register.

Arguments

- max : *int* ;

  An optional property that specifies the size of the chain_test_pattern_cnt register, which controls the maximum number of chain test patterns. By default, the tool does not create the register, so there are no chain test patterns.

- hardware_default : *int* ;

  An optional property that specifies an integer value to load into the chain_test_pattern_cnt register for running in hardware default mode. This is because in hardware default mode, the register loads a hardcoded value in the RTL instead of loading through the IJTAG network.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

WarmupPatternCount

This wrapper configures the hybrid TK/LBIST warm-up pattern count.

Usage

```
DftSpecification (module_name,id) {
  LogicBist {
    Controller(id) {
      WarmupPatternCount {
        max              : int ; // default: 255
        hardware_default : int ; // default: 0
      }
    }
  }
}
```

Description

This wrapper configures the hybrid TK/LBIST warm-up pattern count. See "Warm-Up Patterns" in the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- max : *int* ;

  This property specifies the maximum number of warm-up patterns for the controller. This is equivalent to the set_lbist_controller_options command's "-max_warmup_pattern_count" switch pair. The default is 255.

- hardware_default : *int* ;

  This property specifies the default maximum number of warm-up patterns for the controller. The default is 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

NcpOptions

A wrapper that specifies NCP-related hardware options for the LogicBIST controller.

Usage

```
DftSpecification (module_name,id) {
  LogicBist {
    Controller(id) {
      NcpOptions {      // following two properties are mutually exclusive
        count : int ;   // default: 1
        percentage_of_patterns_per_ncp : integer, ... ;
      }
    }
  }
}
```

Description

This wrapper differs from the NcpIndexDecoder wrapper in that NcpOptions specifies NCP-related hardware parameters for the LogicBIST controller whereas you use the NcpIndexDecoder wrapper to configure the index decoder hardware, such as clock waveforms and OCCs.

Specify the NcpOptions wrapper when you are inserting LogicBist and the NCP Index Decoder in different passes. You must specify the count property or the percentage_of_patterns_per_ncp property. The latter property applies the NCPs for the specified percentages of patterns. The count and percentage_of_patterns_per_ncp properties are mutually exclusive.

When you generate the NcpIndexDecoder in the same pass as the LogicBIST controller, Tessent Shell automatically populates the NcpOptions/count parameter from the number of NcpIndexDecoder/Ncp wrappers. In this case, specifying the NcpOptions wrapper results in an error.

Arguments

- count : *int* ;

  An integer that specifies to generate a LogicBIST controller that can support up to the specified integer of NCPs. The default is 1. The integer you specify must equal the number of NCPs as specified by the NcpIndexDecoder wrapper.

- percentage_of_patterns_per_ncp : *integer*, ... ;

  A repeatable integer that represents a percentage value. This value specifies, as a percentage, the number of times an NCP is applied every 256 patterns. The first NCP applies to the first integer specified in the percentage_of_patterns_per_ncp list, the second NCP to the second integer, and so on. The number of patterns each NCP is active in (within a 256 pattern range) is stored in the LogicBIST controller's hardware.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

PostScanActivityControl

An optional wrapper that specifies properties for the LogicBIST controller module cool-down pattern sequence.

Usage

```
DftSpecification (module_name, id) {
  LogicBist {
    Controller(id) {
      PostScanActivityControl {
        activity_type : auto | none | cooldown | unlimited;
        CoolDownPatternCount {
          max              : int; // default: 255
          hardware_default : int; // default: 0
        }
      }
    }
  }
}
```

Description

The PostScanActivityControl wrapper is an optional wrapper defined within the LogicBIST controller wrapper that configures the hybrid TL/LBIST post-scan activity. You can build the hardware to let the controller run continuously to keep the power from dropping after the last scan pattern is applied. This is useful when you are running many controllers in parallel, and you do not want the power to drop between the time that one controller stops and the next controller starts.

A second type of post-scan activity enables a number of post-scan patterns to be applied so that the power can gradually settle rather than drop abruptly.

Arguments

- activity_type :auto | none | cooldown | unlimited;

  An optional property that specifies the type of post-scan activity to use. When set to unlimited, the controller keeps running continuously after the specified number of patterns has completed. When set to cooldown, the CoolDownPatternCount/max property specifies how many extra patterns are run.

  When this property is set to auto, it resolves to cooldown if the CoolDownPatternCount wrapper is present; otherwise, it resolves to none.

- CoolDownPatternType/max : *int*;

  An optional property that specifies the maximum number of cool-down patterns for the controller. This is equivalent to "set_lbist_controller_options -max_cooldown_pattern_count." The default is 255 when activity_type is set to cooldown, and 0 otherwise.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- CoolDownPatternType/hardware_default : *int*;

  An optional property that specifies the maximum number of cool-down patterns for the controller. The default is 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Interface

An optional wrapper that specifies the port names on the interface of the LogicBIST controller module.

Usage

```
DftSpecification (module_name, id) {
   LogicBist {
      Controller(id) {
         Interface {
            IjtagScanInterface {
               tck              : port_name; // default: ijtag_tck
               reset            : port_name; // default: ijtag_reset
               select           : port_name; // default: ijtag_sel
               capture_en       : port_name; // default: ijtag_ce
               shift_en         : port_name; // default: ijtag_se
               update_en        : port_name; // default: ijtag_ue
               scan_in          : port_name; // default: ijtag_si
               scan_out         : port_name; // default: ijtag_so
            }
            ControllerChain {
               enable           : port_name; // default: ccm_en
               scan_in          : port_name; // default: ccm_scan_in
               scan_out         : port_name; // default: ccm_scan_out
               scan_en          : port_name; // default: ccm_scan_en
            }
            test_clock_out      : port_name;
                                    // default: lbist_test_clock_out
            edt_update          : port_name;
                             // default: [edt_update_in|edt_update_out]
            shift_clock_src     : port_name; // default: shift_clock_src
            clock_out           : port_name; // default: edt_lbist_clock
            reset_out           : port_name; // default: lbist_reset
            enable_out          : port_name; // default: lbist_en
            done                : port_name; // default: -
            prpg_en             : port_name; // default: lbist_prpg_en
            misr_en             : port_name; // default: misr_accumulate_en
            ijtag_select_out    : port_name; // default: edt_sib_en
            scan_en             : port_name; // default: scan_en[_in|_out]
            async_set_reset_en  : port_name;
                                    // default: lbist_async_set_reset_en
            self_test_en        : port_name; // default: self_test_en
            cooldown_counter    : port_name; // default: -
            warmup_counter      : port_name; // default: -
            ext_lbist_en      : port_name; // default: ext_lbist_en
         }
      }
   }
}
```

Description

The Interface wrapper is an optional wrapper defined within the LogicBIST controller wrapper that specifies the port names on the interface of the LogicBIST module. You can use this

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

wrapper to rename the listed ports. Each property has a default value as described in the configuration data syntax.

Arguments

- IjtagScanInterface/tck : *port_name* ;

  An optional property that specifies the name of the test clock IJTAG control signal. The default is ijtag_tck.

- IjtagScanInterface/reset : *port_name* ;

  An optional property that specifies the name of the reset IJTAG control signal. The default is ijtag_reset.

- IjtagScanInterface/select : *port_name* ;

  An optional property that specifies the name of the select IJTAG control signal. The default is ijtag_sel.

- IjtagScanInterface/capture_en : *port_name* ;

  An optional property that specifies the name of the enable capture IJTAG control signal. The default is ijtag_ce.

- IjtagScanInterface/shift_en : *port_name* ;

  An optional property that specifies the name of the enable shift IJTAG control signal. The default is ijtag_se.

- IjtagScanInterface/update_en : *port_name* ;

  An optional property that specifies the name of the enable update IJTAG control signal. The default is ijtag_ue.

- IjtagScanInterface/scan_in : *port_name* ;

  An optional property that specifies the name of the scan in IJTAG control signal. The default is ijtag_si.

- IjtagScanInterface/scan_out : *port_name* ;

  An optional property that specifies the name of the scan out IJTAG control signal. The default is ijtag_so.

- ControllerChain/enable : *port_name* ;

  An optional property that specifies the name of the controller chain enable port. This port is only used if DftSpecification/LogicBist/Controller/ControllerChain/present is on. The default is ccm_en.

- ControllerChain/scan_in : *port_name* ;

  An optional property that specifies the name of the controller chain scan in port. This port is only used if DftSpecification/LogicBist/Controller/ControllerChain/present is on. The default is ccm_scan_in.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ControllerChain/scan_out : *port_name* ;

  An optional property that specifies the name of the controller chain scan out port. This port is only used if DftSpecification/LogicBist/Controller/ControllerChain/present is on. The default is ccm_scan_out.

- ControllerChain/scan_en : *port_name* ;

  An optional property that specifies the name of the controller chain scan enable port. This port is only used if DftSpecification/LogicBist/Controller/ControllerChain/present is on. The default is ccm_scan_en.

- test_clock : *port_name* ;

  An optional property that specifies the name of an input clock port with which to operate the LogicBIST controller. The default is test_clock.

- test_clock_out : *port_name* ;

  An optional property that specifies the name of an output clock port that pulses during shift and capture for LogicBIST. The default is lbist_test_clock_out.

- shift_clock_src : *port_name* ;

  An optional property that specifies the name of an input clock port with which to operate the LogicBIST controller. A dash (-) value indicates that the shift clock source is not desired and is not created; in this case, only TCK and edt_clock are allowable shift clock sources. The default is shift_clock_src.

- edt_update : *port_name* ;

  An optional property that is used to create the input and output ports that are connected to the source of an intercepted EDT update signal. The tool appends the "_in" and "_out" suffixes to the specified port name to create the input and output ports. The defaults are edt_update_in and edt_update_out.

- clock_out : *port_name* ;

  An optional property that specifies the name of an output port for the generated LogicBIST clock. The default is lbist_clock.

- reset_out : *port_name* ;

  An optional property that specifies the name of an output port that indicates if the synchronous reset of LogicBIST controller occurs. The default is lbist_reset.

  The tool prepends the "ext_" prefix to the specified port name if ext_mode_edt_present is set to "on".

- enable_out : *port_name* ;

  An optional property that specifies the name of an output port that indicates if LogicBIST is enabled. The default is lbist_en.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- done : *port_name* ;

  An optional property that specifies the name of an output port that indicates when the LogicBIST controller has finished its test. The tool creates this port only if you provide a port name.

- prpg_en : *port_name* ;

  An optional property that specifies the name of an output port that indicates if the PRPG is active during the scan load/unload. The default is prpg_en.

  The tool prepends the “ext_” prefix to the specified port name if ext_mode_edt_present is set to “on”.

- misr_en : *port_name* ;

  An optional property that specifies the name of an output port that indicates if the MISR is to accumulate the scan unload data. The default is misr_accumulate_en.

  The tool prepends the “ext_” prefix to the specified port name if ext_mode_edt_present is set to “on”.

- ijtag_select_out : *port_name* ;

  An optional property that specifies the name of the IJTAG ToSelectPort that enables the EDT IJTAG network. The default is edt_sib_en.

- scan_en : *port_name* ;

  An optional property that is used to create the input and output ports that are connected to the source of an intercepted scan enable signal. The tool appends the “_in” and “_out” suffices to the specified port name to create the input and output ports. The defaults are scan_en_in and scan_en_out.

- async_set_reset_en : *port_name* ;

  An optional property that specifies the name of an output port that indicates if the asynchronous set and reset signals are pulsed during the LogicBIST pattern. The default is lbist_async_set_reset_en.

- self_test_en : *port_name* ;

  An optional property that specifies the name of the self-test enable port. This property is only used if the Controller/self_test property is on. The default port name is self_test_en.

  The tool prepends the “ext_” prefix to the specified port name if ext_mode_edt_present is set to “on”.

- cooldown_counter : *port_name* ;

  An optional property that specifies the name of the cooldown_counter port. You can use this port to monitor the progress of the cooldown pattern sequence. The tool creates this port only if you provide a port name, and if the PostScanActivityControl/activity_type property does not equal “none.”

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- warmup_counter : *port_name* ;

  An optional property that specifies the name of the warmup_counter port. You can use this port to monitor the progress of the warmup pattern sequence. The tool creates this port only if you provide a port name, and if the PostScanActivityControl/activity_type property does not equal “none.”

- ext_lbist_en : *port_name* ;

  An optional property that specifies the name of an external LBIST enable port. The default is ext_lbist_en.

Examples

The following example renames LogicBIST ports back to their pre-2018.1 release versions for purposes of backwards annotation.

```
DftSpecification(<module_name>,<id>) {
  LogicBist {
    Controller(<id>) {
      Interface {
        IjtagScanInterface {
          tck        : tck;
          reset      : sib_reset;
          select     : sib_en;
          capture_en : sib_capture_en;
          shift_en   : sib_shift_en;
          update_en  : sib_update_en;
          scan_in    : lbist_scan_in;
          scan_out   : lbist_scan_out;
        }
        ControllerChain {
          enable   : ccm_en
          scan_in  : ccm_scan_in
          scan_out : ccm_scan_out
          scan_en  : ccm_scan_en
        }
        test_clock       : edt_clock;
        clock_out        : lbist_clock;
        reset_out        : lbist_reset;
        enable_out       : lbist_en;
        prpg_en          : prpg_en;
        misr_en          : misr_en;
        ijtag_select_out : edt_sib_en;
        scan_en          : scan_en;
        shift_clock_src  : shift_clock_src;
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Connections

This wrapper adds and specifies properties for the hybrid TK/LBIST controller connections.

Usage

```
DftSpecification (module_name,id) {
  LogicBist {
    Controller {
      Connections {
        edt_clock                  : port_pin_name ;
                  // default: OptionalDftSignal(edt_clock)
        edt_update_in              : port_pin_name ;
                  // default: OptionalDftSignal(edt_update)
        shift_clock_src            : port_pin_name ;
        scan_en_in                 : port_pin_name ;
                  // default: OptionalDftSignal(scan_en)
        controller_chain_enable    : port_pin_name ;
        controller_chain_scan_in   : port_pin_name ;
        controller_chain_scan_out  : port_pin_name ;
                       // default: (none)
        ext_lbist_en : port_pin_name ;
                       // default: DftSignalOrTiedLow(ext_lbist_en)
        done                       : port_pin_name, ... ;
      }
    }
  }
}
```

Arguments

- edt_clock : *port_pin_name* ;

Note

When you use a dft signal for edt_clock, you must source it from an existing design pin rather than create it from other signals. That is, you must specify the edt_clock signals with the “add_dft_signal edt_clock -source_node” command rather than the “add_dft_signal edt_clock -create_from_other_signals” command.

An optional property used to specify a source node to connect the edt_clock port to. The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the set_design_level command, is not “chip” and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it is created as an input port during insertion.When design level is “chip”, specified port names must already be connected to a pad cell. The connection is automatically made to the internal side of the input pad buffer. You can specify OptionalDftSignal(edt_clock) or DftSignal(edt_clock) as the value, and it is interpreted to request a connection to the DFT signal edt_clock as it was added using the add_dft_signals command. When using OptionalDftsignal(edt_clock), you do not get an error if the DFT signal edt_clock was not added; a connection to the port called “edt_clock” is made instead and the port is created if not present unless design level is equal to “chip” in which case you get an error.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- edt_update_in : *port_pin_name* ;

  An optional property used to specify a source node to connect the edt_update_in port.

  The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the set_design_level command, is not "chip" and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it is created as an input port during insertion.

  When the design level is "chip," the specified port names must already be connected to a pad cell. The tool automatically makes the connection to the internal side of the input pad buffer. You can specify OptionalDftSignal(edt_update) as the value, and the specification interprets this as a request for a connection to the DFT signal edt_update at the time it is added using the add_dft_signals command. When you use OptionalDftSignal(edt_update), the tool does not report an error if the DFT signal edt_update was not added. The tool makes a connection to the port called "edt_update" instead. The tool also creates the port if it is not present, unless the design level is equal to "chip," in which case the tool reports an error.

  This property defaults to OptionalDftSignal(edt_update).

- shift_clock_src : *port_pin_name* ;

  A property that defines the shift clock source. This property is required when the shift_clock_src property in the Controller/Interface wrapper is not set; however, it may be omitted when the shift_clock_src property in the Controller/Interface wrapper is set to "-".

- scan_en_in : *port_pin_name* ;

  An optional property used to specify the source node to connect the scan enable port to. If you do not specify this, the default value is OptionalDftSignal(scan_en).

- controller_chain_enable : *port_pin_name* ;

  Note

  The controller chain connections for the enable, scan in, and scan out signals cannot be specified when you have enabled the Controller/ControllerChain/segment_per_instrument property. The tool issues an error.

  An optional property used to specify the source node to connect the CCM enable port to. It is tied to "0" by default. When the Controller/ControllerChain/segment_per_instrument property is set to "off", the controller_chain_enable property evaluates to OptionalDftSignal(controller_chain_mode).

- controller_chain_scan_in : *port_pin_name* ;

  An optional property used to specify the source node to connect the CCM scan in port to. When the Controller/ControllerChain/segment_per_instrument property is set to "off", the default value is control_chain_scan_in.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- controller_chain_scan_out : *port_pin_name* ;

  An optional property used to specify the design node to connect the CCM scan out port to. When the Controller/ControllerChain/segment_per_instrument property is set to “off”, the default value is control_chain_scan_out.

- ext_lbist_en : *port_pin_name* ;

  An optional property used to specify a source node to connect to the ext_lbist_en port. This port is created only if the extest_lbist property is set to on.

  The default connection is DftSignalOrTiedLow(ext_lbist_en), meaning that it is automatically connected to the ext_lbist_en signal if it is present. Otherwise, it is tied low.

- done : *port_pin_name ...* ;

  An optional repeatable property used to specify the design node to connect the LBIST “done” output port to. This property enables you to specify one or more destination pin or port objects.

  You can specify this property only if the “done” property in the Controller/Interface wrapper specifies a valid port name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

AsyncSetResetPatternCount

This wrapper configures the hybrid TK/LBIST asynchronous set/reset pattern count.

Usage

```
DftSpecification (module_name,id) {
  LogicBist {
    Controller(id) {
      AsyncSetResetPatternCount {
        max              : int ; // default: 0
        hardware_default : int ; // default: 0
      }
    }
  }
}
```

Description

This wrapper configures the hybrid TK/LBIST asynchronous set/reset pattern count. For complete information, see "Asynchronous Set/Reset Patterns" in the *Hybrid TK/LBIST Flow User's Manual*.

Arguments

- max : *int* ;

  This property specifies the maximum number of asynchronous set/reset patterns for the controller. This is equivalent to the set_lbist_controller_options command's -max_async_set_reset_pattern_count switch pair. The default is 0.

- hardware_default : *int* ;

  This property specifies the default maximum number of asynchronous set/reset patterns for the controller. The default is 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SetLoadUnloadTimingOptions

A wrapper used to configure the hardware default values for registers that support extra setup and hold cycles.

Usage

```
DftSpecification (module_name,id) {
  LogicBist {
    Controller(id) {
      SetLoadUnloadTimingOptions {
        max_cycles_per_signal : 2 | 4 | 8;  // default: 8
        HardwareDefaults {
          scan_en_setup_extra_cycles     : int; // default: 0
          scan_en_hold_extra_cycles      : int; // default: 0
          edt_update_setup_extra_cycles  : int; // default: 0
          edt_update_hold_extra_cycles   : int; // default: 0
          prpg_en_setup_extra_cycles     : int; // default: 0
          prpg_en_hold_extra_cycles      : int; // default: 0
          misr_en_setup_extra_cycles     : int; // default: 0
          misr_en_hold_extra_cycles      : int; // default: 0
        }
      }
    }
  }
}
```

Description

Use the SetLoadUnloadTimingOptions wrapper to specify the hardware default setup and hold values at IP generation. This wrapper controls the maximum number of cycles for the scan_en, edt_update, prpg_en, and misr_en signals. See "set_load_unload_timing_options" on page 2684 for more information.

Arguments

- max_cycles_per_signal : 2 | 4 | 8 ;

  An optional property to control the maximum number of cycles that will be allowed with the scan_en, edt_update, prpg_en, and misr_en signals. The default is 8.

- HardwareDefaults/scan_en_setup_extra_cycles : *int*;

  An optional property that specifies the default number of extra cycles from the transition of the scan_en signal to the first shift_capture_clock pulse in shift or capture. The default is 0.

- HardwareDefaults/scan_en_hold_extra_cycles : *int* ;

  An optional property that specifies the default number of extra cycles from the last shift_capture_clock pulse to the transitions of the scan_en signal. The default is 0.

- HardwareDefaults/edt_update_setup_extra_cycles : *int* ;

  An optional property that specifies the default number of extra cycles from the transition of the edt_update signal to the rising edge of the shift_capture_clock. The default is 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- HardwareDefaults/edt_update_hold_extra_cycles : *int* ;

  An optional property that specifies the default number of extra cycles from the transition of the edt_update signal to the rising edge of the shift_capture_clock. The default is 0.

- HardwareDefaults/prpg_en_setup_extra_cycles : *int* ;

  An optional property that specifies the default number of extra cycles from the transition of the prpg_en signal to the rising edge of the edt_lbist_clockshift_capture_clock. The specified value cannot be different than the misr_en_setup_extra_cycles value. The default is 0.

- HardwareDefaults/prpg_en_hold_extra_cycles : *int* ;

  An optional property that specifies the default number of extra cycles from the rising edge of the edt_lbist_clock to the transitions of the prpg_en signal. The specified value cannot be different than the misr_en_hold_extra_cycles value. The default is 0.

- HardwareDefaults/misr_en_setup_extra_cycles : *int* ;

  An optional property that specifies the default number of extra cycles from the transition of the misr_accumulate_en signal to the rising edge of the edt_lbist_clock. The specified value cannot be different than the prpg_en_setup_extra_cycles value. The default is 0.

- HardwareDefaults/misr_en_hold_extra_cycles : *int* ;

  An optional property that specifies the default number of extra cycles from the rising edge of the edt_lbist_clock to the transitions of the misr_accumulate_en signal. The specified value cannot be different than the prpg_en_hold_extra_cycles value. The default is 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

NcpIndexDecoder

Decodes the NCP index output of the hybrid TK/LBIST controller into clock sequences to be generated by the Tessent OCCs across all capture procedures.

Usage

```
DftSpecification (module_name,id) {
  LogicBist {
    NcpIndexDecoder {
      parent_instance  :  instance_name ;
      leaf_instance_name  :  name  ;
      extest_lbist : on | off | auto ;
      Interface {
        ncp_index      :  port_name ;  // default: ncp_index
        clock_sequence :  port_name ;  // default: occ%d_clock_sequence
        ext_ltest_en       : port_name ; // default: ext_ltest_en
        ext_clock_sequence : port_name ;
                        // default: ext_occ%d_clock_sequence
      }
      DesignInstance (instance_name){
      }
      Connections {
        ncp_index      :  port_pin_constant_name, ... ;
        ext_ltest_en : pin_port_name ;
                        // default: DftSignalOrTiedLow(ext_ltest_en)
        ExtestClockSequence {
          Occ(occ_spec)  : pin_port_constant_name ;
        }
      }
      Ncp(id) {
        cycle(index)   :  occ_spec, ... ; // repeatable
      }
    }
  }
}
```

Description

The LBIST NCP index decoder is a simple combinational logic block that decodes the NCP index output of the LBIST controller into clock sequences to be generated by the OCCs across all capture procedures.

See "NCP Index Decoder" in the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- parent_instance : *instance_name* ;

  Instance path name of the parent in which the NCP index decoder is instantiated. The default is the top-level of the design.

- leaf_instance_name : *name* ;

  Name of the leaf instance.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- extest_lbist : on | off | auto ;

  An optional property used to specify whether the LBIST controller includes hardware to enable reuse of the internal OCC of the core in the LBIST of its parent physical region. This property inherits its values from the LogicBist/Controller(id)/extest_lbist property if it is not specified locally, and the LBIST controller is specified in the same pass. If the NCP Index decoder is specified in a separate pass, then the default is auto. If you add the ext_lbist_en DFT signal using the add_dft_signals command, auto resolves to on; otherwise, it resolves to off.

- Interface/ncp_index : *port_name* ;

  Name of the decoder's NCP index input. The default is ncp_index.

- Interface/clock_sequence : *port_name* ;

  The default is occ%d_clock_sequence.

- Interface/ext_ltest_en : *port_name* ;

  An optional property that specifies the name of an external ltest enable port. The default is ext_ltest_en.

- Interface/ext_clock_sequence : *port_name* ;

  An optional property that specifies the name of an external clock sequence port. The default is ext_occ%d_clock_sequence.

- DesignInstance (*instance_name*)

  An optional wrapper that specifies a design instance inserted during a prior insertion pass in the current or child design levels. You can specify this wrapper more than once. Each instance is searched for OCCs and the associated TCD file are loaded so that the information needed to create the NCPID's files are available. The id of the wrapper is an instance_name for child design levels or "." for the current design.

- Connections/ext_ltest_en : *port_pin_name* ;

  An optional property used to specify a source node to connect to the ext_ltest_en port. This port is created only if the extest_lbist property is set to on.

  The default connection is DftSignalOrTiedLow(ext_ltest_en), meaning that it is automatically connected to the ext_ltest_en signal if it is present. Otherwise, it is tied low.

- Connections/ExternalClockSequence/Occ(*occ_spec*) : *port_pin_constant name* ;

  An optional repeatable property used to specify a source node to connect to the ext_clock_sequence port corresponding to a specific *occ_spec*. There can be only one Occ(*occ_spec*) property for each *occ_spec* used in the Ncp(*id*) wrappers. The connection is created only for OCCs used in the NCPs and specified in the ExtestClockSequence wrapper.

  occ_spec can be one of the following:

  - The OCC instance path name.
  - The OCC module name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The OCC core name used in the TCD.
- The OCC controller ID generated in the same pass.

Note

When the connection is not specified for the OCC handled by the NCPID, this OCC is not used in the extest LBIST. The NCPID hardware is optimized so that the ext_clock_sequence path is not created for this OCC.

- Connections/ncp_index : *port_pin_constant_name*, ... ;

  Points to the LBIST controller's NCP output. The specified pins are connected to the NCP index decoder input. When not specified, the NCP index decoder input pins are tied to "0".

- Ncp(*id*)

  Repeatable wrapper that describes a single NCP. A capture procedure could either describe activity in one clock while others are OFF, or across multiple clocks.

- Ncp(*id*)/cycle(*cycle_number*) : *occ_spec*, ... ;

  Repeatable named property that specifies the clock sequence for a cycle within an NCP.

  The *cycle_number* is an integer starting from 0, which corresponds to the first cycle of capture. The clock activity on the first cycle is parallel loaded into the LSB of the OCC shift register, the second cycle onto the next significant bit and so on. When a particular cycle number is omitted, it means there is no clock activity for that cycle. The largest cycle number that can be specified is one less than the width of the OCC control Shift Register. The output NCP contains cycles from 0 till the largest cycle number specified, including empty cycles.

  The *occ_spec* is a repeatable list of Tessent OCC specification that can include:

  - OCC instance path names.
  - OCC module names.
  - OCC core names used in TCD.
  - OCC controller IDs generated in the same pass.
  - Literal string "bscan_occ" to indicate a mini-occ inside boundary scan logic.
  - Literal string "sti_occ" to indicate sib(sti) logic.

  Ambiguity in specification (for example, a module and a top level instance with same name, or a top level instance and instance leaf name with same names) are resolved in the above order. Avoid potential ambiguity by using the full instance path names.

  The tool issues an error if you specify only one Ncp wrapper, and it does not generate the NCP index output.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

In this example, assume the design has two top level OCC instances named *m8051_gate_tessent_occ_clk1_inst* and *m8051_gate_tessent_occ_clk2_inst* of the same OCC module *m8051_gate_tessent_occ*.

```
DftSpecification {
  LogicBist {
    NcpIndexDecoder {
      Connections {
        ncp_index: m8051_lbist_i/ncp ;
      }
      Ncp(stuck) {
        cycle(0): m8051_gate_tessent_occ ;
        //Specified using module name, refers to both the OCC instances.
      }
      Ncp(clk1_double_pulse) {
        cycle(0): m8051_gate_tessent_occ_clk1_inst ;
        cycle(1): m8051_gate_tessent_occ_clk1_inst ;
      }
      Ncp(clk2_double_pulse) {
        cycle(0): m8051_gate_tessent_occ_clk2_inst ;
         //Note – cycle(1) is omitted, so no clock activity
        cycle(2): m8051_gate_tessent_occ_clk2_inst ;
      }
    }
  }
}
```

Example 2

The following example inserts OCC and the NCP index decoder at the same time. As shown below, you must specify *occ_spec* with the ID as specified for the Occ/Controller that you are generating in the same pass.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(m8051, rtl)  {
  Occ {
    Controller(1) {                           // Controller ID = 1
      clock_intercept_node: NX1g/Z;
    }
    Controller(2) {                           //Controller ID = 2
      clock_intercept_node: NX2g/Z;
    }
  }
  LogicBist {
    NcpIndexDecoder {
      Ncp(slow) {
        cycle(0): Occ(1);               // Occ/Controller generated above
        cycle(0): Occ(2);               // Occ/Controller generated above
      }
      Ncp(NX1_fast) {
        cycle(0): Occ(1);
        cycle(1): Occ(1);
      }
      Ncp(NX2_fast) {
        cycle(0): Occ(2);
        cycle(1): Occ(2);
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

LpctType3

As part of the process_dft_specification flow, the LpctType3 DftSpecification syntax defines the configuration of a low pin count test (LPCT) Type3 controller to be generated and inserted into the design.

Usage

```
DftSpecification(module_name,id) {
  LpctType3 {
    scan_enable_condition : on | off ;
    reset_condition       : on | off ;
    Interface {
    }
    Connections {
    }
    Controller {
    }
  }
}
```

Description

You use an LPCT Type3 controller when there is no scan enable at the top level of the design and the design is not using a JTAG TAP controller for test. The LPCT Type3 controller generates the scan enable signal and all EDT specific control signals and you must use it in combination with and EDT wrapper. This controller requires an OCC instrument present in the design. OCC instruments may be specified in the configuration data or added as TCD instruments with the add_core_instances command. The LPCT Type3 controller uses and drives OCCs from a mix of OCC configuration data and TCD instrument instances. It also supports third-party OCCs that are added using the LpctType3/Connections/ThirdPartyOCC configuration data wrapper.

The inserted LPCT Type3 controller generates the scan enable and all the EDT specific control signals that drive the EDT and OCC controllers. This creates a restriction for interfaces used on the EDT and OCC modules; none of the instruments driven by the LPCT controller can have an IJTAG compatible interface. An error is reported during validation if an EDT or OCC instrument that has an IJTAG compatible interface and is used in the PDS flow in combination with the LPCT.

The tool generates the RTL files for the LPCT Type3 controller inside the Tessent Shell Data Base (TSDB) when the LpctType3 wrapper is present in the configuration data and the process_dft_specification command is used. During the generation phase, the tool appends entries for the created LPCT files and modules to the synthesis dictionary inside the TSDB instrument container. This allows for the automated synthesis of the LPCT RTL files with the run_synthesis command.

The LPCT controller is typically used when the design has only 3 top level ports available for “logic test”: a free-running clock, an EDT channel input port, and an EDT channel output port By default, the data_in connection of the LPCT controller instance is shared with the EDT

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

channel input connection. If pipeline stages are used for EDT input channel connections, there is no pipeline stages in the data path of the LPCT controller.

For more information, see Type 3 LPCT Controller in the *TestKompress User's Manual.*

Arguments

- scan_enable_condition : on | off ;

  An optional argument that specifies whether the tool generates scan_en_condition logic inside LPCT controller.

- reset_condition : on | off ;

  An optional argument that specifies whether the tool generates reset_condition logic inside LPCT controller.

Examples

Example 1

This example shows a minimal set of configuration data that creates an LPCT Type3 Controller.

```
DftSpecification(CoreA,rtl1) {
  EDT {
    Controller(1) {
      ijtag_host_interface : none;
      longest_chain_range : 60, 100;
      scan_chain_count : 2;
      input_channel_count : 1;
      output_channel_count : 1;
    }
  }
  OCC {
    ijtag_host_interface : none;
    Controller(1) {
      clock_intercept_node : CLK;
    }
  }
  LpctType3 {
    Controller {
      max_capture_cycles : 5;
    }
  }
}
```

Example 2

In this example, an LPCT Type3 controller is specified and OCC core instances are added.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//  command: add_core_instances -module CoreA_rtl1_tessent_occ
//  Note: Reading TCD file './tsdbA/instruments/CoreA_rtl1_occ.instrument/
//                                CoreA_rtl1_tessent_occ.tcd' from TSDB.
//  Added core instance 'CoreA_rtl1_tessent_occ_1_inst'.
//  Added core instance 'CoreA_rtl1_tessent_occ_2_inst'.
//  Added core instance 'CoreA_rtl1_tessent_occ_3_inst'.
//  command: read_config_data -from_string {
  DftSpecification(CoreA,rtl2) +{
     EDT {
      Controller(1) {
        scan_chain_count: 100;
        longest_chain_range: 100,110;
        input_channel_count: 2;
        output_channel_count: 2;
      }
      Controller(2) {
        scan_chain_count: 100;
        longest_chain_range: 100,110;
        input_channel_count: 2;
        output_channel_count: 2;
      }
    }
    LpctType3 {
      Controller {
        max_shift_cycles: auto;
        max_capture_cycles : 5;
        load_unload_cycles : 1, 2 ;
        max_scan_patterns  : 100 ;
        max_chain_patterns : 10 ;
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Interface

A wrapper that specifies the port names for the LPCT Type 3 controller module.

Usage

```
DftSpecification(module_name,id) {
  LpctType3 {
    Interface {
      clock            : port_name ;  // default: lpct_clock
      clock_mux_select : port_name ;  // default: lpct_clock_mux_select
      capture_enable   : port_name ;  // default: lpct_capture_en
      shift_enable     : port_name ;  // default: lpct_shift_en
      shift_clock      : port_name ;  // default: lpct_shift_clock
      reset            : port_name ;  // default: lpct_reset
      scan_enable      : port_name ;  // default: lpct_scan_en
      data_in          : port_name ;  // default: lpct_data_in
      test_mode        : port_name ;  // default: lpct_test_mode
      reset_out        : port_name ;  // default: lpct_reset_out
      reset_polarity   : active_high | active_low ;
      test_end         : port_name ;  // default: lpct_test_end
      test_active      : port_name ;  // default: lpct_test_active
      reset_condition  : port_name ;  // default: lpct_reset_condition
      scan_enable_condition : port_name ;
                                      // default: lpct_scan_en_condition
    }
    Connections {
    }
    Controller {
    }
  }
}
```

Description

You can use this wrapper to specify the names of the LPCT Type 3 controller ports. Each property has a default value, as described in the configuration data syntax. You can override the default name if you provide a value to be used for the port names of the LPCT controller module.

Arguments

- clock : *port_name* ;

  Specifies the name of the input clock for the LPCT controller. The default name is lpct_clock.

- clock_mux_select : *port_name* ;

  Specifies the LPCT clock mux select port name. The default is lpct_clock_mux_select.

- capture_enable : *port_name* ;

  Specifies the LPCT capture enable port name. The default is lpct_capture_enable.

- shift_enable : *port_name* ;

  Specifies the LPCT shift clock enable port name. The default is lpct_shift_enable.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- shift_clock : *port_name* ;

  Specifies the LPCT shift clock port name. The default is lpct_shift_clock.
- reset : *port_name* ;

  Specifies the LPCT reset port name. The default is lpct_reset.
- scan_enable : *port_name* ;

  Specifies the LPCT scan enable port name. The default is lpct_scan_en.
- data_in : *port_name* ;

  Specifies the LPCT data in port name. The default is lpct_data_in.
- test_mode : *port_name* ;

  Specifies the LPCT test mode port name. The default is lpct_test_mode.
- reset_out : *port_name* ;

  Specifies the LPCT reset output port name. The default is lpct_reset_out.
- reset_polarity : <u>active_high</u> | active_low ;

  An optional switch and literal pair that specifies the activity state for the input reset signal of the generated LPCT controller. The default is active_high.
- test_end : *port_name* ;

  Specifies the LPCT test_end port name. The default is lpct_test_end.
- test_active : *port_name* ;

  Specifies the LPCT test_active port name. The default is lpct_test_active.
- reset_condition : *port_name* ;

  An optional argument that specifies the port when LpctType3/reset_condition is on. By default, the port_name is lpct_reset_condition.
- scan_enable_condition : *port_name* ;

  An optional argument that specifies the port when LpctType3/scan_enable_condition is on. By default, the port_name is lpct_scan_en_condition.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Connections

This wrapper specifies the LPCT Type3 controller connections.

Usage

```
DftSpecification(module_name,id) {
  LpctType3 {
     Interface {
     }
     Connections {
       clock          : port_pin_name ; // default: lpct_clock
       reset          : port_pin_name ; // default: lpct_reset
       scan_enable    : port_pin_name ;
       data_in        : port_pin_name ; // default: lpct_data_in
       test_mode      : port_pin_name ; // default: lpct_test_mode
       reset_out      : port_pin_name, … ;
       test_end       : port_pin_name ;
       test_active    : port_pin_name ;
       reset_condition  : port_pin_name ;      // default: 1
       scan_enable_condition : port_pin_name ; // default: 1

       ThirdPartyOCC {
       }
     }
     Controller {
    }
  }
}
```

Description

The connections wrapper for the LPCT Type3 controller identifies its connections, which you specify with port or pin names. When the LPCT Type3 controller is inserted into the chip design top-level, all of the connections with a specified value must have an existing top-level port that is connected to a PAD cell. An error is reported if this is not the case.

When the LPCT Type3 controller is inserted in a physical block or sub-block design, the specified connections are created to the specified ports or pins. If the ports do not exist, the tool creates them.

Arguments

- clock : *port_pin_name* ;

  Required argument that specifies an always-pulse clock for the LPCT. Default clock name is lpct_clock.

- reset : *port_pin_name*;

  An optional property that specifies the input reset signal connection for the LPCT controller. The reset pin can be shared with the design's functional reset. This can be either a top-level port or an internal power-on-reset pin. The default reset name is lpct_reset.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can manually specify the input and output reset connections for the LPCT controller. Use the reset property to specify the input reset connection to the controller and the reset_out property to specify the connection points that the LPCT controller resets.

- scan_enable : *port_pin_name*;

  An optional argument for LPCT controllers, that specifies the list of pins to which the LPCT output scan enable pin is connected.

- data_in : *port_pin_name*;

  Optional argument that specifies the pin or port connected to the LPCT controller data_in pin. For the LPCT controller, a test sequence and configuration is scanned in through this port. This port is usually shared with the edt_channel input. The default data in name is lpct_data_in.

- test_mode :*port_pin_name*;

  Optional argument that specifies the pin is a test mode signal used to enable test mode. This signal is not used when sequence detection is used. Default test mode name is lpct_test_mode.

- reset_out : *port_pin_name*, …;

  Specifies the output connection, or connections, of the LPCT reset signal sent to the design. The value of this property can be a bus or a list of connection points. The tool makes the specified connections from the output reset pin of the controller and connects the reset port or pin to the input reset pin of the controller.

- test_end : *port_pin_name*;

  Optional argument that specifies the pin is a test mode signal used to enable test mode. This signal is not used when sequence detection is used.

- test_active : *port_pin_name*;

  Optional argument that specifies an output signal from that LPCT controller that indicates the LPCT controller is active and controlling the scan test.

- reset_condition : *port_pin_name* ;

  Optional argument that specifies the pin or port connected to the LPCT reset_condition pin as input for reset_condition logic when LpctType3/reset_condition is on. The default name is “1”.

- scan_enable_condition : *port_pin_name* ;

  Optional argument that specifies the pin or port connected to the LPCT scan_enable_condition pin as input for scan_enable_condition logic when LpctType3/ reset_condition is on. The default name is “1”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ThirdPartyOCC

A wrapper that specifies connections for a third-party OCC. If any of the properties is specified in this wrapper indicates a third-party OCC is used in the design.

Usage

```
DftSpecification(module_name,id) {
  LpctType3 {
    Interface {
    }
    Connections {
      ThirdPartyOCC {
       clock_mux_select : port_pin_name ; // default:
                                          // lpct_clock_mux_select
       capture_enable   : port_pin_name ; // default: lpct_capture_en
       shift_enable     : port_pin_name ; // default: lpct_shift_en
       shift_clock      : port_pin_name ; // default: lpct_shift_clock
      }
    }
    Controller {
    }
  }
}
```

Description

Third-party wrappers are specified only within the LpctType3/Connections wrapper. Use this wrapper to specify properties for a third-party OCC.

Arguments

- clock_mux_select : *port_pin_name* ;// default: lpct_clock_mux_select

  An optional argument that specifies the connection to be made to the clock_mux_select output of the LPCT controller. This pin is a MUX select signal that selects between the shift clock generated by the LPCT controller and the capture clock. If clock_mux_select is high, shift clock is selected; if clock_mux_select is low, the capture clock is selected.

- capture_enable : *port_pin_name ;// default: lpct_capture_en*

  A property that specifies the connection for the capture_enable output signal of the LPCT contoller.

- shift_enable : *port_pin_name* ;// default: lpct_shift_en

  A property that specifies the connection for the shift_enable output signal of the LPCT controller.

- shift_clock : *port_pin_name* ;// default: lpct_shift_clock

  A property that specifies the connection for the shift_clock signal of the LPCT controller.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Controller

A wrapper used to specify all the LPCT Type3 specific settings.

Usage

```
DftSpecification(module_name,id) {
  LpctType3 {
    Interface {
    }
    Controller {
      max_shift_cycles      : int | auto ;
      max_capture_cycles    : int ;
      load_unload_cycles    : count1, count2 ;
      max_scan_patterns     : int ;
      max_chain_patterns    : int ;
      shift_control_type    : enable | clock | none ;
      TestModeDetect {
      }
      parent_instance       : instance ;
      leaf_instance_name    : instance ;
    }
  }
}
```

Description

The controller wrapper specifies all the LPCT Type3 IP specific settings. There can only be one controller wrapper within the LpctType3 wrapper; therefore, the tool only creates a single LPCT controller during a single process_dft_specification pass.

Arguments

- max_shift_cycles : *integer* | auto ;

  A property that specifies the maximum number of shift cycles for the LPCT controller. By default the value of this property is set to auto, which indicates that the maximum number of shift cycles is calculated automatically by the tool. The value of this property affects the size of the LPCT shift cycle counter register. If it is manually set, it needs to be large enough so the LPCT shift cycle counter can handle the longest shift length (from all EDT controllers). If is is too small, an error is reported during the validation phase.

- max_capture_cycles : *integer* ;

  A property that specifies the maximum number of capture cycles for all test patterns. This is a required property.

- load_unload_cycles : *count1*, *count2* ;

  A property that specifies the number of cycles that precede and follow the shift cycles in the load_unload procedure. The value count1 is the number of cycles that precede the shift cycles in the load_unload procedure, while the value for count2 is the number of cycles that follow the shift cycles in the load_unload procedure.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- max_scan_patterns : integer ;

  A property that specifies the maximum number of scan test patterns generated in the test application.

- max_chain_patterns : *integer ;*

  A property that specifies the maximum number of chain test patterns generated in the test application.

- shift_control_type : <u>enable</u> | clock | none ;

  A property that specifies how the LPCT controller generates the shift clock control signal. By default the value of this property is set to enable. The enable value is the only value allowed when using Tessent OCCs. When you use third-party OCCs, the value of this property can be set to any value allowed by the syntax.

- parent_instance : *instance* ;

  A property that specifies the name of the parent instance in which the LPCT is instantiated. By default, the LPCT controller is instantiated on the top level.

- leaf_instance_name : *instance* ;

  A property that specifies the name of the LPCT controller instance. By default the instance name follows this pattern:

  ${design_name}_${design_id}_tessent_${instrument_type}

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TestModeDetect

A wrapper used to configure how the LPCT enters the test mode.

Usage

```
DftSpecification(module_name,id) {
  LpctType3 {
    Interface {
    }
    Connections {
    }
    Controller {
      TestModeDetect {
        type            : enum ; // legal: signal sequence
        input_sequence  : binary_seq, cycle_count ;
      }
    }
  }
}
```

Description

The LPCT controller can enter test mode in one of two ways: by detecting a specific input sequence within a particular number of cycles after test controller reset, or by a signal that indicates test mode. These two ways of entering test mode are mutually exclusive.

Use this wrapper to specify the controller enable type: signal or sequence. By default, the tool uses the test_mode connection from the Connections wrapper to cause the controller to enter test mode. If you select the sequence detection type, you must specify the input sequence that places the LPCT controller into test mode. The test controller enters test mode by detecting a specific sequence within a particular number of cycles or by a signal that indicates test mode.

Arguments

- type : signal | sequence ;

  A property that specifies the LPCT controller enable type, sequence or signal. By default, the value is set to signal, indicating that the tool uses the test_mode connection from the Connections wrapper.

  When using a test mode signal, test mode starts after the signal is asserted. This signal needs to stay asserted for only one cycle, after which it can be de-asserted at any time. The test session does not end when this signal is de-asserted; instead it ends only after end of test as determined by the pattern counters. In this case, it is your responsibility to ensure test mode is not entered during functional operation. When this signal is a top level pin, the tool can generate the correct test_setup procedure. When this signal is an internal pin, you must modify the test setup procedure for pattern generation to ensure that the internal test mode signal is asserted as desired.

  To use sequence detection, you specify the input sequence and the number of cycles within which it should be detected during IP creation. Doing this ensures the chip does not enter

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

test mode during functional operation. To use sequence detection, must specify the input sequence property.

When the input test procedure already has an existing test_setup, the sequence detect period should also consider the additional cycles necessary for applying test_setup.

- input_sequence : *binary_seq* , *cycle_count* ;

  A property that specifies the binary sequence that triggers the LPCT test mode if it is detected within the number of specified cycles after the LPCT controller has been reset.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

MemoryBisr

Specifies the creation and the optional insertion of BISR registers.

Usage

```
DftSpecification(module_name,id) {
  MemoryBisr {
    bisr_segment_order_file      : filename ;
    memory_repair_loading_method : serial | from_read_buffer ;
    memory_library_name_list     : mem_lib_name, ... ;
    AdvancedOptions {
    }
    BisrElement(memory_instance) {
    }
    Interface {
    }
    Controller {
    }
    SecondaryHostChainInterface {
    }
  }
}
```

Description

A wrapper that specifies the creation and the optional insertion of BISR registers. The presence of the controller wrapper adds a controller in the current design. A chip is allowed to have more than one Bisr controller as long as you have a set of fuses for each controller. Refer to "Built-In Self Repair (BISR)" in the *Tessent MemoryBIST User's Manual for use with Tessent Shell* for a detailed introduction to the topic, including what your memory library needs to include in order to perform Built-In Repair Analysis, and which Interface your fuse box needs to have to be usable by the controller.

Arguments

- bisr_segment_order_file : *filename* ;

  This property is used to reference a file that contains the power domain and ordering definition of the BISR segments using the syntax shown Figure 13-72. This file is typically auto-generated when the check_design_rules command runs unless the set_dft_specification_requirements -bisr_segment_order_file command was used, in which case the specified file is not generated but validated.

  This property is typically needed unless the process_dft_specification command is run with the -no_insertion option; if this option is not used, the property is optional.

  When the check_design_rules command runs, a bisr_segment_order_file is generated when all of these conditions are met:

  - The set_dft_specification_requirements -memory_test option was specified as "on".
  - The set_dft_specification_requirements -memory_bisr_chain option was not set to off.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The set_dft_specification_requirements -bisr_segment_order_file option was not specified.
- At least one memory instance exists with a Memory library description that has at least one Memory/.../SpareElement wrapper present and the set_memory_instance_options -use_in_memory_bisr_dft_specification option was not set to "off" for that instance, or, alternatively, at least one block with an ICL file that describes one or more BISR chains is present and the ignore_for_dft_specification pin attribute is not set to "on" for at least one of the BISR scan-in pins. A BISR scan-in pin as described in ICL has a ScanInPort that is part of a ScanInterface with a tessent_bisr_chain_length Attribute set to a value greater than 0.

the bisr_segment_order_file is created using the format shown in Figure 13-72. If you load a DEF file using the read_def command, the elements in the OrderedElements wrapper are ordered so as to minimize wire lengths between them. If you load a UPF or CPF file using the read_upf or read_cpf commands, a PowerDomainGroup wrapper is created for each power domain name that has at least one element. You can also use the Instance and Pin-level attribute "bisr_power_domain_name" to define the power domains. When both the "power_domain_name" and the "bisr_power_domain_name" attributes are defined on the same instance, the latter takes precedence. The advantage of the "bisr_power_domain_name" attribute is that it can be set differently on different BISR scan-in pins to associate them to different power domain names. You can also use the attribute to create extra BISR power domains that are not defined in the UPF/CPF files.

Note

The value of "none" is a reserved string and you cannot specify it as a power domain group label. Refer to the PatternsSpecification MemoryBisr/Controller/power_domain_group_labels property description for more information.

When the check_design_rules command runs and the following conditions are met:

- The set_dft_specification_requirements -memory_test option was specified as "on"
- The set_dft_specification_requirements -memory_bisr_chain option was not set to off
- The set_dft_specification_requirements -bisr_segment_order_file option was specified to refer to an existing file

The tool validates the referred bisr_segment_order_file to make sure that it lists all existing BISR chain elements and only existing BISR chain elements. An existing BISR chain element is a memory instance with a Memory library description that has at least one Memory/.../SpareElement wrapper (see "Tessent Core Description" on page 4793) present and the set_memory_instance_options -use_in_memory_bisr_dft_specification option was not set to "off" for that instance or a BISR scan-in pin on a block having an ICL file describing the BISR chains and the Pin attribute called "ignore_for_dft_specification" is not set to "on" on that pin. A BISR scan-in pin is described in ICL and has a ScanInPort that is

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

part of a ScanInterface with the Attribute "tessent_bisr_chain_length" set to a value greater than 0.

When running the process_dft_specification command, the validation of the file pointed to by the bisr_segment_order_file property is repeated again to make sure that it lists all existing BISR chain elements and only existing BISR chain elements. If the property is not specified, an empty file is assumed and an error is generated for all missing existing BISR chain elements.

The format of the BISR segment order file is shown in Figure 13-72. When referring to an instance of a block that has more than one BISR scan segment, you must add the name of the scanin port to distinguish one chain from the other. The <block_instance_name>/<bisr_scanin_port_name> paths must point to pins of an instance with an associated ICL file where the Scanin Port is described as a ScanInPort that is part of a ScanInterface with the Attribute "tessent_bisr_chain_length" set to a value greater than 0. The extract_icl command automatically sets this attribute when generating an ICL module that contains BISR chains, The BisrModules wrapper is used when the process_dft_specification command is run with the -no_insertion option and it is desired to generate BISR registers for the memory specified by BisrModules/<modulename>. BisrModules/<modulename> must correspond to a memory core description that is loaded into memory using the read_core_descriptions command or the set_design_sources -format tcd_memory command. When running process_dft_specification with the -no_insertion option, the module names of repairable memories must be provided by the user because there is no design information that can be referenced to obtain this information.

Figure 13-72. BISR Segment Order File

```
BisrSegmentOrderSpecification {
 PowerDomainGroup(power_domain_name) { // Repeatable
  OrderedElements {
   memory_instance_name;
   block_instance_name[/bisr_scanin_port_name];
   SecondaryHostChainInterface[interface_name];

   [parent_instance_name/]leaf_instance_name : pipeline;
  }
 }
 BisrModules {
  modulename;  // Repeatable
 }
}
```

You use the SecondaryHostChainInterface[*interface_name*] declaration in Figure 13-72 in a tiling design flow, which is configured when you set the set_dft_specification_requirements -design_type option to "tile" for the current design. When you specify the SecondaryHostChainInterface[*interface_name*] property, the tool includes a SecondaryHostChainInterface wrapper in the DftSpecification, which creates the pass-through connections of the complete BISR scan interface at the boundary of the current design for the power domain group. The "%s" symbol used within the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SecondaryHostChainInterface wrapper *port_pin_name* values are automatically replaced by the *interface_name* label string. You can specify any descriptive string for *interface_name* to describe the interface, such as “top”, “left”, “right”, and so on. SecondaryHostChainInterface[*interface_name*] can only be declared once per PowerDomainGroup and *interface_name* must be unique among all PowerDomainGroup declarations.

Pipeline registers can also be inserted within each power domain group BISR chain, as shown with the “: pipeline” declaration in Figure 13-72. The pipeline instance path value is composed of the optional *parent_instance_name* and a *leaf_instance_name* of the pipeline register. A one-bit BISR register is generated and instantiated at the specified instance path. Multiple pipeline instances can be specified, as long as all instance names are unique. When specifying pipeline instances inside non-unique parent instances, it is recommended to use the same *leaf_instance_name* inside all instances of a non-unique module, to avoid the parent instances being uniquified by the tool.

- memory_repair_loading_method : serial | from_read_buffer ;

  When this property is set to its default value of “serial”, the repair information is read from the fuse box, decompressed and then sent serially to the repair registers located near the memories. The advantages are that a relatively small number of fuses and very few wires are required which simplifies layout. The disadvantage is the longer length of time required to load the repair registers. It might be necessary to reduce the number of repair registers by enabling sharing of repair logic. See the repair_sharing property, and other related properties, in the MemoryCluster wrapper of the DftSpecification.

  All memories in the design, including those inside sub-blocks and physical blocks, must use a parallel repair interface when “from_read_buffer” is specified. If one or more repairable memories in the design have a serial repair interface, an error is generated.

  When this property is set to “from_read_buffer”, a multiplexer is inserted between the BISR module and the memory repair ports. This multiplexer allows the repair information to be sourced from either the fuse box read buffer or from the BISR registers. When using the read buffer as the repair source, the repair information is driven from the fuse box read buffer and broadcast to all memory repair ports in parallel. The advantages are that the repair information is available very quickly at the memory inputs and there is no limit on the number of incremental repair sessions. The disadvantages are that the memory repair information is stored in an uncompressed form and a lot of connections are created between the BISR controller and the memories. Therefore, a lot more fuses are required and routing congestion might occur during layout. For these reasons, it might be necessary to reduce the number of repair registers by enabling sharing of repair logic. See the repair_sharing property, and other related properties, in the MemoryCluster wrapper of the DftSpecification.

  The multiplexer select signal is connected to a test data register, which is controllable during patterns generation.

  When generating the BISR controller, a new signal call “SerialRepairEnable” is present on the BISR controller if the design contains BISR chains that support fast BISR loading.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- memory_library_name_list : *mem_lib_name*, ... ;

  A property that specifies a list of memory libraries associated with repairable memories in the design. This property is only used and needed if you are running the process_dft_specification command with the -no_insertion option and the design is not elaborated in memory. Otherwise, the memory_library_name is extracted from the design for the memory instances indicated inside the file specified in bisr_segment_order_file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

MemoryBisr/AdvancedOptions

Specifies advanced BISR chain properties.

Usage

```
DftSpecification(module_name,id) {
  MemoryBisr {
    AdvancedOptions {
      Memory(pattern) {                                  // Repeatable
        DisablePortDuringShifting(Select | BistOn | BistEn | all) :
                                    auto | on | off ;  // Repeatable
      }
    }
  }
}
```

Description

The MemoryBisr/AdvancedOptions wrapper is used to specify advanced BISR chain properties. The enclosed Memory wrapper identifies the memory TCD names to which the Memory wrapper properties are to be applied. The Memory wrapper is repeatable and order dependent, in that the properties specified last overwrite previously specified properties.

Arguments

- Memory(*pattern*)

  A repeatable wrapper that specifies which memory TCD names the properties contained within the wrapper are to be applied. The specified *pattern* is a glob pattern that matches the desired memory TCD name of the memories to be included. Wildcard characters are supported, as described in "Glob and Regular Expression Pattern Matching Syntax".

- Memory/DisablePortDuringShifting(Select | BistOn | BistEn | all): auto | on | off ;

  A repeatable property that specifies the memory port function and whether it is intercepted by the BISR register during repair chain access. This property only applies to memories with a serial repair interface. The property specification is for memories that match the glob pattern specified by the enclosing Memory(*pattern*) wrapper.

  This property is used to deselect specific memories with internal repair registers during an internal repair chain access. By default, the "auto" setting resolves to "on" for memories with serial repair interfaces. Gating is added on ports with Function Select, BistEn, and BistOn to force them inactive when shifting the internal BISR register of memories with a serial repair interface. The gating introduces a delay that can become critical for high-frequency designs or when BISR registers are moved away from the memory to an always-on power domain. This gating is not necessary for all memories with a serial repair interface. Please consult the datasheet for your memory to determine if the property can be set to "off".

  The Select, BistOn, and BistEn port function can be individually specified with the DisablePortDuringShifting property, or "all" can be specified to apply the setting to all three

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

port functions. When the property is set to “on”, the port function is intercepted. When set to “off”, the port function is not intercepted during BISR repair chain access.

Examples

The following example globally excludes Select, BistOn, and BistEn port functions for all memories from the BISR register during repair chain access, then enables only the select port function for memories using the SYNC_8192X32_BISR memory TCD:

```
AdvancedOptions {
  // Global setting - applies to all memories
  Memory(*) {
      DisablePortDuringShifting(all) : off;
  }
  // Setting applies to SYNC_8192X32_BISR only
  Memory(SYNC_8192X32_BISR) {
    DisablePortDuringShifting(select) : on;
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BisrElement

Overrides the settings of the BISR registers and specifies their location.

Usage

```
DftSpecification(module_name,id) {
  MemoryBisr {
    BisrElement(memory_instance | SegmentStart(pdg_label) |
                SegmentEnd(pdg_label)) {   // repeatable
      parent_instance : relative_instance ;
      parent_instance_anchor : current_design |
                  parent_instance_of_memory | attribute_expression ;
      Pipeline(before | after) {  // Repeatable once
        parent_instance    : instance_name ;
        leaf_instance_name : instance_name ;
                             //default: %s_bisr_pipeline_inst
        so_retiming : on | off ;
      }

    }
  }
}
```

Description

A repeatable wrapper that is used to override settings of the BISR registers for repairable memory instances and specify their insertion into a particular parent instance of the design. If memory instances are matched by multiple BisrElement wrappers due to wildcards, the last BisrElement property in the DftSpecification overrides any prior wrappers.

Note

When relocating the BISR modules inside non-unique design instances, Tessent Shell attempts to preserve design uniqueness. However, if the relocation of the BISR modules causes non-unique instances to differ, Tessent Shell uniquifies the design instances in order to satisfy the BISR relocation requirements.

If the BisrElement wrapper is not specified, or if it is specified but has no properties specified within the wrapper, the BISR registers are inserted adjacent to their respective memory instances.

Arguments

- BisrElement(*memory_instance* | SegmentStart(*pdg_label*) | SegmentEnd(*pdg_label*)

  The *memory_instance* string is a pattern expression matching repairable memory instances in the design. The wildcard character “*” may be used to select multiple memory instances.

  The SegmentStart(*pdg_label*) or SegmentEnd(*pdg_label*) switches can be specified in the BisrElement wrapper to identify either the start or end of a BISR chain for the power domain group specified by *pdg_label*. The *pdg_label* string can match a specific power domain group, or the “*” wildcard can be used to match all power domain groups. The

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SegmentStart of a power domain group corresponds to the position before the first element listed in the OrderedElements wrapper within the BISR Segment Order File. The SegmentEnd corresponds to the location after the last element listed. These two switches are usually specified when the Pipeline wrapper is used within the BisrElement wrapper.

Note

If you use the "*" wildcard to match all power domain groups, enclose the entire switch in double quotes. For example, BisrElement("SegmentStart(*)") or BisrElement("SegmentEnd(*)").

- parent_instance : *relative_instance* ;

  A property that specifies a design instance relative to the instance specified with parent_instance_anchor. If not specified, the default is a null string which instantiates the BISR register in the same design hierarchy as specified by the parent_instance_anchor property.

- parent_instance_anchor : current_design | parent_instance_of_memory | *attribute_expression* ;

  This property is only valid if parent_instance is specified. The parent_instance_anchor property specifies the starting point, or anchor, for the relative path specified with the *relative_instance* value of the parent_instance property. The default value of current_design specifies the top level of the current design. Specifying parent_instance_of_memory sets the anchor one level up in the design hierarchy from the memory instance(s) matched by *memory_instance*. The *attribute_expression*, enclosed in double quotation marks, is used to find a design instance on which the attribute expression is true. The expression is evaluated beginning with the parent instance of the memory. If the expression is true, this instance is set as the parent_instance_anchor. If the expression is false, the next level parent instance is then evaluated. This continues until the expression is true or the top level of the design is reached.

- Pipeline(before | after)

  The repeatable Pipeline wrapper allows you to define a pipeline register that is specified to be placed either before or after the inserted BISR register. The Pipeline wrapper can be specified only once with each before and after ID.

- Pipeline/parent_instance : *instance_name* ;

  An optional property and string pair that specifies the parent instance where the pipeline register is placed. If you do not specify it, the register is placed in the current design.

- Pipeline/leaf_instance_name : *instance_name* ;

  An optional property and string pair that specifies the pipeline register instance name. If you do not specify it, the default instance name is %s_bisr_pipeline_inst. When you use %s within the specified *instance_name*, it is replaced by the power domain name used by the memory instance matched by the BisrElement(*memory_instance*) wrapper. This can be useful when you use wildcard characters for a BisrElement(*memory_instance*) regular expression that match many memory instances from different power domain groups.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Additionally, the %m symbol can also be specified within *instance_name* and will be replaced by the matched *memory_instance*. When you use wildcard characters for matching BisrElement memory instances and do not use the %s or %m symbols for specifying the pipeline register instance name, the tool will generate an error message if it detects that two or more pipeline registers will have the same leaf_instance_name.

- Pipeline/so_retiming : on | off ;

  An optional property and boolean pair that specifies whether the pipeline register is a negedge or posedge clocked register. The default setting of "on" specifies a negedge register.

  The presence of the negedge register reduces setup margin by half a clock period, however it simplifies the BISR hold timing closure with the addition of a half clock period of margin. You should only need a BISR pipeline register without the final negedge register in special cases where you do not meet setup timing closure due to excess delay from the transmitting BISR register to the receiving register. Additionally, when the negedge register is not present, it is mandatory to balance the BISR clock throughout the design to control skew and avoid hold time closure issues.

Examples

Example 1

The following example shows how the BISR register instances for the repairable memory instances memA and memB, originally matched by a wildcard pattern, are placed into the desired top-level design locations by subsequent BisrElement wrappers.

```
BisrElement(*) {
  parent_instance_anchor : current_design ;
  parent_instance : block_instC;
}
BisrElement(core_inst*/memA) {
  parent_instance_anchor : current_design ;
  parent_instance : block_instA;
}
BisrElement(core_inst*/memB) {
  parent_instance : block_instB;   // parent_instance_anchor defaults to
}                                  // current_design if not specified.
```

If memA and memB instances are located throughout the design hierarchy, the instances located within these blocks match the subsequent BisrElement *memory_instance* expression:

```
core_instA/memA, core_instB/memA, core_instC/memB, core_instA/memB
```

The final location for the BISR register instances are then as follows:

```
block_instA        // all matched memA BISR registers
block_instB        // all matched memB BISR registers
block_instC        // all globally matched memory BISR registers
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example shows how the BISR register instances for the repairable memory instance memA, which was originally matched by a wildcard pattern, are restored to the same design instance as the memory through a final match for that memory instance:

```
BisrElement(*) {
  parent_instance_anchor : current_design ;
  parent_instance : block_instA;
}
BisrElement(*/memA) {
  parent_instance_anchor : parent_instance_of_memory ;
  parent_instance : "";
}
```

If memA is located in the design hierarchy at:

```
top/block_instB/memA
```

The final location for the BISR register instances are then at:

```
top/block_instB            // memA BISR registers adjacent to memA
block_instA                // BISR registers for all other memories
```

Example 3

This example outlines another use of the parent_instance_of_memory option for the parent_instance_anchor property with a more elaborate parent_instance specification. Assume the design contains a memory instance memA, located in the design hierarchy at:

```
block_instA/c_inst1/d_inst/memA
```

The following BisrElement wrapper is defined to instantiate the BISR registers for memA:

```
BisrElement(*) {
  parent_instance_anchor : parent_instance_of_memory ;
  parent_instance : ../memA_bisr_container_inst;
}
```

The parent instance of memA is: block_instA/c_inst1/d_inst, which is the anchor. The parent_instance property may contain "../" sequences to define relative paths from the parent instance anchor. The single "../" sequence in this example moves up one level in hierarchy from the anchor, then appends the instance memA_bisr_container_inst. The final location for memA BISR instances is then:

```
block_instA/c_inst1/memA_bisr_container_inst
```

Example 4

The following example locates each BISR register at the first parent level that has the "power_domain_island" attribute equal to "pdgA_2". The evaluation of the expression starts

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

from the parent instance of each matched memory and works up the design hierarchy. point to attribute filtering equation syntax section for more.

```
BisrElement(*) {
  parent_instance_anchor : "power_domain_island==pdgA_2" ;
}
```

If no match is found, the BISR registers are placed at the top level of the design.

Note that any of the listed Instance built-in attributes, or any user attributes that are registered against a design instance, can be used within the *attribute_expression*. Additionally, any of the Attribute Filtering Equation Syntax relations can be implemented to create more complex expressions.

Example 5

The following example shows how to use the “SegmentStart” keyword to insert a pipeline register (without retiming) at the end of all BISR chains within the current design:

```
BisrElement("SegmentStart(*)") {
  Pipeline(after) {
    so_retiming : off;
  }
}
```

Example 6

The following example outlines a case where the “%m” symbol is needed to avoid errors due to instance name duplications. Assume the current design has three memory instances: memA, block_inst1/memB, and block_inst1/memC, where block_inst1 is not a physical region.

The bisr_segment_order file assigns the BISR registers to the following power domain groups:

```
BisrSegmentOrderSpecification {
  PowerDomainGroup(-) {
    OrderedElements { memA; }
  }
  PowerDomainGroup(pdg_B) {
    OrderedElements{
      block_inst1/memB;
      block_inst1/memC;
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The DftSpecification for this example specifies the following pipeline registers are to be added:

```
BisrElement(memA) {
  Pipeline(before) {
  }
}

BisrElement(block_inst1/*) {
  Pipeline(after) {
  }
}
```

The BisrElement(memA) wrapper creates a BISR pipeline register that is inserted before memA. The pipeline register's instance name will default to "bisr_pipeline_inst" since it is part of the always-on power domain, for which the "%s" and underscore are not used.

The BisrElement(block_inst1/*) wrapper specifies that pipeline registers are to be added after each memory's BISR register in block_inst1. Since the leaf_instance_name is not specified, the tool uses the default pipeline instance name "%s_bisr_pipeline_inst", which resolves to "pdg_B_bisr_pipeline_inst" for all pipeline registers in block_inst1. This results in an error since the instance names are not unique. You can resolve this error by specifying the leaf_instance_name property and using the "%m" variable as shown in the example below:

```
BisrElement(block_inst1/*) {
  Pipeline(after) {
    leaf_instance_name : %s_%m_bisr_pipeline_inst ;
  }
}
```

The BISR pipeline registers will now be placed after each memory's BISR register in block_inst1 with the following unique instance names:

```
pdg_B_memB_bisr_pipeline_inst
pdb_B_memC_bisr_pipeline_inst
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Interface

Specifies the names of the ports created when connecting the BISR chains to ports of a module of type sub_block or physical_block as specified using the set_design_level command.

Usage

```
DftSpecification(module_name,id) {
  MemoryBisr {
    Interface {
      scan_in             : port_naming ; // default: %s_bisr_si
      scan_out            : port_naming ; // default: %s_bisr_so
      capture_shift_clock : port_naming ; // default: %s_bisr_clk
      shift_en            : port_naming ; // default: %s_bisr_shift_en
      reset               : port_naming ; // default: %s_bisr_reset
      parallel_in         : port_naming ; // default: %s_bisr_parallel_in
      serial_repair_enable: port_naming ;
                                 // default: %s_bisr_serial_repair_enable
      memory_disable      : port_naming ; // default: %s_bisr_mem_disable
      memory_chain_select : port_naming ; // default:
                                          // %s_bisr_mem_chain_select
    }
  }
}
```

Description

A wrapper that specifies the names of the ports created when connecting the BISR chains to ports of a module of type sub_block or physical_block as specified using the set_design_level command.

The “%s” symbol is required within the port name and can be specified at the start or end of the port name string. It is replaced by the power domain name followed or preceded by an underscore when the module contains power domain names other than “-”. When the power domain name is “-”, the “%s” is replaced by an empty string and the adjacent leading or trailing underscore character is removed from the port name.

The ports are only created when the Controller wrapper is not specified, in which case the chains are connected to it. The ports are allowed to already be present. If the has_functional_source attribute on the scan_out port is true, the BISR chain is inserted between the net sourcing the scan_out port and the port itself. ICL extract checks that the chains trace back to the specified scan_in port. This feature is useful to support multi-pass insertion flows.

Arguments

- scan_in : *port_naming* ;

  A string that specifies the naming to use for the scan_in port. When unspecified, it defaults to the value specified in the MemoryBisr/ChainInterface/scan_in property inside the DefaultsSpecification/DftSpecification wrappers. See the get_defaults_value command description for more details about the company, group, and user defaults specifications.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- scan_out : *port_naming* ;

  A string that specifies the naming to use for the scan_out port. When unspecified, it defaults to the value specified in the MemoryBisr/ChainInterface/scan_out property inside the DefaultsSpecification/DftSpecification wrappers. See the get_defaults_value command description for more details about the company, group, and user defaults specifications.

- capture_shift_clock : *port_naming* ;

  A string that specifies the naming to use for the capture_shift_clock port. When unspecified, it defaults to the value specified in the MemoryBisr/ChainInterface/capture_shift_clock property inside the DefaultsSpecification/DftSpecification wrappers. See the get_defaults_value command description for more details about the company, group, and user defaults specifications.

- shift_en : *port_naming* ;

  A string that specifies the naming to use for the shift_en port. When unspecified, it defaults to the value specified in the MemoryBisr/ChainInterface/shift_en property inside the DefaultsSpecification/DftSpecification wrappers. See the get_defaults_value command description for more details about the company, group and user defaults specifications.

- reset : *port_naming* ;

  A string that specifies the naming to use for the active low reset port. When unspecified, it defaults to the value specified in the MemoryBisr/ChainInterface/reset property inside the DefaultsSpecification/DftSpecification wrappers. See the get_defaults_value command description for more details about the company, group and user defaults specifications.

- parallel_in : *port_naming* ;

  A string that specifies the naming to use for the BISR chain parallel input port. This property is only effective when DftSpecification/MemoryBisr/ memory_repair_loading_method: from_read_buffer has been specified, otherwise it is ignored. When unspecified, it defaults to the value specified in the MemoryBisr/ ChainInterface/parallel_in property inside the DefaultsSpecification/DftSpecification wrappers. See the get_defaults_value command description for more details about the company, group and user defaults specifications.

- serial_repair_enable : *port_naming* ;

  A string that specifies the naming to use for the BISR chain loading enable port. This property is only effective when DftSpecification/MemoryBisr/ memory_repair_loading_method: from_read_buffer has been specified, otherwise it is ignored. This port is connected to the “SerialRepairEnable” output port of the BISR controller. When unspecified, it defaults to the value specified in the MemoryBisr/ ChainInterface/serial_repair_enable property inside the DefaultsSpecification/ DftSpecification wrappers. See the get_defaults_value command description for more details about the company, group and user defaults specifications.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- memory_disable : *port_naming* ;

  A string that specifies the naming to use for the memory_disable port. When unspecified, it defaults to the value specified in the MemoryBisr/ChainInterface/memory_disable property inside the DefaultsSpecification/DftSpecification wrappers. See the get_defaults_value command description for more details about the company, group and user defaults specifications.

  This port is only created when at least one memory instance has a Memory wrapper (see "Tessent Core Description" on page 4793) with function "bisrSerialData" and a Memory with function "select" in its Memory library description or a child block with this port is present. Such ports are identified in the ICL module as DataInPorts with the Attribute "tessent_bisr_function" set to "memory_disable".

- memory_chain_select : *port_naming* ;

  A string that specifies the naming to use for the memory_chain_select port. When unspecified, it defaults to the value specified in the MemoryBisr/ChainInterface/memory_chain_select property inside the DefaultsSpecification/DftSpecification wrappers. See the get_defaults_value command description for more details about the company, group, and user defaults specifications.

  This port is only created when at least one memory instance has a Memory wrapper (see "Tessent Core Description" on page 4793) with function bisrSerialData in its Memory library description or a child block with this port is present. Such ports are identified in the ICL module as DataInPorts with the Attribute "tessent_bisr_function" set to "memory_chain_select".

Examples

The following example defines an interface naming convention that uses capital letters and TS as an identifier.

```
DftSpecification(my_design,rtl) {
  MemoryBisr {
    Interface {
      scan_in             : %s_TS_BISR_SI ;
      scan_out            : %s_TS_BISR_SO ;
      capture_shift_clock : %s_TS_BISR_CLK ;
      shift_en            : %s_TS_BISR_SHIFT_EN ;
      reset               : %s_TS_BISR_RESETB ;
      memory_disable      : %s_TS_BISR_MEM_DISABLE ;
      memory_chain_select : %s_TS_BISR_MEM_CHAIN_SEL ;
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SecondaryHostChainInterface

A wrapper that specifies the custom pass-through connection port/pin names of the BISR scan interface at the boundary of the current design.

Usage

```
DftSpecification(module_name,id) {
  MemoryBisr {
   SecondaryHostChainInterface {
     to_scan_in     : port_pin_name ; //default: %s_bisr_to_si
     from_scan_out : port_pin_name ; //default: %s_bisr_from_so
     to_capture_shift_clock : port_pin_name ; //default: %s_bisr_to_clk
     to_shift_en    : port_pin_name ; //default: %s_bisr_to_shift_en
     to_reset       : port_pin_name ; //default: %s_bisr_to_reset
     to_memory_disable : port_pin_name ; //default: %s_bisr_to_mem_disable
     to_memory_chain_select : port_pin_name ;
                                //default: %s_bisr_to_mem_chain_select
     Pipeline(before | after) {  // One of each always inferred
       parent_instance     : instance_name ;
       leaf_instance_name : instance_name ;
                                //default: %s_bisr_pipeline_%d_inst
       so_retiming : on | off ;
     }
   }
  }
```

Description

The SecondaryHostChainInterface wrapper defines custom port names for the pass-through connections of the complete BISR scan interface. The ports are available at the boundary of the current design and connect to the corresponding ports defined in the DftSpecification MemoryBisr/Interface wrapper. You do not need to specify the SecondaryHostChainInterface wrapper if you want to use the default port/pin names.

Use the MemoryBisr/SecondaryHostChainInterface wrapper in conjunction with the IjtagNetwork/HostScanInterface(*id*)/SecondaryHostScanInterface to implement a tiling design flow composed of tiles that have IJTAG, memory BIST, and memory BISR elements. The IJTAG element in a tile is mandatory. However, only tiles that have repairable memories incorporate the MemoryBisr/SecondaryHostChainInterface wrapper and the BISR pass-through connections. The current design is implemented in a tile configuration when you specify "set_dft_specification_requirements -design_type tile" command, and the BISR Segment Order File has a SecondaryHostChainInterface[*interface_name*] declaration for the power domain that applies to the BISR interface defined for the current design.

The "%s" symbol, specified anywhere within the port name string, is a required symbol within the port name. By default, the tool replaces the "%s" symbol with the "*interface_name*" string, which is followed or preceded by an underscore.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- to_scan_in : *port_pin_name* ;

  This property defines the output port or pin name that passes the BISR_SI connection of the current design to the BISR_SI port of the following BISR tile, or alternately, used to pass a loopback connection to the from_scan_out input port of the current design for the final BISR tile instance. The default name is %s_bisr_to_si.

- from_scan_out : *port_pin_name* ;

  This property defines the input port or pin name that receives the BISR_SO connection for the current design. The default name is %s_bisr_from_so.

- to_capture_shift_clock : *port_pin_name* ;

  This property defines the output port or pin name that passes the BISR_CLK connection of the current design to the BISR_CLK port of the following BISR tile. The default name is %s_bisr_to_clk.

- to_shift_en : *port_pin_name* ;

  This property defines the output port or pin name that passes the BISR_SHIFT_EN connection of the current design to the BISR_SHIFT_EN port of the following BISR tile. The default name is %s_bisr_to_shift_en.

- to_reset : *port_pin_name* ;

  This property defines the output port or pin name that passes the BISR_RESET connection of the current design to the BISR_RESET port of the following BISR tile. The default name is %s_bisr_to_reset.

- to_memory_disable : *port_pin_name* ;

  This property defines the output port or pin name that passes the BISR_MEM_DISABLE connection of the current design to the BISR_MEM_DISABLE port of the following BISR tile. The default name is %s_bisr_to_mem_disable.

- to_memory_chain_select : *port_pin_name* ;

  This property defines the output port or pin name that passes the BISR_MEM_CHAIN_SELECT connection of the current design to the BISR_MEM_CHAIN_SELECT port of the following BISR tile. The default name is %s_bisr_to_mem_chain_select.

- Pipeline(before | after)

  The Pipeline wrapper enables you to define a pipeline register placed either before or after the given SecondaryHostChainInterface. The Pipeline(after) wrapper configures the added pipeline inside the from_scan_out port, while the Pipeline(before) wrapper configures the added pipeline inside the to_scan_in port. For timing purposes, the tool always infers a pipeline register before and after each SecondaryHostChainInterface, even when it is not specified. You can specify Pipeline wrapper properties for each pipeline register to override default values as needed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Pipeline/parent_instance : *instance_name* ;

  An optional property and string pair that specifies the parent instance where the pipeline register is placed. If you do not specify it, the register is placed in the current design.

- Pipeline/leaf_instance_name : *instance_name* ;

  An optional property and string pair that specifies the pipeline register instance name. If you do not specify it, the default instance name is %s_bisr_pipeline_%d_inst, where %d is "out" for the Pipeline(before) wrapper and "in" for the Pipeline(after) wrapper.

- Pipeline/so_retiming : on | off ;

  An optional property and boolean pair that specifies whether the pipeline register is a negedge or posedge clocked register. The default setting of "on" specifies a negedge register.

  The presence of the negedge register reduces setup margin by half a clock period, however it simplifies the BISR hold timing closure with the addition of a half clock period of margin. You should only need a BISR pipeline register without the final negedge register in special cases where you do not meet setup timing closure due to excess delay from the transmitting BISR register to the receiving register. Additionally, when the negedge register is not present, it is mandatory to balance the BISR clock throughout the design to control skew and avoid hold time closure issues.

Examples

Example 1

This example shows you how to configure the DftSpecification to insert a BISR controller, read in a BISR segment order file, and configure the SecondaryHostChainInterface wrapper to specify custom port names for the pass-through connections in the current design block:

```
DftSpecification() {
 MemoryBisr {
    bisr_segment_order_file : tower_base.bisr_segment_order;
    Controller {
      …
    }
    SecondaryHostChainInterface {
      to_scan_in : %s_to_bisr_si;
      from_scan_out : %s_from_bisr_so;
      to_capture_shift_clock : %s_to_bisr_clk;
      to_reset : %s_to_bisr_reset;
      to_shift_en : %s_to_bisr_shift_en;
      to_memory_disable : %s_to_bisr_mem_disable;
      to_memory_chain_select : %s_to_bisr_mem_chain_select;
    }
 }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

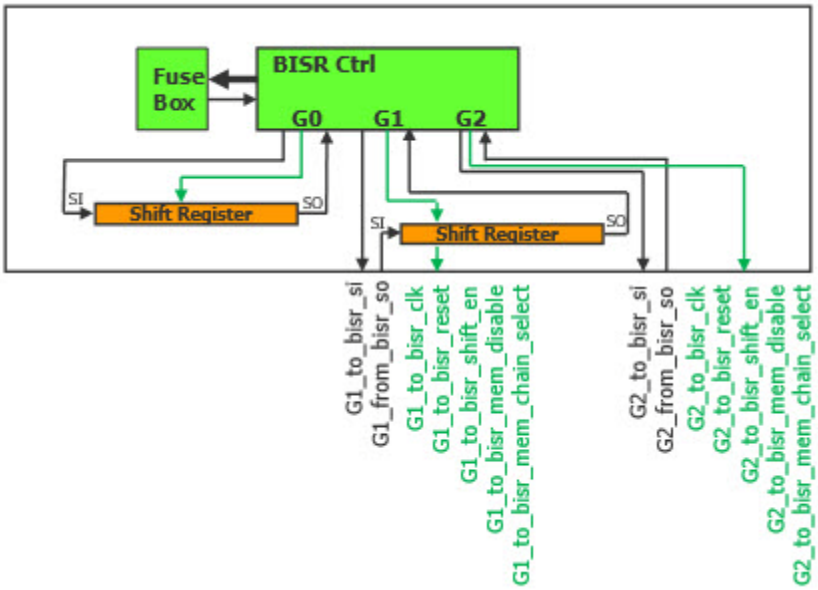
The BISR segment order file for this example is as follows:

```
BisrSegmentOrderSpecification {
  PowerDomainGroup(-) {
    OrderedElements {
      mem1;
      mem2;
    }
  }
  PowerDomainGroup(G1) {
    OrderedElements {
      mem3;
      mem4;
      SecondaryHostChainInterface(G1);
    }
  }
  PowerDomainGroup(G2) {
    OrderedElements {
      SecondaryHostChainInterface(G2);
    }
  }
}
```

The SecondaryHostChainInterface(*interface_name*) label can be any descriptive string. Common names are “pdgA_top”, “pdgB_right”, and so on. When you only have one power domain group, you can remove the power domain ID as part of the interface name (for example: “top”, “right”). In this example, it simply replicates the associated power domain group label.

The figure below shows the resulting names at the boundary of the current design for each power domain group that implements the SecondaryHostChainInterface. This example design block is used as the base, or starting tile, in a BISR tiling design flow.

Figure 13-73. Tiling BISR Ports With a BISR Controller



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

This example shows you how to configure the DftSpecification to read in a BISR segment order file and configure the SecondaryHostChainInterface wrapper that specifies custom port names for the pass-through connections in the current design block. This shows the creation of a BISR tiling block that does not have a BISR controller, and is used in conjunction with other tiling blocks to construct the desired sequence.

```
DftSpecification() {
 MemoryBisr {
    bisr_segment_order_file : tower_base.bisr_segment_order;
    SecondaryHostChainInterface {
      to_scan_in : %s_to_bisr_si;
      from_scan_out : %s_from_bisr_so;
      to_capture_shift_clock : %s_to_bisr_clk;
      to_reset : %s_to_bisr_reset;
      to_shift_en : %s_to_bisr_shift_en;
      to_memory_disable : %s_to_bisr_mem_disable;
      to_memory_chain_select : %s_to_bisr_mem_chain_select;
    }
 }
}
```

The BISR segment order file for this example is as follows:

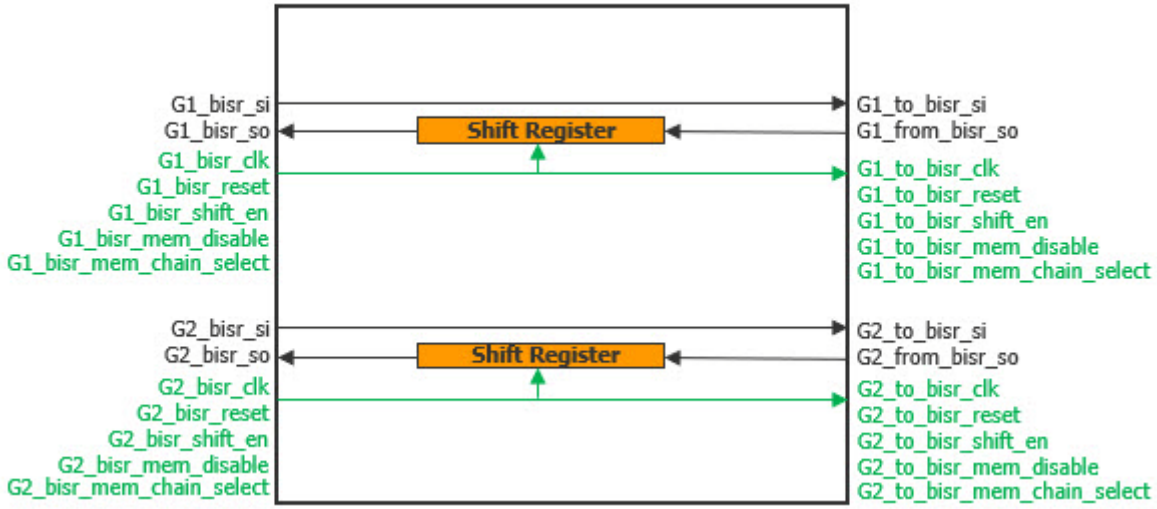
```
BisrSegmentOrderSpecification {
  PowerDomainGroup(G1) {
    OrderedElements {
      mem1;
      mem2;
      SecondaryHostChainInterface(G1);
    }
  }
  PowerDomainGroup(G2) {
    OrderedElements {
      mem3;
      mem4;
      SecondaryHostChainInterface(G2);
    }
  }
}
```

As with the previous example, the SecondaryHostChainInterface(*interface_name*) label can be any descriptive string. Common names are “top”, “base”, “up”, “right”, and so on. In this example, it simply replicates the associated power domain group label.

The figure below shows the resulting names at the boundary of the current design for each power domain group that implements the SecondaryHostChainInterface. This example design block is used as a tiling block in a BISR tiling design flow.

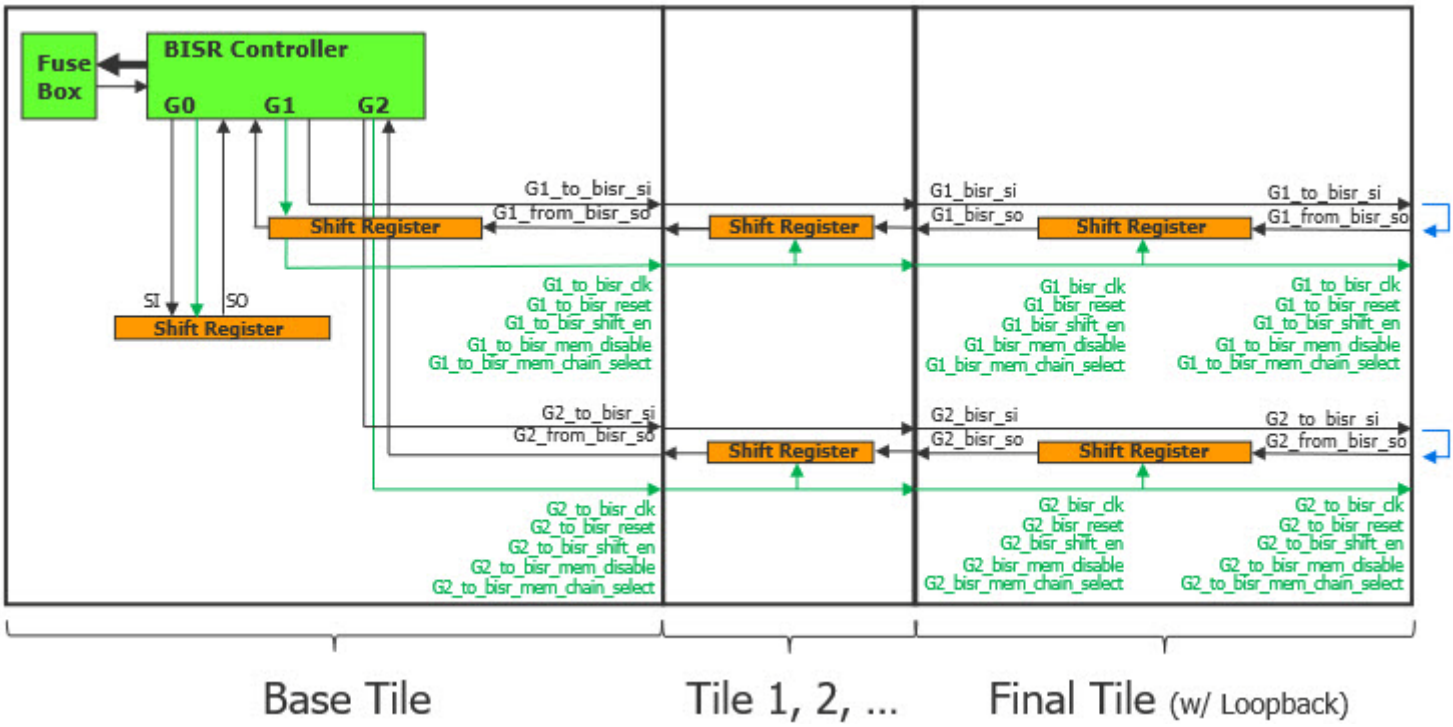
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-74. Tiling BISR Ports Without a BISR Controller



The figure below illustrates a system-level BISR tiling configuration, consisting of the base tile with the BISR controller from Example 1 and a chain of BISR tile blocks described in this example. To run simulations through a SecondaryHostChainInterface, you must add loopbacks between the final BISR_SO port and the BISR_SI port. See the add_loadboard_loopback_pairs command for pattern sets and the LoadBoardInfo wrapper for the PatternsSpecification.

Figure 13-75. System-Level BISR Tiling

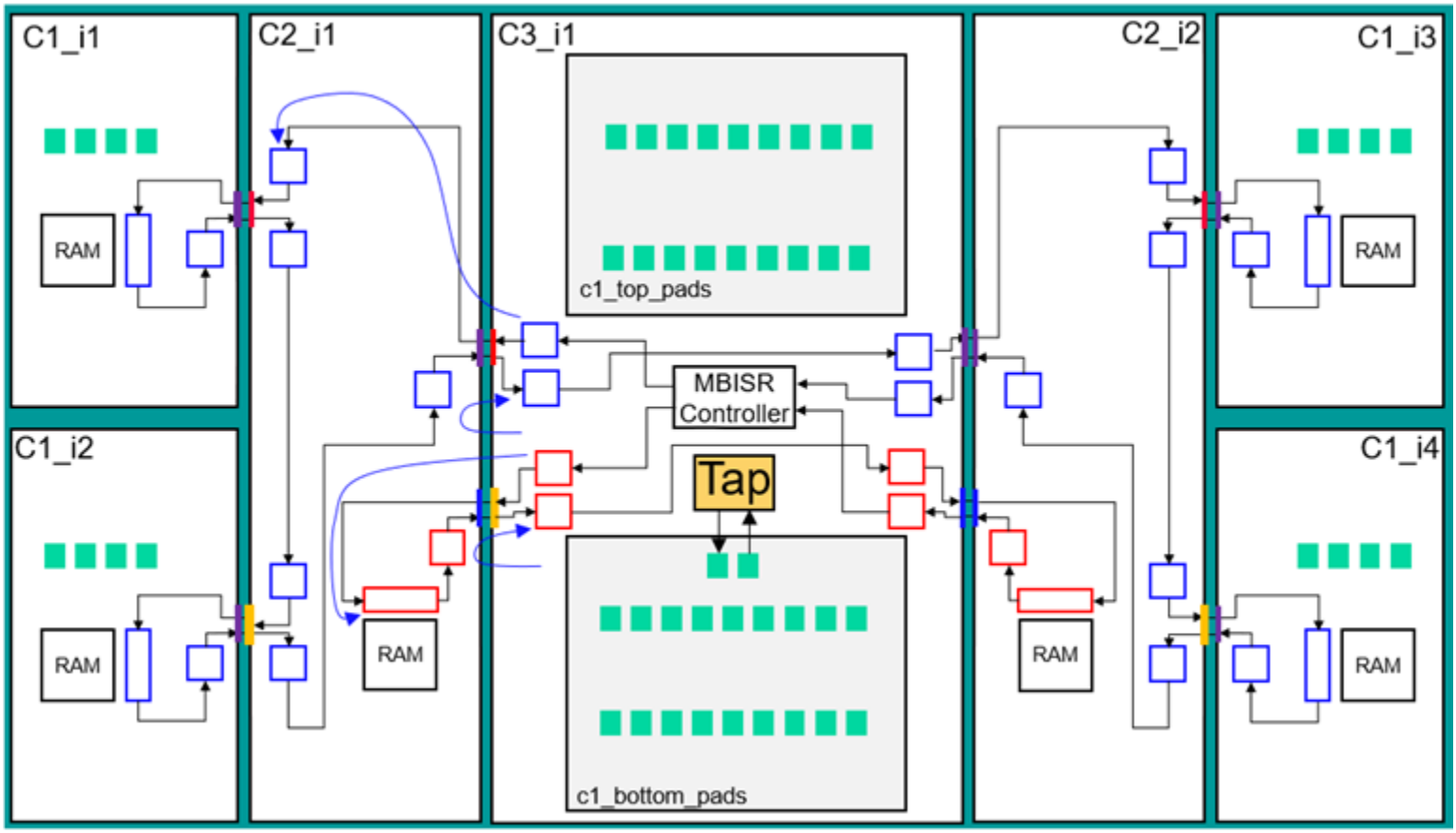


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 3

This example shows how memory BISR chains are inserted into a tile-based chip design. The example focuses on the Memory BISR part of the reference design presented in Example 4 of the OCC wrapper. In that example, the functional clocking is explained, with the localized functional clock trees and the forward- and loop-timing interfaces. The memory BISR chains are best designed to follow a clocking that is like the functional mode, where the BISR clock trees are localized to each tile and the leaf node of one tree sources the BISR clock of the adjacent tiles. Figure 13-76 shows a schematic representation of the BISR network to be used in the example tiled design.

Figure 13-76. Memory BISR Chains in a Tiled Design



The IJTAG network section is implemented in the same insertion pass as the memory BIST and memory BISR components. The IJTAG insertion is shown in Example 3 of the IjtagNetwork wrapper.

C1 Memory BISR Insertion

Below is an excerpt of the dofile to insert the memory BISR chain within the C1 module. The tool reads a **UPF** file to determine the power group ID for each memory. In this simple example, only one power domain called pdgB exists for everything with C1. The presence of the **-design_type tile** switch in the set_dft_specification_requirements command instructs the create_dft_specification command to create the MemoryBISR wrapper with the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BisrElement("SegmentEnd(*)") wrapper, which means that the tool inserts a BISR pipeline at the end of each BISR chain with no retiming element. Removing the retiming element is needed because we are using a loop timing interface in the return path. We want to give the full BISR clock period for the BISR SO data to return after the rising edge of the clock.

```
read_design c1 -design_id rtl1
set_current_design c1

read_upf ../../design/c1/c1.upf

set_dft_specification_requirements -memory_test on -design_type tile
check_design_rules

set spec [create_dft_specification]
set_config_value allow_port_creation_on_current_design -in $spec off

report_config_data ${spec}/MemoryBISR
MemoryBisr {
  bisr_segment_order_file : c1.bisr_segment_order;
  BisrElement("SegmentEnd(*)") {
    Pipeline(after) {
      so_retiming : off;
    }
  }
}

cat c1.bisr_segment_order
BisrSegmentOrderSpecification {
  PowerDomainGroup(pdgB) {
    OrderedElements {
      RAM;
    }
  }
}
```

Note

In a typical design with many repairable memories within the tile, reading in a **DEF** file with placement information of each memory ensures the **bisr_segment_order** file is created in a way to minimize the routing of the scan chain through the memories.

C2

Below is the excerpt from the dofile to insert memory BISR within the C2 tile. Just as with the C1 tile, the "set_dft_specification_requirements **-design_type tile**" command triggers the insertion of a pipelining BISR stage at the end of each segment, where the retiming element is removed to provide a full clock period for the loop timing interface. The C2 tile also needs to route the pdgB BISR chain to the two C1 tiles beyond it, as shown in Figure 13-76. These requirements are communicated by the "**-memory_bisr_host_list**" switch of the set_dft_specification_requirements command. The value of the switch is a dictionary that provides a power domain ID followed by a list of host ports needed by the adjacent tiles for the given power domain group.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The memory BISR host ports are reflected in the *.bisr_order_file* using the **SecondaryHostChainInterface(id)** elements. (See the bisr_segment_order_file property in the MemoryBisr wrapper section for more details on the *.bisr_order_file*.) The pdgB power domain group has no memories in this simple example. If you had memories and provided a **DEF** file containing the physical location of all the memories and the bisr_to_si ports, the tool creates the **.bisr_order_file** with RAMs, and the SecondaryHostChainInterface ordered in such a way to minimize routing of the scan chains through them. When you do not provide a DEF (as in this example), the SecondaryHostChainInterface is ordered following the list of elements you provide in the "**-memory_bisr_host_list**" switch value.

```
read_design c2 -design_id rtl1
set_current_design c2

read_upf ../../design/c2/c2.upf

set_dft_specification_requirements -memory_test on -design_type tile \
            -memory_bisr_host_list {pdgB {pdgB_r1 pdgB_r2}}

check_design_rules

set spec [create_dft_specification]

set_config_value allow_port_creation_on_current_design -in $spec off

report_config_data ${spec}/MemoryBISR
MemoryBisr {
  bisr_segment_order_file : c2.bisr_segment_order;
  BisrElement("SegmentEnd(*)") {
    Pipeline(after) {
      so_retiming : off;
    }
  }
}

cat c2.bisr_segment_order
BisrSegmentOrderSpecification {
  PowerDomainGroup(pdgA) {
    OrderedElements {
      RAM;
    }
  }
  PowerDomainGroup(pdgB) {
    OrderedElements {
      "SecondaryHostChainInterface(pdgB_r1)";
      "SecondaryHostChainInterface(pdgB_r2)";
    }
  }
}
```

C3

Below is the excerpt from the dofile to insert the memory BISR within the C3 tile, as shown in Figure 13-76. The C3 tile contains the memory BISR controller, which is why we use the "set_dft_specification_requirements -memory_bisr_controller on" command. By default, the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

tool only inserts a BISR controller automatically when the design level is “chip”. This example specifies the -memory_bisr_host_list switch with a dictionary containing two power domain IDs and a list of two host ports for each. The ID used for the BISR host port includes both the power domain label and a left and right suffix. The ID must be unique across all host ports as it is used as part of the default port names to replace the %s string. Refer to the Usage section to see the default values of the SecondaryHostChainInterface functions and see their %s string replaced by the specified ID of the SecondaryHostChainInterface wrapper.

```
read_design c3 -design_id rtl1
read_design c3_top_pads -design_id rtl1 -view ijtag_graybox
read_design c3_bottom_pads -design_id rtl1 -view ijtag_graybox
set_current_design c3

set_dft_specification_requirements -memory_test on \
                   -memory_bisr_controller on \
                   -design_type tile \
                   -memory_bisr_host_list {pdgA {pdgA_left pdgA_right} \
                                           pdgB {pdgB_left pdgB_right}}

check_design_rules

set spec [create_dft_specification]
set_config_value allow_port_creation_on_current_design -in $spec off

report_config_data ${spec}/MemoryBISR
MemoryBisr {
  bisr_segment_order_file : c3.bisr_segment_order;
  Controller {
    ijtag_host_interface : Sib(mbisr);
    repair_clock_connection : top_pads/bisr_clock_int;
    repair_trigger_connection : top_pads/bisr_trigger_int;
    fuse_box_location : external;
    fuse_box_interface_module : my_fuse_box_interface;
    AdvancedOptions {
      repair_word_size : 3;
      zero_counter_bits : 20;
    }
  }
}

cat c3.bisr_segment_order
BisrSegmentOrderSpecification {
  PowerDomainGroup(pdgA) {
    OrderedElements {
      "SecondaryHostChainInterface(pdgA_left)";
      "SecondaryHostChainInterface(pdgA_right)";
    }
  }
  PowerDomainGroup(pdgB) {
    OrderedElements {
      "SecondaryHostChainInterface(pdgB_left)";
      "SecondaryHostChainInterface(pdgB_right)";
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Controller

Specifies the creation and optionally the insertion of a BISR loader controller.

Usage

```
DftSpecification(module_name,id) {
  MemoryBisr {
    Controller {
      ijtag_host_interface                : node_id ;
      parent_instance                     : inst_name ;
      leaf_instance_name                  : inst_name ;
      repair_clock_connection             : pin_or_port_name ;
      repair_trigger_connection           : pin_or_port_name ;
      repair_clock_period                 : time ;
      write_duration_count_connection     : auto | pin_or_port_name ;
      repair_mode_connection : pin_or_port_name,[...] ; //3 bits required
      serial_repair_enable                : pin_or_port_name ;
                                                          // default: auto
      programming_voltage_source          : pin_or_port_name ;
      programming_voltage_port            : pin_or_port_name ;
      fuse_box_location                   : internal | external ;
      fuse_box_interface_module           : module_name ;
      max_fuse_box_programming_sessions : int | unlimited ; // default: 1
      repair_method                       : hard | soft ;
      AdvancedOptions  {
      }
      ExternalFuseBoxOptions  {
      }
      PowerDomainOptions  {
      }
    }
  }
}
```

Description

A wrapper that specifies the creation and optionally the insertion of a BISR loader controller. A controller needs a dedicated set of fuses associated with it and you typically need only one controller per chip. The architecture of the controller is described in the "The BISR Controller" section of the *Tessent MemoryBIST User's Manual*.

Arguments

- ijtag_host_interface : *node_id* ;

  A property that specifies where the BISR controller should be attached to the iJTAG network. The string node_id must have one of the following formats: HostScanInterface(id), Tap(id)/HostIjtag(id), ScanMux(id)/Input(1|0) or Sib(id). The specified node id must exist in the IjtagNetwork wrapper of the same DftSpecification wrapper.

  When using the create_dft_specification command, the BIST controller is attached to the Sib(sti) node. The Bisr controller is scan-testable and its test_en pin must be driven high during any logictest mode. The logictest modes often rely on the fact that memories were repaired prior to running the ATPG patterns when the sync write through feature is used.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The bisr_clock ports are held low when ltest_en is high to make sure the BISR controller can be scan tested without disturbing the unloaded repair solution in the BISR registers.

- parent_instance : *inst_name* ;

  A property that specifies the parent instance in which the memory BISR controller instance is to be inserted. When unspecified, it defaults to the top level of the design. When set to ".", it explicitly points to the top level of the design.You specify this option if you want the memory BISR controller to be instantiated inside a different hierarchy. The parent instance must point to a unique instance; otherwise, it is uniquified during process_dft_specification.

- leaf_instance_name : *inst_name* ;

  An optional property that specifies the memory BISR controller instance name explicitly. The default value is *design_name_design_id*_tessent_mbisr_controller_inst. This property is not used nor needed when the process_dft_specification command is run with the -no_insertion option.

- repair_clock_connection : *pin_or_port_name* ;

  A property that defines a functional clock that is going to be free-running during power-up and is to be used to unload the fuse content into the BISR registers. When you point to a top level port, the connection is automatically transferred on the from_pad port the input pad buffer when the pad cell is properly described in your cell library.

  The distributed architecture and conservative clocking methodology used for self-repair require some care in the selection of the functional repair clock that is used to apply repair during chip power-up. It is recommended that you use a functional clock of 50 MHz or less in that functional mode. Note that all other modes of operation for the BISR controller used for manufacturing test use the TAP clock (TCK), which is 10 MHz by default. Using such a low frequency simplifies timing closure. At 10 MHz, self-repair takes approximately 1 ms to execute for a BISR chain with 5000 bits, which is sufficient to repair hundreds of memories. If this time needs to be shortened, you can use a faster functional clock. Refer to the "Choosing a Functional Repair Clock" topic in the *Tessent MemoryBIST User's Manual for use With Tessent Shell* manual for further information.

- repair_trigger_connection : *pin_or_port_name* ;

  A property that specifies the functional signal that resets the BISR controller and BISR registers and initiates the memory repair. This signal is typically derived from a power-on detector or the power management unit of your device. When you point to a top level port, the connection is automatically transferred on the from_pad port of the input pad buffer when the pad cell is properly described in your cell library.

- repair_clock_period : *time* ;

  An optional property that specifies the repair clock period used to calculate the strobe counter value for the case where repair_method is set to hard, and the connection specified with repair_clock_connection cannot be traced to a clock source. This property is applicable when repair_mode_connection is specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- write_duration_count_connection : auto | *pin_or_port_name* ;

  An optional property that can be specified when the repair_method property is set to hard. When a system logic connection is specified with *pin_or_port_name*, the signal is used to drive the strobe counter, used by the fuse box interface during fuse programming when the controller is operating in direct access mode. If no value is specified, or the value auto is used, the strobe counter value is calculated using the fuse box write_duration value specified in the tcd_fbox file and the functional repair clock period. This property is not used if repair_mode_connection is not specified.

- repair_mode_connection : *pin_or_port_name*, [...] ;

  An optional property that specifies the signals used to operate the BISR controller from the system logic. The repair_mode_connection property adds the direct access ports on the Memory BISR controller and automates the connection to the system logic. The repair_mode_connection property requires a three bit source, which can be specified either as a bus or as three scalar nets.

  The BISR controller is initiated when a low value followed by a rising edge is detected on the resetN input, which is configured with the Interface/reset property. All BISR chains maintain their values if resetN remains high after the controller execution. If resetN is set low and opcode is set to 3'b00x (Functional Power-Up or BISR Chain Length Calculation modes), the BISR chain is cleared. When there are multiple power domain groups, only the BISR chains that are enabled using the PowerDomainGroupEnable input on the controller is cleared.

  Please refer to the following sections in *Tessent MemoryBIST User's Manual for use With Tessent Shell* for details on the BISR protocol:

  - BISR Protocol (Single Power Domain Group)
  - BISR Protocol (Multiple Power Domain Groups)

Table 13-12. repair_mode_connection Repair Modes

opcode[2:0]	Run Mode	Description
3'b000	Functional Power-Up	Extracts the repair data from the eFuse and loads it inside the BISR chains. This mode also calculates the BISR chain length during the chain loading process.
3'b001	BISR Chain Length Calculation	Performs an asynchronous reset of the BISR chains, followed by a serial loading of 0s and calculation of the BISR chain length.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-12. repair_mode_connection Repair Modes (cont.)

opcode[2:0]	Run Mode	Description
3'b010	Self Fuse Box Program	Compresses the content of the BISR chains and programs it into the eFuse. A BISR chain rotation is performed during this process. **Note:** The ijtag_reset signal must be inactive on the device or design block where the BISR controller is located. An active ijtag_reset signal inhibits write operations inside the eFuse during the Self Fuse Box Program operation.
3'b100	Verify Fuse Box	Performs a BISR chain rotation and compares the content against the compressed data inside the eFuse. The BISR controller GO output is high if the content matches, otherwise GO stays low.
3'b101	Rotate BISR chain (No capture)	Rotates the content of the BISR chain. Use this mode to transfer the BISR chain content into internal repair registers of memories with serial repair interfaces.
3'b110	Capture BIRA into BISR without chain rotation	Performs a capture of the redundancy analysis values into the BISR chain. This mode is typically used after running redundancy analysis, and before the Self Fuse Box Program mode. This mode is also used when performing soft repair in designs that contain memories with parallel repair interfaces.
3'b111	Capture BIRA into BISR chain with chain rotation	Performs a capture of the redundancy analysis values into the BISR chain, followed by a BISR chain rotation. This mode is also used when performing soft repair in designs that contain memories with serial repair interfaces.

- serial_repair_enable : *pin_or_port_name* ;

  A property that specifies the input that selects between serial or parallel BISR chain loading. The specified pin or port is connected to the “SerialRepairEnable” input port of the BISR controller. When driven with a “0” the BISR chains are loaded in parallel from the read buffer of the fuse box. When driven with a “1”, the BISR controller initializes the BISR chains serially. This property is required when the MemoryBisr/ memory_repair_loading_method parameter is set to “from_read_buffer”, otherwise it is ignored.

- programming_voltage_source : *pin_or_port_name* ;

  A property that defines the top-level port or internal voltage source that supplies the programming voltage for the fuse box programming. If the programming voltage is generated using an on-chip charge pump, then specify the voltage output pin of the charge pump. This property must be specified when the fuse_box_location property is set to internal. This property is used to make the connection between the specified programming

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

voltage source and the fuse box programming voltage pin. You do not need to specify this property when the fuse_box_location property is set to external because the programming voltage source should already be connected to the fuse box interface in the pre-dft inserted design. Refer to "Connecting the BISR Controller to an External Fuse Box" for further information about external fuse box implementation.

- programming_voltage_port : *port_name* ;

  An optional property that defines the top-level port that supplies the programming voltage. The voltage port must be identified if it needs to be driven to a specific programming value when the fuses are being programmed. You do not need to specify this property when the programming_voltage_source is specified and references a top-level port or when the voltage is generated from an internal charge pump.

- fuse_box_location : internal | external ;

  A property that specifies if the fuse box is to be instantiated inside the BISR controller, or if it already exists in the design and connections should be made to and from it. It is typically external when the up address range of the fuse box is used by other functional or test modes. Refer to the ExternalFuseBoxOptions wrapper if your fuse box is external to the controller.

  When not specified, this property defaults to the MemoryBisr/fuse_box_location property found inside the DefaultsSpecification/DftSpecification wrapper.

- fuse_box_interface_module : *module_name* ;

  A property that specifies the module name of the fuse box interface to instantiate in the design. The interface module name specified with this property overrides any that are inferred from tcd_fusebox libraries loaded in the design. If the fuse box module is already instantiated into the design, its associated FuseBoxInterface wrapper is automatically searched and loaded. See the set_module_matching_options and the "set_design_sources -format tcd_fusebox" command descriptions for more information about the searching mechanism.

  When the fuse_box_location property is set to internal and only one tcd_fusebox file is present, the fuse_box_interface_module property is inferred from the tcd_fusebox if the fuse_box_interface_module property is not specified. If multiple tcd_fusebox library files are present, then it is required to specify the fuse_box_interface_module property in this wrapper.

  When the fuse_box_location property is set to external and fuse_box_interface_module is not specified, the fuse_box_interface_module is inferred from the specified ExternalFuseBoxOptions/design_instance property. If neither the design instance or fuse_box_interface_module properties are specified, and only a single tcd_fusebox exists in the design, the fuse_box_interface_module is inferred from this tcd_fusebox. If multiple tcd_fusebox files are present, an error results and you are required to specify which fuse_box_interface_module to use.

  Refer to the FuseBoxInterface wrapper description and "Connecting the BISR Controller to an External Fuse Box" topic in the *Tessent MemoryBIST User's Manual* for further information on tcd_fusebox usage and DftSpecification requirements.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- max_fuse_box_programming_sessions : *int* | unlimited ;

  A property that enables hard incremental repair and specifies how many fuse box programming sessions are to be supported by the controller. More than one programming session is useful if you need to test the memories at different times (such as wafer, packaging and burn-in) and process, voltage or temperature conditions, and if you want to be able to program the repair fuses after each repair analysis. Refer to the "Incremental Repair" topic in the *Tessent MemoryBIST User's Manual* for a detailed description of this feature.

  When specifying an integer value, the repair information is compressed before being stored in the fuse box. The number of fuses required to store the repair information is proportional to the integer value.

  When specified as unlimited, the fuse box compression hardware is disabled inside the BISR controller. This enables the repair solution from the BISR chains to be stored uncompressed inside the fuse box. Subsequent repair solutions, from different power domains for example, can be written directly inside the fuse box without affecting the repair solution from previous repair sessions. Therefore, the repair solution for each power domain can be individually programmed into the fusebox rather than all at once. This method enables an unlimited number of self_fuse_box_program autonomous mode programming sessions. Disabling the compression hardware is also useful in the rare case where more than 50 percent of the repair registers are expected to be used. The compression algorithm becomes inefficient in that case and might require more fuses than there are bits in the BISR chain.

  Note

  When fuse box compression is disabled through the "unlimited" setting, the fuse box size must be greater than or equal to the total number of bits in the BISR chains of all power domain groups, plus one for each BISR chain.

  When not specified, this property defaults to the MemoryBisr/ max_fuse_box_programming_sessions property found inside the DefaultsSpecification/ DftSpecification wrapper.

- repair_method : hard | soft ;

  An optional property that defines the way repair data is applied to repairable memories. When not specified, the property defaults to hard, which is the standard approach employing the instantiation of a fuse box to store the repair data on-chip.

  When the soft method is selected, no fuse box is inserted and a minimal fuse box interface consisting of a single flip-flop is generated that allows applying the repair data to repairable memories. Since the repair data is not stored on-chip, the repair solution always needs to be determined upon device power-up by running memory BIST. BIRA to BISR transfer and BISR rotation operations need to be subsequently executed in order to apply the repair data to the memories.

  For the soft repair_method selection, the following properties must not be specified in the Controller wrapper: fuse_box_location, fuse_box_interface_module,

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

max_fuse_box_programming_sessions, programming_voltage_port,and programming_voltage_source. Additionally, the AdvancedOptions/FuseBoxOptions and ExternalFuseBoxOptions wrappers are not allowed when soft repair_method is selected.

Examples

The following example shows the specification of the controller wrapper. The ijtag_host_interface property is linked to Sib(bisr) node. The repair_clock_connection property points to a port of the device. The cell library information is automatically used to map the connection on the other side of the input pad buffer. The repair_trigger_connection property is used to specify the connection to the power management unit. Because the fuse box is internal, FuseBoxInterface(my_fuse_box_internal) wrapper (see "Tessent Core Description" on page 4793) is required to have been pre-loaded into memory using the read_core_descriptions command.

```
DftSpecification(my_design,rtl) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Sib(sti) {
        Sib(bisr) {
        }
      }
    }
  }
  MemoryBisr {
    Controller {
      ijtag_host_interface              : Sid(bisr) ;
      repair_clock_connection           : Clk10Mhz ;
      repair_trigger_connection         : pmu/repair_en ;
      programming_voltage_source        : VPP ;
      fuse_box_location                 : internal ;
      fuse_box_interface_module         : my_fuse_box_interface ;
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

AdvancedOptions

Specifies advanced options for the BISR controller.

Usage

```
DftSpecification(module_name,id) {
  MemoryBisr {
    Controller {
      AdvancedOptions {
        FuseBoxOptions {
          write_duration               : time | 8us ;
          read_duration                : time | 200ns ;
          init_duration                : time | 1us ;
          read_word_size               : int  | auto ;
          number_of_fuses_for_repair   : int  | auto ;
          align_access_en_with_address : on | off | auto ;
          programming_method           : buffered | unbuffered | auto ;
        }
        repair_word_size               : int | auto ;
        max_bisr_chain_length          : int | auto ;
        zero_counter_bits              : int | auto ;
        bisr_done_connection           : pin_name ;
        bisr_pass_connection           : pin_name ;
        power_up_chain_select          : internal | external ;
      }
    }
  }
}
```

Description

A wrapper that specifies advanced options for the BISR controller. You do not need to specify any of the properties with an auto default value when you have a FuseBoxInterface wrapper (see "Tessent Core Description" on page 4793) associated with the specified fuse_box_module_name property. The two connection properties are typically used to connect the status signal to the power management unit when different portions of the chip are powered up and down during the normal operation of a device, and those portions have repairable memories that need to be repaired each time the region is turned back on. Refer to the *Tessent MemoryBIST User's Manual* for detailed information about power domains and their impact on memory BISR.

Arguments

- FuseBoxOptions/write_duration : *time* | 8us ;

  A property that specifies the time it takes to program a fuse. When unspecified, the value found inside the associated FuseBoxInterface wrapper is used. If no associated FuseBoxInterface wrapper exists, the default value of "8us" is used. The value of the write duration can be found in the fuse box datasheet.

- FuseBoxOptions/read_duration : *time* | 200ns ;

  A property that specifies the time it takes to read a fuse. When unspecified, the value found inside the associated FuseBoxInterface wrapper is used. If no associated FuseBoxInterface

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

wrapper exists, the default value of 200ns is used. The value of the read duration can be found in the fuse box datasheet.

- FuseBoxOptions/init_duration : *time* | 1us ;

  A property that specifies the fuse box initialization time. A delay of the specified duration is inserted before the first read access to the fuse box after it has been selected. When unspecified, the value found inside the associated FuseBoxInterface wrapper is used. If no associated FuseBoxInterface wrapper exists, the default value of 1us is used.

- FuseBoxOptions/read_word_size : *int* | auto ;

  A property that specifies how many fuses are read in parallel. The fuse box interface module is designed to not reread the fuse box if the next read address it is getting is part of the same word that it just read. This information is needed to properly compute how long it takes to read the fuse values and estimate the worst case repair time needed when all fuses need to be read.

  When set to auto, the value found inside the associated FuseBoxInterface wrapper is used. If no associated FuseBoxInterface wrapper exists, the default value of 1 is used. It is okay to specify a read_word_size value that is actually smaller than the true word size of the fuse box that was read. This is because it only makes the calculated worst case run time for the power up emulation mode longer. This property does not affect the controller hardware and the true worst case time it takes to power up a domain. However, it is important to not specify a read_word_size that is larger than reality as the power up emulation pattern would be generated with a wait loop that is too short.

- FuseBoxOptions/number_of_fuses_for_repair : *int* | auto ;

  A property that defines the number of fuses available for repair in the fuse box. You need to reserve the bottom part of the address space for repair but you may use the upper part for other purposes. When the value is auto, the number of fuses for repair is set to the value of number_of_fuses specified in the Core/FuseBoxInterface wrapper. If no FuseBoxInterface description is available, the value is computed as 2 to the power of the number of address ports specified with the ExternalFuseBoxOptions/ConnectionOverrides/address property.

- FuseBoxOptions/align_access_en_with_address : on | off | auto ;

  A property that specifies if the pulse triggering a fuse box access and generated by the BISR controller is expected to align with the address being accessed, or if it is one cycle ahead. The value on means that the fuse box interface can only accept a pulse that aligns with the address. The tool forces this property to on when you enable hard incremental repair by setting the max_fuse_box_programming_sessions property to a value greater than 1. The value off means that the fuse box interface can also accept a pulse occurring one cycle ahead of the address allowing a slightly faster fuse box access. For example, a read access can take as little as two clock cycles instead of three for some fuse box interfaces. When set to auto, the tool uses the value found inside the associated FuseBoxInterface wrapper. If no associated FuseBoxInterface wrapper exists, it defaults to on if fuse_box_location is external, and to off if fuse_box_location is internal. Refer to the Fuse Box Protocol topic in *Tessent MemoryBIST User's Manual For Use With Tessent Shell* for further information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- FuseBoxOptions/programming_method : buffered | unbuffered | auto ;

  A property that specifies that it is not allowed to program individual fuses at randomly specified addresses. In some fuseboxes, individual fuse bits cannot be addressed directly; all fuse bits must be read or programmed as a group. When this property is set to buffered, a signal called programFB is generated by the BISR controller and connected to the fuse box interface. This signal is used to initiate final fuse box programming. Extra steps are also added to the patterns to transfer and program the fuses during the "self_fuse_box_program" autonomous operation and in the "program" and "read" fuse box access modes. When set to auto, the value found inside the associated FuseBoxInterface wrapper is used. If no associated FuseBoxInterface wrapper exists, it defaults to "unbuffered".

- repair_word_size : *int* | auto ;

  A property that specifies the repair word size which is the number of bits read from the fuse box as soon as one of the bits in the word is a one. Refer to the compression algorithm description in the *Tessent MemoryBIST User's Manual* for more information.

  When set to auto, this property is automatically set to match the largest repair word size of all your repairable memories providing the most effective encoding capacity.

- max_bisr_chain_length : *int* | auto ;

  A property that specifies the maximum length a BISR chain could have. Unless you are creating a BISR controller without having all BISR chains visible, you do not need to specify this property because the default auto value implies that the controller is built to be capable of loading BISR chains that are as much as four times larger than they currently are, allowing for any number of last minute additions.

- zero_counter_bits : *int* | auto ;

  A property that specifies how many bits the zero counter of the encoder must have. This property should never need to be specified as it is automatically set to the log2 of the specified or inferred max_bisr_chain_length property. Refer to the compression algorithm description in the *Tessent MemoryBIST User's Manual* for more information about the zero counter.

- bisr_done_connection : *pin_name* ;

  A property that specifies a connection to be made from the done status port of the BISR controller to the power management unit. This done status port indicates that all enabled power domains are programmed. See the done_to_pmu_connection property inside the PowerDomainOptions wrapper to know how to connect each power domain-specific done status port to the power management unit.

- bisr_pass_connection : *pin_name* ;

  A property that specifies a connection to be made from the pass status port of the BISR controller to the power management unit. This pass status port indicates that no errors happened while reading the fuse and loading the enabled BISR chains.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- power_up_chain_select : internal | external ;

  This property changes the default BISR chain select output on the BISR controller when performing a power-up or a calculate_bisr_chain_length pattern. This property is used for designs that contain memories with serial repair interfaces that do not perform a complete reset of their internal BISR chain during an asynchronous reset from the BISR controller. Certain TSMC repairable memories implement such a BISR reset mechanism.

  When power_up_chain_select is set to internal, the BISR controller selects the internal BISR chains when performing the chain length calculation. However, if the memory's internal BISR chain does not reset completely, this causes the BISR chain length calculation to fail. It is recommended to use the "internal" chain when possible in order to verify the internal chains every time the chip is powered up.

  When set to external, the external BISR register is used to calculate the BISR chain length during power up. This register is guaranteed to reset completely and ensures that the BISR chain length calculation succeeds.

Examples

The following example connects the done and pass status ports to pins of the power management unit.

```
DftSpecification(my_design,rtl) {
  MemoryBisr {
    Controller {
      AdvancedOptions {
        bisr_done_connection         : pmu/global_bisr_done ;
        bisr_pass_connection         : pmu/global_bisr_pass ;
      }
    }
  }

}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ExternalFuseBoxOptions

Specifies options relative to an external fuse box and its interface.

Usage

```
DftSpecification(module_name,id) {
  MemoryBisr {
    Controller {
      ExternalFuseBoxOptions {
        design_instance          : inst_name ;
        multiplexing             : on | off | auto ;
        ConnectionOverrides {
          // Inputs
          bisr_en                : pin_name,... ;
          clock                  : pin_name ;
          select                 : pin_name ;
          reset                  : pin_name ;
          access_en              : pin_name ;
          write_en               : pin_name ;
          write_duration_count   : pin_name ;


          read_buffer_select     : pin_name ;
          write_buffer_transfer  : pin_name ;
          address                : pin_name ;
          // Outputs
          done                   : pin_name ;
          read_data              : pin_name ;
          read_buffer_output     : pin_name ;
        }
      }
    }
  }
}
```

Description

A wrapper that specifies options relative to the external fuse box and its interface. This wrapper is only used when the fuse_box_location property in the DftSpecification/MemoryBisr/Controller wrapper is set to external. The ConnectionOverrides wrapper is used to define connections between the external fuse box and BISR controller. These connections can also be specified in the tcd_fusebox FuseBoxInterface module library. When defining connections in the ConnectionOverrides wrapper, it is mandatory to specify the following properties:

- done
- read_data
- write_en
- select
- access_en
- address

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- write_duration_count

Refer to the fuse_box_location and fuse_box_interface_module property descriptions, the FuseBoxInterface wrapper description and the "Connecting the BISR Controller to an External Fuse Box" topic in the *Tessent MemoryBIST User's Manual* for further details.

Arguments

- design_instance : *inst_name* ;

  A property that is only specified when the DftSpecification/MemoryBisr/Controller fuse_box_interface_module property is not specified. The fuse_box_interface_module is inferred from the design_instance property. If neither the design instance or fuse_box_interface_module properties are specified, and only a single tcd_fusebox exists in the design, the fuse_box_interface_module is inferred from this tcd_fusebox. If multiple tcd_fusebox files are present, an error results and the user is required to specify which fuse_box_interface_module to use.

- multiplexing : on | off | auto ;

  A property that specifies if the connection to the external fuse box should be multiplexed or not. The connection must be multiplexed if it is also used by some other circuitry. If it is multiplexed, a 2-to-1 multiplexer local to the connection end point is inserted and the select is driven by the bisr_en signal. When the property is set to auto, the multiplexer is only inserted if the has_functional_source Pin attribute of the specified pin_name in the connection is set to true. See "Pin" on page 3710.

- ConnectionOverrides/bisr_en : *pin_name* ;

  A property that specifies one or many connections to create from the bisr_en port to any pins in the circuit. This signal is typically used as a select of multiplexers when the fuse box is shared with other circuitry and the multiplexer is already inside the interface.

- ConnectionOverrides/clock : *pin_name* ;

  A property that specifies where to connect the clock signal for the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper associated with a fuse_box_module_interface property, or if the clock connection needs to be made to any place other than the ports of the fuse box interface module. For a description of the clock signal on the fuse box interface, refer to the FuseBoxInterface wrapper description.

- ConnectionOverrides/select : *pin_name* ;

  A property that specifies where to connect the select signal for the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper associated with a fuse_box_module_interface property, or if the select connection needs to be made to any place other than the ports of the fuse box interface module. For a description of the select signal on the fuse box interface, refer to the FuseBoxInterface wrapper description.

- ConnectionOverrides/reset : *pin_name* ;

  A property that specifies where to connect the reset signal for the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper associated with a

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

fuse_box_module_interface property, or if the reset connection needs to be made to any place other than the ports of the fuse box interface module. For a description of the reset signal on the fuse box interface, refer to the FuseBoxInterface wrapper description.

- ConnectionOverrides/access_en : *pin_name* ;

  A property that specifies where to connect the access_en signal for the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper associated with a fuse_box_module_interface property, or if the access_enable connection needs to be made to any place other than the ports of the fuse box interface module. For a description of the access_enable signal on the fuse box interface, refer to the FuseBoxInterface wrapper description.

- ConnectionOverrides/write_en : *pin_name* ;

  A property that specifies where to connect the write_en signal for the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper associated with a fuse_box_module_interface property, or if the write_enable connection needs to be made to any place other than the ports of the fuse box interface module. For a description of the write_enable signal on the fuse box interface, refer to the FuseBoxInterface wrapper description.

- ConnectionOverrides/write_duration_count : *pin_name* ;

  A property that specifies where to connect the write_duration_count signal for the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper associated with a fuse_box_module_interface property, or if the write_duration_count connection needs to be made to any place other than the ports of the fuse box interface module. For a description of the write_duration_count signals on the fuse box interface, refer to the FuseBoxInterface wrapper description.

  If the write_duration_count pins are a series of numbered scalar ports, you can use the %<integer>d[msb:lsb] symbol to specify them. For example, wcount%2d[31:0] connects to the scalar ports called wcount31 to wcount00.

- ConnectionOverrides/read_buffer_select : *pin_name* ;

  A property that specifies where to connect the read_buffer_select signal for the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper associated with a fuse_box_module_interface property, or if the read_buffer_select connection needs to be made to any place other than the ports of the fuse box interface module. When MemoryBisr/memory_repair_loading_method is set to the default setting “serial”, the control of the read_buffer_select input on the fuse box interface is accomplished by the user through assigning a TDR register inside the IjtagNetwork, then manually controlling it in the patterns specification. For further information of the read_buffer_select signal on the fuse box interface, refer to the FuseBoxInterface/Interface wrapper description.

- ConnectionOverrides/read_buffer_output : *pin_name* ;

  A property that specifies where to connect the read_buffer_output signals for the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

associated with a fuse_box_module_interface property, or if the read_buffer_output connections need to be made to any place other than the ports of the fuse box interface module. For a description of the read_buffer_output signal on the fuse box interface, refer to the FuseBoxInterface wrapper description.

If the read_buffer_output pins are a series of numbered scalar ports, you can use the %<integer>d[msb:lsb] symbol to specify them. For example RB%2d[12:0] connects to the scalar ports called RB12 to RB00.

- ConnectionOverrides/write_buffer_transfer : *pin_name* ;

  A property that specifies where to connect the write_buffer_transfer signal for the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper associated with a fuse_box_module_interface property, or if the write_buffer_enable connection needs to be made to any place other than the ports of the fuse box interface module. For a description of the write_buffer_transfer signal on the fuse box interface, refer to the FuseBoxInterface wrapper description.

- ConnectionOverrides/address : *pin_name* ;

  A property that specifies where to connect the address signals for the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper associated with a fuse_box_module_interface property, or if the address connection needs to be made to any place other than the port of the fuse box interface module. For a description of the address signals on the fuse box interface, refer to the FuseBoxInterface wrapper description.

  If the address pins are a series of numbered scalar ports, you can use the %<integer>d[msb:lsb] symbol to specify them. For example add%2d[13:0] connects to the scalar ports called add13 to add00.

- ConnectionOverrides/done : *pin_name* ;

  A property that specifies where to connect the done signal from the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper associated with a fuse_box_module_interface property, or if the done connection needs to be made from any other place than the ports of the fuse box interface module. For a description of the done signal on the fuse box interface, refer to the FuseBoxInterface wrapper description.

- ConnectionOverrides/read_data : *pin_name* ;

  A property that specifies where to connect the read_data signal for the fuse box interface. This property is used when you do not have a FuseBoxInterface wrapper associated with a fuse_box_module_interface property, or if the read_value connection needs to be made from any place other than the port of the fuse box interface module. For a description of the read_value signal on the fuse box interface, refer to the FuseBoxInterface wrapper description.

Examples

The following examples uses the design_instance property to specify which instance of the fuse box module is to be used by the BISR controller. This is only needed in the rare case that the design contains more than one instance of the module specified by the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

fuse_box_interface_module property. The clock connection is overridden and connects to the input of a clock buffer instead of to the pin of the fuse box interface module.

```
DftSpecification(my_design,rtl) {
  MemoryBisr {
    Controller {
      ExternalFuseBoxOptions {
        design_instance  : fuse_box_inst1 ;
        ConnectionOverrides {
          clock          : clock_buf/A ;
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

PowerDomainOptions

Specifies options specific to the power domains.

Usage

```
DftSpecification(module_name,id) {
  MemoryBisr {
    Controller {
      PowerDomainOptions {
        power_domain_priority_order : power_domain_name, ... ;
        PowerDomainName(power_domain_name) {
          enable_from_pmu_connection : pin_or_net_name ;
          busy_to_pmu_connection     : pin_or_net_name ;
          done_to_pmu_connection     : pin_or_net_name ;
          ChainConnectionOverride {
            to_scan_in          : port_or_pin_name ;
            from_scan_out       : port_or_pin_name ;
            capture_shift_clock : port_or_pin_name ;
            shift_en            : port_or_pin_name ;
            reset               : port_or_pin_name ;
            memory_disable      : port_or_pin_name ;
            memory_chain_select : port_or_pin_name ;
          }
        }
      }
    }
  }
}
```

Description

A wrapper that specifies options specific to the power domains. The power_domain_priority_order property is used to specify in which order the power domains are prioritized during the power up sequence.

You should order the list such that the power domains that must be programmed first are listed first. You use the PowerDomainName wrapper to specify connections between the controller and the power management unit. The power_domain_name used to identify the PowerDomainName wrappers must match the value of the bisr_power_domain_name attribute associated with memory and BISR segment instances. For more details about the usage of the bisr_power_domain_name attribute, see the description of the MemoryBisr wrapper.

Arguments

- power_domain_priority_order : *power_domain_name*,... ;

  A property that lists the power domain names in the order they must be programmed during power up. The power domains that must be powered-up faster than the others should be specified at the beginning of the list. If you are not using the -no_insertion mode of the process_dft_specification command and a memory and BISR segment instance have a bisr_power_domain_name attribute value that is not found in the list, an error is generated. An error is also generated if the list includes a power_domain_name that is not associated with any memory or BISR segment instances, unless the ChainConnectionOverride wrapper

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is specified in the corresponding PowerDomainName wrapper. Refer to "Example 2" on page 4307 for an example usage of the ChainConnectionOverride wrapper.

- PowerDomainName/enable_from_pmu_connection : *pin_or_net_name* ;

  A property that specifies a connection from the power management unit to the power domain-specific enable signal. This property is required when the design contains more than one power domain group. Refer to "Connecting a BISR Controller to System Logic" in *Tessent MemoryBIST User's Manual for use With Tessent Shell* for more information about the signals between the power management unit and the BISR controller.

- PowerDomainName/busy_to_pmu_connection : *pin_or_net_name* ;

  A property that specifies a connection to the power management unit from the power domain-specific BISR busy signal. Refer to "Connecting a BISR Controller to System Logic" in *Tessent MemoryBIST User's Manual for use With Tessent Shell* for more information about the signals between the power management unit and the BISR controller.

- PowerDomainName/done_to_pmu_connection : *pin_or_net_name* ;

  A property that specifies a connection to the power management unit from the power domain-specific BISR done signal. Refer to "Connecting a BISR Controller to System Logic" in *Tessent MemoryBIST User's Manual for use With Tessent Shell* for more information about the signals between the power management unit and the BISR controller.

- PowerDomainName/ChainConnectionOverride/to_scan_in : *port_or_pin_name* ;

  A property that overrides the scan_in connection for the chain associated with the power domain name. You use this property when you are creating and instantiating the BISR controller in a module in which the chains are either pre-connected or do not exist at all. Refer to "Example 2" on page 4307 for an example. If you override any connections for a given power domain name, you must override all of them. When using the ChainConnectionOverride wrapper to connect the BISR controller to top-level ports, there cannot be a mix of pre-connected BISR chains and top-level connections at the same time.

- PowerDomainName/ChainConnectionOverride/from_scan_out : *port_or_pin_name* ;

  A property that overrides the scan_out connection for the chain associated with the power domain name. You use this property when you are creating and instantiating the BISR controller in a module in which the chains are either pre-connected or do not exist at all. Refer to "Example 2" on page 4307 for an example. If you override any connections for a given power domain name, you must override all of them. When using the ChainConnectionOverride wrapper to connect the BISR controller to top-level ports, there cannot be a mix of pre-connected BISR chains and top-level connections at the same time.

- PowerDomainName/ChainConnectionOverride/capture_shift_clock : *port_or_pin_name* ;

  A property that overrides the capture_shift_clock connection for the chain associated with the power domain name. You use this property when you are creating and instantiating the BISR controller in a module in which the chains are either pre-connected or do not exist at all. Refer to "Example 2" on page 4307 for an example. If you override any connections for a given power domain name, you must override all of them. When using the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ChainConnectionOverride wrapper to connect the BISR controller to top-level ports, there cannot be a mix of pre-connected BISR chains and top-level connections at the same time.

- PowerDomainName/ChainConnectionOverride/shift_en : *port_or_pin_name* ;

  A property that overrides the shift_en connection for the chain associated with the power domain name. You use this property when you are creating and instantiating the BISR controller in a module in which the chains are either pre-connected or do not exist at all. Refer to "Example 2" on page 4307 for an example. If you override any connections for a given power domain name, you must override all of them. When using the ChainConnectionOverride wrapper to connect the BISR controller to top-level ports, there cannot be a mix of pre-connected BISR chains and top-level connections at the same time.

- PowerDomainName/ChainConnectionOverride/reset : *port_or_pin_name* ;

  A property that overrides the reset connection for the chain associated with the power domain name. You use this property when you are creating and instantiating the BISR controller in a module in which the chains are either pre-connected or do not exist at all. Refer to "Example 2" on page 4307 for an example. If you override any connections for a given power domain name, you must override all of them. When using the ChainConnectionOverride wrapper to connect the BISR controller to top-level ports, there cannot be a mix of pre-connected BISR chains and top-level connections at the same time.

- PowerDomainName/ChainConnectionOverride/memory_disable : *port_or_pin_name* ;

  A property that overrides the memory_disable connection for the chain associated with the power domain name. You use this property when you are creating and instantiating the BISR controller in a module in which the chains are either pre-connected or do not exist at all. Refer to "Example 2" on page 4307 for an example. If you override any connections for a given power domain name, you must override all of them. When using the ChainConnectionOverride wrapper to connect the BISR controller to top-level ports, there cannot be a mix of pre-connected BISR chains and top-level connections at the same time. The specification of the memory_disable connection overrides remains optional because this is an optional connection that is only used if your memories have serial repair access as described in the "Built-In Self Repair (BISR)" topic in *Tessent MemoryBIST User's Manual for use With Tessent Shell*.

- PowerDomainName/ChainConnectionOverride/memory_chain_select : *port_or_pin_name* ;

  A property that overrides the memory chain select connection for the chain associated with the power domain name. You use this property when you are creating and instantiating the BISR controller in a module in which the chains are either pre-connected or do not exist at all. Refer to "Example 2" on page 4307 for an example. If you override any connections for a given power domain name, you must override all of them.When using the ChainConnectionOverride wrapper to connect the BISR controller to top-level ports, there cannot be a mix of pre-connected BISR chains and top-level connections at the same time.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example defines the priority order for the power domain names A, B, and C where power domain A has highest priority and C has the lowest priority. The connections per power domain are also specified.

```
DftSpecification(my_design,rtl) {
  MemoryBisr {
    Controller {
      PowerDomainOptions {
        power_domain_priority_order : A,B,C ;
        PowerDomainName(A) {
          enable_from_pmu_connection : pmu/bisr_en_A ;
          busy_to_pmu_connection     : pmu/bisr_busy_A ;
          done_to_pmu_connection     : pmu/bisr_done_A ;
        }
        PowerDomainName(B) {
          enable_from_pmu_connection : pmu/bisr_en_B ;
          busy_to_pmu_connection     : pmu/bisr_busy_B ;
          done_to_pmu_connection     : pmu/bisr_done_B ;
        }
        PowerDomainName(C) {
          enable_from_pmu_connection : pmu/bisr_en_C ;
          busy_to_pmu_connection     : pmu/bisr_busy_C ;
          done_to_pmu_connection     : pmu/bisr_done_C ;
        }
      }
    }
  }
}
```

Example 2

The following example defines explicit connections for the chains associated with the power domain names. This syntax is used when constructing and inserting a BISR controller into a design in which the chains are not visible. In this example, the BISR controller is inserted into a module that is used in many designs. The module already contains the power management unit and an instance of the fuse box interface module called fuse_2k module. The list of power domain names and the max chain length and repair word size values are specified because their values cannot be extracted from the chains because they do not exist yet.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(my_design,rtl) {
  MemoryBisr {
    Controller {
      ijtag_host_interface            : Sib(bisr) ;
      repair_clock_connection         : bisr_clk ;
      repair_trigger_connection       : pmu/bisr_trigger ;
      fuse_box_location               : external ;
      fuse_box_interface_module       : fuse_2k ;
       {
        repair_word_size              : 8 ;
        max_bisr_chain_length         : 4000 ;
        bisr_done_connection          : pmu/bisr_done ;
        bisr_pass_connection          : pmu/bisr_pass ;
      }
      PowerDomainOptions {
        power_domain_priority_order : A,B,C ;
        PowerDomainName(A) {
          enable_from_pmu_connection : pmu/bisr_en_A ;
          busy_to_pmu_connection     : pmu/bisr_busy_A ;
          done_to_pmu_connection     : pmu/bisr_done_A ;
          ChainConnectionOverride {
            to_scan_in                : A_bisr_si ;
            from_scan_in              : A_bisr_so ;
            capture_shift_clock       : A_bisr_clk ;
            shift_en                  : A_bisr_se ;
            reset                     : A_bisr_rst ;
            memory_disable            : A_bisr_md ;
            memory_chain_select       : A_bisr_csel ;
          }
        }
        PowerDomainName(B) {
          enable_from_pmu_connection : pmu/bisr_en_B ;
          busy_to_pmu_connection     : pmu/bisr_busy_B ;
          done_to_pmu_connection     : pmu/bisr_done_B ;
          ChainConnectionOverride {
            to_scan_in                : B_bisr_si ;
            from_scan_in              : B_bisr_so ;
            capture_shift_clock       : B_bisr_clk ;
            shift_en                  : B_bisr_se ;
            reset                     : B_bisr_rst ;
            memory_disable            : B_bisr_md ;
            memory_chain_select       : B_bisr_csel ;
          }
        }
        PowerDomainName(C) {
          enable_from_pmu_connection : pmu/bisr_en_C ;
          busy_to_pmu_connection     : pmu/bisr_busy_C ;
          done_to_pmu_connection     : pmu/bisr_done_C ;
          ChainConnectionOverride {
            to_scan_in                : C_bisr_si ;
            from_scan_in              : C_bisr_so ;
            capture_shift_clock       : C_bisr_clk ;
            shift_en                  : C_bisr_se ;
            reset                     : C_bisr_rst ;
            memory_disable            : C_bisr_md ;
          }
        }
      }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
    }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

MemoryBist

Specifies the creation and the optional insertion of memory BIST controllers in a design.

Usage

```
DftSpecification(module_name,id) {
  MemoryBist {
    ijtag_host_interface  : node_id ;
    Controller(id)
    }
    ReusedController(id) {
    }
    BistAccessPort {
    }
    TCKInjectionPoints {
    }
  }
}
```

Description

A wrapper that specifies the creation and the optional insertion of memory BIST controllers in a design.

The presence of the MemoryBist wrapper within the DftSpecification adds one or more controllers to the current design. Refer to the *Tessent MemoryBIST User's Manual* to get started with the MemoryBist insertion flow.

Arguments

- ijtag_host_interface : *node_id* ;

  A required property that specifies the ScanInterface that is hosting the memory BIST controllers. The *node_id* is of the form "Sib(id)" when the DftSpecification is created using the create_dft_specification command. However, the *node_id* is allowed to point to any valid host scan interface: Tap/HostIjtag, Sib(id), or HostScanInterface(id).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Controller

A wrapper that defines a new memory BIST controller. The id value uniquely identifies the controller. The id value is reflected in the module names associated with that controller.

Usage

```
DftSpecification(module_name,id) {
  MemoryBist {
    Controller(id) {
      parent_instance              : inst_path ;
      leaf_instance_name           : inst_name ;
      clock_domain_label           : label ;
      clock_period                 : time ;
      AdvancedOptions {               // *DefSpec
      }
      AlgorithmResourceOptions {      // *DefSpec
      }
      RepairOptions {                 // *DefSpec
      }
      DiagnosisOptions {              // *DefSpec
      }
      Step {                          // Mutually exclusive with
      }                               // MemoryCluster wrapper
      MemoryCluster(cluster_id) {     // Mutually exclusive with
      }                               // Step wrapper
      DirectAccessOptions {           // *DefSpec
      }
    }
  }
}
```

Description

A wrapper that defines a new memory BIST controller. The *id* value uniquely identifies the controller. The *id* value is reflected in the module names associated with that controller.

Arguments

- parent_instance : *inst_path* ;

  An optional property that specifies where the memory BIST controller is instantiated in the design. When set to “.”, it points to the top level of the design. When unspecified, it defaults to a null string that is interpreted as the common instance ancestor of all memories tested by the controller. See the description of the get_common_parent_instance command to see how the common ancestor is computed.

  This property is not used nor needed when the process_dft_specification command is run with the -no_insertion option.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- leaf_instance_name : *inst_name* ;

  An optional property that specifies the memory BIST controller instance name explicitly. By default, an instance name is automatically generated by appending the "_inst" string to the controller module name, which itself defaults to the following:

  `design_name_design_id_tessent_mbist_controller_id_controller`

  This property is not used nor needed when the process_dft_specification command is run with the -no_insertion option.

- clock_domain_label : *label* ;

  A property that specifies the label (see add_clocks) of the clock domain used by the memory BIST controller and all the associated memories. Any memory tested by the controller that has a clock port sourced by a different clock domain has a multiplexer added in its interface module to inject the clock used by the controller when running memory BIST.

  This property is not used nor needed when the process_dft_specification command is run with the -no_insertion option and the design is not present. When running in -no_insertion mode without the design loaded in memory, specify the clock_period property instead. This is needed to correctly create the synthesis script.

  During MemoryBIST insertion, compatible memories including multi-port memories, can be assigned to the same memory controller. From a clocking perspective, memories are compatible when:

  - All memory clock pins are connected to the same clock label.

  OR

  - Memory clock pins are connected to different clock labels, but are assigned to the same synchronous clock group and have the same frequency.

  Compatible memories can be assigned to the same controller as long as all partitioning rules are met. If memories are assigned to a different controller, it can be for reasons other than their clock source.

- clock_period : *time* ;

  A property that is only used when process_dft_specification is run with the -no_insertion option and the design with the memory instances is not loaded into the tool. In this case, the clock_domain_label property cannot be specified and instead the clock_period property is used. When the design is present, the clock period used in the synthesis script is extracted from the period of the clock referred to by the clock_domain_label property. In -no_insertion mode, there may be no design present with defined clocks to refer to. In this case, you use the clock_period property to specify at which rate the given controller is meant to operate.

  Also note that when the clock_domain_label property is not specified, the information needed to control the insertion of the clock multiplexers inside the memory interface modules is missing. You are therefore responsible for adding those multiplexers where

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

needed when you insert the controller inside your design such that all memory clocks are driven by the clock used by the controller during memory BIST.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

AdvancedOptions

Specifies advanced options for the memory BIST controller.

Usage

```
DftSpecification(module_name,id) {
  MemoryBist {
    Controller(id) {
      AdvancedOptions {
        algorithm                          : algo_name ;    // *DefSpec
                                             // default: from_library
        operation_set                      : opset_name ;
                                             // default: from_library
        extra_algorithms                   : algo_name, ... ;
        extra_operation_sets               : opset_name, ... ;
        incremental_test_mode              : on | off ;     // *DefSpec
        pipeline_controller_outputs        : on | off ;     // *DefSpec
        observation_xor_size               : off | 1..MaxPosInt |
                                             all ;          // *DefSpec
        selective_parallel_memory_test     : on | off ;     // *DefSpec
        shared_comparators_per_go_id       : int | all ;
                                             //default: 1      *DefSpec
        use_multicycle_paths               : on | off ;     // *DefSpec
        min_misr_segment_bits              : 24 | 256 ;     // *DefSpec
        rom_signature_location             : external_straps |
               min_address | max_address | off ;            // *DefSpec
      }
    }
  }
}
```

Description

A wrapper that specifies advanced options for the memory BIST controller.

Arguments

- algorithm : *algo_name* ;

  An optional property that specifies the name of an algorithm, This value serves as the default value for the Step/algorithm property when is it not specified in the Step wrapper. When unspecified, it defaults to “from_library”. See the Step/algorithm property description to understand the meaning of the default value.

  The *algo_name* value may be a user-defined algorithm or one chosen from the list of pre-defined algorithms. Refer to the chapter “Using Tessent User-Defined Algorithms” in the *Tessent MemoryBIST User’s Manual* for information on implementing your own algorithm. The pre-defined algorithms are listed in the “MemoryBIST Algorithms” appendix of the *Tessent MemoryBIST User’s Manual*.

- operation_set : *opset_name* ;

  An optional property that specifies the name of an operation set. This value serves as the default value for the Step/operation_set property when is it not specified in the Step

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

wrapper. When unspecified, it defaults to "from_library". See the Step/operation_set property description to understand the meaning of the default value.

The *opset_name* value may be a user-defined operation set or chosen from the list of pre-defined operation sets. Refer to the "Custom Operation Sets" section in the *Tessent MemoryBIST User's Manual* for information on implementing your own operation set. The pre-defined operation sets are listed in the "Tessent MemoryBIST Library Operation Sets" section of the *Tessent MemoryBIST User's Manual*.

- extra_algorithms: *algo_name*, ... ;

  An optional property that specifies the names of additional algorithms that are to be built into the controller such that they can later be selected at run time. The algorithms can be explicitly listed or defined using a regular expression.

  The *algo_name* value may be a user-defined algorithm or one chosen from the list of pre-defined algorithms. Refer to the chapter "Using Tessent User-Defined Algorithms" in the *Tessent MemoryBIST User's Manual* for information on implementing your own algorithm. The pre-defined algorithms are listed in the "MemoryBIST Algorithms" appendix of the *Tessent MemoryBIST User's Manual*.

- extra_operation_sets: *opset_name*, ... ;

  An optional property that specifies the names of additional operation sets that can be built into the controller such that they can later be selected at run time. The operation set names can be explicitly listed or defined using a regular expression.

  The *opset_name* value may be a user-defined operation set or chosen from the list of pre-defined operation sets. Refer to the "Custom Operation Sets" section in the *Tessent MemoryBIST User's Manual* for information on implementing your own operation set. The pre-defined operation sets are listed in the "Tessent MemoryBIST Library Operation Sets" section of the *Tessent MemoryBIST User's Manual*.

- incremental_test_mode : on | <u>off</u> ;

  An optional property that specifies that the necessary setup chain muxing and controller ports are generated to allow functional system debug. Provides read and write access to any BISTable memory with a negligible impact on area. For more information, see the "Functional Debug Memory Access" Appendix in *Tessent MemoryBIST User's Manual*.

- pipeline_controller_outputs : on | <u>off</u> ;

  A property that specifies the insertion of one pipeline stage on all BIST outputs from the memory BIST controller. The pipeline stage is added for all controller steps and memories under test. The pipeline flip-flops are instantiated in the controller module. This property facilitates timing closure on paths from the controller.

- observation_xor_size : off | <u>1..MaxPosInt</u> | all;

  A property that specifies the number of signals combined in an XOR tree for observing the address and control ports of the memories in the memory interface during logic test. The specified reduction ratio applies to all memory interfaces associated with the current controller. The minimum value is 1, which assigns one flop per signal. The maximum value

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is the sum of the memory address and control port widths. The all value combines all the signals into one observation register. The off value disables insertion of observation logic for logic test. If not specified, a default value of 3 is used.

- selective_parallel_memory_test: on | off ;

  A property that specifies whether or not specific memories are to be enabled during memory BIST. The compare status of any disabled memories remain at the pass status throughout the test and do not contribute to the global pass/fail status. In addition, disabled memory inputs with port function Select and WriteEnable are forced to their inactive value. The memories to be enabled for memory BIST are selected at runtime using the enable_memory_list property in the PatternsSpecification/Patterns/TestStep/MemoryBist/Controller/AdvancedOptions wrapper.

- shared_comparators_per_go_id : *int* | all ;

  A property that specifies the number of comparators to share a given GO_ID register located in the controller. You can use this feature to manage the area overhead by reducing the number of registers used to log the comparator status. However, using a value other than 1 complicates and lengthens diagnosis with bit level resolution as you need to repeat the test as many times as there are comparators sharing a common GO_ID register. For this diagnosis, you must modify the PatternsSpecification and create separate patterns for each possible integer value with the MemoryInterface/ property to manually iterate through the comparators. No automation is present for this process and you need to follow a similar manual diagnosis flow when using Tessent SiliconInsight Desktop.

  The value should be set to 1 to enable reading complete memory words in a single read operation. Also, the MemoryBist/Controller/AlgorithmResourceOptions/data_register_bits property should be set to an appropriate value based on the complexity of the patterns to be written to memory. For complete control of the write data, this property should be set to the number of bits of the widest memory tested by the controller.

  For memories that implement IO/Column repair analysis or Row and IO/Column repair analysis, the number of comparators per GO_ID must be 1. The number is automatically adjusted whenever required and a warning is issued.

  Specifying the value “all” infers that there is one GO_ID register for all comparators located in the controller.

- use_multicycle_paths : on | off ;

  A property that specifies whether or not multi-cycle paths are declared in the generated SDC and STA scripts. Omitting those greatly simplifies the SDC constraints but limits the speed at which the controller can operate as all timing paths needs to be closed as if they were all single-cycle paths when the timing exceptions are not declared.

- min_misr_segment_bits : 24 | 256 ;

  A property that specifies the number of bits contained within each segment that composes the MISR. When the default setting of 24 is specified, the MISR is composed of 24-bit and 32-bit segments. The tool selects the combination of segments, based on ROM word size, to

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

minimize the total length of the MISR. A setting of 256 only uses 256-bit segments for the MISR and reduce aliasing.

- rom_signature_location : external_straps | min_address | max_address | off ;

  A property that indicates where the expected signature that is used to validate the ROM test is stored. The selected scheme applies to all ROMs tested by the same controller. The test algorithm reads from the ROM and accumulates the ROM content into a MISR signature. When the test algorithm completes, the computed signature in the MISR is compared with an expected ROM signature to determine the memory pass/fail status. The ROM signature can be hard coded at IP generation or read from a designated ROM address. At runtime, the MISR can be shifted out and its value compared with a user-defined value specified at pattern generation.

  The default value of external_straps hard codes the ROM signature derived from the ROM content provided at IP generation. The hard-coded signature determines the memory test status. If the ROM content changes, the hard-coded signature becomes invalid. In that case, the MISR signature must be extracted at runtime and compared with the new ROM signature.

  The max_address and min_address values reserve the ROMs maximum or minimum address, as defined in the memory library, for storing the ROM signature. The designated address is unavailable for functional purposes. The length of the expected signature is typically longer than the ROM word size. Only the LSB portion of the ROM signature can be stored and compared. The memory test status is determined by reading the designated ROM address and comparing the read value with the MISR signature. When max_address is specified, all ROMs tested concurrently in the same controller step must have identical dimensions. Often, the ROM content is unknown at design time, or is updated after DFT insertion, or even in the system. This scheme facilitates the revision of the ROM content in that the new ROM signature can be reprogrammed together with the final ROM content without modifying the test circuit or the test pattern. For more information on the procedure, refer to the include_rom_signature_address property in the PatternsSpecification.

  The off value indicates the ROM signature is not stored on-chip. The memory test status is determined by extracting the computed MISR signature and comparing it with a known good value. For more information on the procedure, refer to the expected_rom_signature property in the PatternsSpecification.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

AlgorithmResourceOptions

Specifies properties related to memory BIST controller hardware to support custom algorithm implementation.

Usage

```
DftSpecification(module_name,id) {
  MemoryBist {
    Controller(id) {
      AlgorithmResourceOptions { // *DefSpec
        soft_instruction_count                : int ; // default: 0
        data_register_bits                    : int | auto ;
        counter_a_bits                        : int | auto ;
        delay_counter_bits                    : int | auto ;
        preserve_microcode_initial_values     : on | off ;
        soft_algorithm_address_min_max        : on | off ;
        max_data_inversion_address_bit_index  : int | max_index ;
                                                  // default: 0
        a_equals_b_command_allowed            : on | off ;
        address_segment_x0_y0_allowed         : on | off ;
        max_x0_segment_bits                   : 1 | auto ;
        max_y0_segment_bits                   : 1 | auto ;
      }
    }
  }
}
```

Description

A wrapper that specifies the amount of hardware to include into the memory BIST controller to support custom algorithm implementation. The more you add, the more flexible your algorithms can be but the more hardware it takes to implement the controller.

This entire wrapper can be specified in the DefaultsSpecification/DftSpecification/MemoryBist wrapper such that it gets created as you want when you run the create_dft_specification command.

Arguments

- soft_instruction_count : *int*;

  Specifies the number of instructions you want available for programming your soft programmable algorithms. The default value is 0, which means that only algorithms hardened into the controller are usable.

  The soft_instruction_count property specifies the number of instruction words that are implemented in the microprogram memory of the soft programmable controller. The number of bits per instruction word depends on many factors such as the logical address map, number of operations, and the number of instructions.

  The soft programmable controller supports the downloading of programmable algorithms at run time. You can load any soft algorithm at run time using the apply_algorithm property inside the MemoryBist/Controller/AdvancedOptions wrapper of the PatternsSpecification.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The number of instructions in the soft algorithm must not exceed the value specified for this property.

Specifying the soft_instruction_count property is a trade-off between the amount of instructions your custom soft programmed algorithm can have and area. If a large number of instructions is specified, there is a high cost in hardware to implement the microprogram memory. However, if the number of instructions is small, the types of algorithms that can be implemented is significantly reduced.

- data_register_bits :*int* | auto ;

  The data_register_bits property allows you to specify the number of bits in the write and expect data registers. The number of bits in the write and expect data registers can represent the entire width of the memory data bus or a segment of the memory data bus. If the data register size is wider than the memory data bus width, the data pattern is truncated to fit the bus width. If the data register size is narrower than the memory data bus width, the data pattern is replicated to fill the bus width. The default value of auto means 2 bits if the hard-coded algorithms are limited to SMarch, SMarchCHKB, SMarchCHKBci, SMarchCHKBcil, and SMarchCHKBvcd. If other algorithms are built into the controller, the default value is the number of bits on the widest memory tested by the controller.

- counter_a_bits : *int* | auto ;

  The optional counter_a_bits property specifies the width of the general purpose CounterA counter. The maximum width that can be specified is 64 bits. The default value of auto gets converted to an integer that is the largest of the following:

  Number of counter bits required to count the number of rotate commands to move a bit across the length of the data register.

  Number of counter bits required to count the number of rotate commands to move a bit across the length of the Z address register plus the Y address register plus the X address register.

- delay_counter_bits : *int* | auto ;

  The optional delay_counter_bits property specifies the number of bits in the general purpose DelayCounter. The default auto value for delay_counter_bits gets converted into an integer based on the MemoryType property of the Memory library wrapper—see "Tessent MemoryBIST Library Algorithms" in the *Tessent MemoryBIST User's Manual*. If the controller only test memories of type SRAM or ROM, the auto value gets converted to 0. If the controller tests one or more memories of type DRAM, the auto value gets converted to a number large enough to meet the refresh requirements of the DRAM memories taking into account the period of the clock used by the controller.

- preserve_microcode_initial_values : on | off ;

  When the property is specified to on and the memory BIST controller is soft programmable (the property soft_instruction_count is greater than 0) and at least one of the memories tested by the controller is a multi-port memory other than a 1R1W configuration, the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

preserve_microcode_initial_values property inserts shadow registers for the following controller registers:

X, Y, and Z segments of Address Register A

X, Y, and Z segments of Address Register B

Write data register

Expect data register

When a soft-coded algorithm is applied at runtime, the microcode values are loaded once into the soft programmable controller at the start of the test. The shadow registers replicate the initial values of the registers above. For multi-port memory testing, the algorithm is applied for each test port. Since the soft-coded algorithm is not reloaded for each test port, the microcode initial values must be preserved as the actual registers might be manipulated during the algorithm execution.

Depending on the memory word width and the number of words, the shadow registers might add significant area overhead to the soft programmable controller. An alternative approach is to specify this property to off and use the freeze_test_port property inside the AdvancedOptions wrapper to apply the soft-coded algorithm to one specific test port of the multiport memory at a time.

- soft_algorithm_address_min_max : on | off ;

  The soft_algorithm_address_min_max property allows the user to specify a user defined address count range in the soft algorithms. This must be set to on if any of the following properties are specified in the soft algorithm:

  - load_bank_address_min
  - load_bank_address_max
  - load_column_address_min
  - load_column_address_max
  - load_row_address_min
  - load_row_address_max

  If freeze_step is not specified and some, but not all, address count range properties are not set in the soft algorithm, each unspecified range is automatically set to the maximum default range for that address count, across all steps defined for the controller in the DftSpecification. This may result in address wrapping on smaller memories being tested among the differing steps. The freeze_step property should be used to limit testing to a specific controller step if this is an issue.

  If freeze_step is not specified, and all address count range properties are not set in the soft algorithm, the tool runs the default step address range for each step.

  If freeze_step is specified, and some or all address count range properties are not set in the soft algorithm, each unspecified range is automatically set to the default range for that step.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- max_data_inversion_address_bit_index : *int* | max_index ;

  The max_data_inversion_address_bit_index property determines the size of the multiplexer used by the invert_data_with_row_bit and invert_data_with_column_bit properties inside the DataGenerator wrapper of custom Algorithm wrapper—see "Defining a Custom Algorithm" in the *Tessent MemoryBIST User's Manual*. The address bit selection logic inside the memory BIST controller creates critical paths that may affect timing closure when synthesizing at high clock frequencies. You should only specify a non-zero value for this property unless you intend to use the invert_data_with_row_bit or invert_data_with_row_bit property with values other than r[0] and c[0] respectively. When unspecified, this property defaults to 0. Valid values are as follows:

  integer — limits the highest bit index of the row and column address that can be specified for the invert_data_with_row_bit and invert_data_with_column_bit properties. Valid values are from 0 to N.

  max_index — allows any row or column address bits to be selected for write or expect data register inversion.

- a_equals_b_command_allowed : on | off ;

  The a_equals_b_command_allowed property specifies whether the comparator that is used for the address_a_equals_b property (used in the InstructionData wrapper) is generated inside the memory BIST controller—see "Instruction/AdvancedOptions" in the *Tessent MemoryBIST User's Manual*. The presence of the address comparator in the memory BIST controller creates critical paths that may affect timing closure when synthesizing at high clock frequencies. It is recommended to set the pipeline_controller_outputs property (see AdvancedOptions) property to on when this property is set to on. If you do not intend to define custom algorithms that use the address_a_equals_b property, the property should be set to off to facilitate timing closure.

  When a_equals_b_command_allowed is set to off, custom algorithms cannot use the address_a_equals_b feature. That is, the address_a_equals_b property must be off in all InstructionData wrappers—see "Instruction/DataCommands" in the *Tessent MemoryBIST User's Manual*.

- address_segment_x0_y0_allowed : on | off ;

  The address_segment_x0_y0_allowed property allows the row and column address counters to be further segmented. The row address counter may be sub-divided into a X1 address counter and a X0 address counter. The column address counter may be sub-divided into a Y1 address counter and a Y0 address counter. The algorithm can use these address segments to achieve complex address count sequences. The size of the individual address segment is configured through the properties max_x0_segment_bits and max_y0_segment_bits in the algorithm.

- max_x0_segment_bits : 1 | auto ;

  The optional max_x0_segment_bits property specifies the limit of the X0 address segment size.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When max_x0_segment_bits is set to 1, the address counter logic is modified so that all critical single-cycle paths to the other registers of the memory BIST controller are removed. Corresponding multi-cycle constraints are added to relax these paths during synthesis, timing analysis, and layout. This facilitates the timing closure of the controller hardware when operating at high clock frequency and when incrementing/decrementing the address in consecutive clock cycles of custom algorithms. The value of 1 for max_x0_segment_bits is suitable for all Tessent library algorithms.

Setting max_x0_segment_bits to Auto allows the implementation of address sequences incrementing or decrementing the address by 2 or more in custom algorithms. The address can be incremented or decremented in consecutive cycles of custom algorithms but the presence of additional critical single-cycle paths might limit the maximum operational speed of the controller, especially for soft-programmable controllers which allow all possible X0 address segment sizes. If a hard-programmable controller is implemented, the tool automatically determines the maximum size of the X0 address segment required. This allows to minimize area and the number of critical single-cycle paths.

Please refer to the description of column_address_count_enable and row_address_count_enable in the Cycle/AdvancedSignals wrapper properties for additional usage conditions concerning the implementation of custom algorithms requiring the address to be incremented/decremented in consecutive clock cycles.

The following usage conditions apply:

- Refer to the row_address_count_enable usage conditions for the max_x0_segment_bits property usage when implementing custom algorithms requiring the address to be incremented/decremented in consecutive clock cycles.

- max_y0_segment_bits : 1 | auto ;

  The optional max_y0_segment_bits property specifies the limit of the Y0 address segment size.

  When max_y0_segment_bits is set to 1, the address counter logic is modified so that all critical single-cycle paths to the other registers of the memory BIST controller are removed. Corresponding multi-cycle constraints are added to relax these paths during synthesis, timing analysis, and layout. This facilitates the timing closure of the controller hardware when operating at high clock frequency and when incrementing/decrementing the address in consecutive clock cycles of custom algorithms. The value of 1 for max_y0_segment_bits is suitable for all Tessent library algorithms.

  Setting max_y0_segment_bits to Auto allows the implementation of address sequences incrementing or decrementing the address by 2 or more in custom algorithms. The address can be incremented or decremented in consecutive cycles of custom algorithms but the presence of additional critical single-cycle paths might limit the maximum operational speed of the controller, especially for soft-programmable controllers which allow all possible Y0 address segment sizes. If a hard-programmable controller is implemented, the tool automatically determines the maximum size of the Y0 address segment required. This allows to minimize area and the number of critical single-cycle paths.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Please refer to the description of column_address_count_enable and row_address_count_enable in the Cycle/AdvancedSignals wrapper properties for additional usage conditions concerning the implementation of custom algorithms requiring the address to be incremented/decremented in consecutive clock cycles.

The following usage conditions apply:

- Refer to the column_address_count_enable usage conditions for the max_y0_segment_bits property usage when implementing custom algorithms requiring the address to be incremented/decremented in consecutive clock cycles.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

MemoryCluster

Specifies the controller properties to generate a memory BIST controller to test memories that use a shared bus interface.

Usage

```
DftSpecification(module_name,id) {
  MemoryBist {
    Controller(id) {
      MemoryCluster(cluster_id) {
        instance_name                : instance_name ;
        memory_cluster_library_name  : library_name ;


        pipeline_cluster_inputs      : on | off ;     // *DefSpec
        pipeline_cluster_outputs     : on | off ;     // *DefSpec
        memory_access_level          : auto | logical | physical ;
                                                      // *DefSpec
        repair_analysis_present      : auto | off ;   // *DefSpec
        max_repair_group_size        : unlimited |
                                       {int[kilobits | megabits]} ;
                                       // *DefSpec
        repair_group_scope           : logical_memory |
                                       physical_memory | controller ;
                                       // *DefSpec
        repair_sharing               : on | off ;
      }
    }
  }
}
```

Description

A wrapper that specifies the controller properties to generate a memory BIST controller to test memories that use a shared bus interface. The MemoryCluster and Step wrappers are mutually exclusive. The shared bus interface provides common access to multiple memories. Please refer to the *Tessent MemoryBIST User's Manual* for further details.

Arguments

- instance_name : *instance_name* ;

  A property that specifies the instance path of a memory cluster that is connected to a shared bus memory BIST controller

- memory_cluster_library_name : *library_name* ;

  An optional property that specifies the cluster library template name that is used to generate a shared bus memory BIST controller. This is only needed if you wish to generate a memory BIST controller without insertion.

- pipeline_cluster_inputs : on | off ;

  A property that specifies whether the memory cluster inputs from the memory BIST controller is pipelined with a single pipelining stage to help with meeting timing closure.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- pipeline_cluster_outputs : on | off ;

  A property that specifies whether the memory cluster outputs to the memory BIST controller is pipelined with a single pipelining stage to help with meeting timing closure.

- memory_access_level : auto | logical | physical ;

  A property that specifies the view of all memories in the memory cluster that the memory BIST controller operates on. The memories accessed through the shared bus interface are called logical memories. A logical memory is an address space composed of one or more physical memories.

  When the value is set to logical, the physical composition of the logical memory is irrelevant. All memories that form the logical memory are tested in one pass of the test algorithm. Repair and diagnosis information is reported with respect to the address space and I/O of the logical memory. This option is also suitable if you want to implement repair sharing for the physical memories that you integrate into the logical memory.

  When the value is set to physical, the memory BIST controller operates at the physical memory level, with a virtual memory being created for each individual physical memory that forms the logical array. Each physical memory has its own memory BIST interface and a dedicated controller step associated with the physical memory access code, if it is defined in the MemoryGroupAddressDecoding wrapper in the logical memory core library. In the case where a RedundancyAnalysis wrapper is present in the logical memory core library, its content is disregarded when the physical option is chosen.

  When the value is set to auto, the access level is determined for each logical memory from the specification of its logical memory core library, weighing the pros and cons of the two options described above. The auto value is converted to physical if all of the following conditions are met, otherwise a logical view is created:

  - At least one PhysicalToLogicalMapping wrapper exists in the logical memory core library that defines the relationship between physical and logical memory ports.
  - The logical memory has no BIRA logic associated with it - no RedundancyAnalysis wrapper is present in the logical memory core library.

- repair_analysis_present : auto | off ;

  A property that specifies if repair analysis logic is generated for the indicated Shared Bus memory cluster. If this property is set to auto and the Shared Bus logical or physical memory's library descriptions (as specified by memory_access_level) contain at least one RedundancyAnalysis/SpareElement wrapper, the repair analysis logic is generated. Refer to the "RedundancyAnalysis" section in the *Tessent MemoryBIST User's Manual* manual for more information.

- max_repair_group_size : unlimited | {*int*[kilobits | megabits]} ;

  A property that specifies the total memory size limit on a shared bus memory group when repair sharing is enabled with set_memory_instance_options. This property is used during logical/physical access level expansion when the process_dft_specification command is run. The default value of "unlimited" indicates that any number of compatible shared bus

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

memories can be assigned to repair groups. The *int*[units] pair specifies the maximum size allowed for a repair group. Any shared bus memory that would cause this limit to be exceeded is assigned to a different repair group. The unit property of kilobits equates to a scaling multiplier of 2^{10} bits (1024 bits) and megabits equates to 2^{20} bits (1048576 bits).

- repair_group_scope : logical_memory | physical_memory | controller ;

  A property used to specify how repair grouping is conducted during logical/physical memory access level expansion for shared bus memories when the process_dft_specification command is run and repair sharing is enabled with set_memory_instance_options.

  The selected value impacts the repair grouping as follows:

  - logical_memory — Specifies the repair groups are restricted to compatible physical memories within a logical memory without exceeding the size specified by the max_repair_group_size property in the MemoryCluster wrapper.
  - physical_memory — Specifies the repair group is restricted within a physical memory. Memories with multiple RowSegment and/or ColumnSegment wrappers are considered as having a single RowSegment and/or ColumnSegment wrapper for redundancy analysis as long as the memory size does not exceed the size specified by the max_repair_group_size property in the MemoryCluster wrapper.
  - controller — Specifies the repair group is restricted to all memories tested by the controller. The repair group cannot contain memories tested by other controllers. Memories are automatically assigned to repair groups as long as the memory size does not exceed the size specified by the max_repair_group_size property in the MemoryCluster wrapper.

- repair_sharing : on | off ;

  A property that globally enables or disables repair sharing for shared bus memories that have “-repair_sharing auto” specified, which is the default. The “-repair_sharing” value can be changed from “auto” for a memory instance with the set_memory_instance_options command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

RepairOptions

Provides repair-related options and properties.

Usage

```
DftSpecification(module_name,id) {
  MemoryBist {
    Controller(id) {
      RepairOptions {
        fuse_set_extraction_sequence     : address_before_fuse_map |
                                           fuse_map_before_address ;
        row_bira_location                : controller |
                                           follow_comparators ;
        enable_multicycle_operation      : on | off | auto ;
      }
    }
  }
}
```

Description

The RepairOptions wrapper specifies what to shift out and monitor with repairable memories, as well as the spare allocation method to use.

Arguments

- fuse_set_extraction_sequence : address_before_fuse_map | fuse_map_before_address ;

  A property that defines the setup chain sequence of the Column/IO BIRA registers that report the faulty column address and the IO decode value. This property specifies whether the faulty column address or the IO decode value is shifted out first when the repair analysis information is extracted from the memory BIST controller. This property can be used when processing of the repair data is done and there is a preference on the sequence of the repair fields in the serial stream. Valid values are as follows:

  AddressBeforeFuseMap — The faulty column address is shifted out before the IO decode value. The MSB of the IO decode register connects to the LSB of the faulty column address register.

  FuseMapBeforeAddress — The IO decode value is shifted out before the faulty column address. The MSB of the faulty column address register connects to the LSB of the IO decode register.

- row_bira_location : controller | follow_comparators ;

  A property that specifies where the Row BIRA module is located. The Row BIRA module can be located in the same module as the comparators or it can be forced to be inside the memory BIST controller.

  When shared comparators are used (comparator_location:shared_in_controller), the Row BIRA module is always located inside the memory BIST controller. However, when local comparators are used (comparator_location:per_interface), the Row BIRA module can be placed inside the interface or inside the controller.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For a memory BIST controller with several memories with Row BIRA, placing the Row BIRA module inside the controller results in a reduction of area due to the sharing of common redundancy analysis logic and pipelining stages.

- enable_multicycle_operation : on | off | auto ;

  A property that allows the memory BIST redundancy analysis circuit to use multi-cycle operations. When enabled, this option simplifies timing closure of the memory BIST BIRA circuit when operating at high frequency and in the presence of wide memories with IO or Column redundancy.

  When set to “off”, the BIRA circuit is implemented using single cycle paths. This option is recommended when using custom operation sets that perform StrobeDataOut on consecutive clock cycles.

  When set to “on”, the BIRA circuit is implemented using multi-cycle paths. This option is recommended when testing wide memories with IO or Column redundancy and when running at high frequency. The use of library operation sets is recommended. If custom operation sets are used, they should not execute StrobeDataOut on consecutive clock cycles.

  The “auto” value resolves to “on” if there are no custom operation sets executing StrobeDataOut on consecutive clock cycles. Otherwise, it resolves to “off”.

  Use of operation sets that implement StrobeDataOut in consecutive clock cycles is restricted when this option is enabled. If the BIRA circuit was implemented for multi-cycle operations, it is not allowed to compare the memory output in consecutive clock cycles when performing repair analysis. This might happen when selecting a combination of custom algorithm and operation sets in the PatternsSpecification. A rule check prevents a pattern from being generated in that case, and the user needs to modify the test configuration

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

to execute the algorithm in two test steps using the “odd” and “even” data compare time slots and accumulate the results as shown in the following example:

```
TestStep(pre_repair_even) {
      MemoryBist {
        run_mode : run_time_prog;
        reduced_address_count : off;
        Controller(chip_rtl_tessent_mbist_c1_controller_inst) {
          RepairOptions {
            check_repair_status : non_repairable;
          }
          DiagnosisOptions {
            StopOnErrorOptions {
              data_compare_time_slots : even;
            }
            compare_go : on;
          }
        }
      }
    }
    TestStep(pre_repair_odd) {
      MemoryBist {
        run_mode : run_time_prog;
        reduced_address_count : off;
        DiagnosisOptions {
          preserve_fuse_register_values : on;
        }
        Controller(chip_rtl_tessent_mbist_c1_controller_inst) {
          RepairOptions {
            check_repair_status : non_repairable;
          }
          DiagnosisOptions {
            StopOnErrorOptions {
              data_compare_time_slots : odd;
            }
            compare_go : on;
          }
        }
      }
    }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DiagnosisOptions

Specifies the diagnosis options that affect diagnostic hardware generation for the memory BIST controller.

Usage

```
DftSpecification(module_name,id) {
  MemoryBist {
    Controller(id) {
      DiagnosisOptions { *DefSpec
        comparator_selection_mux          : on | off | auto ;
        go_status                         : auto | per_memory ;
        StopOnErrorOptions {
          failure_limit                   : int | off ;
        }
      }
    }
  }
}
```

Description

A wrapper that specifies the diagnosis options that affect diagnostic hardware generation for the memoryBIST controller. For information on the diagnosis approaches supported by Tessent MemoryBIST, including a description of features and a guide for using each approach, refer to the "Tessent MemoryBIST Diagnosis" Chapter in the *Tessent MemoryBIST User's Manual For Use With Tessent Shell*.

Arguments

- comparator_selection_mux : on | off | auto ;

  A property that specifies whether bit-level diagnostic resolution is maintained when local_comparators_per_go_id or shared_comparators_per_go_id is set to a value greater than 1. The valid values are:

  on — Specifies that a multiplexer is inserted at the input of each GO_ID register to select the output of only one comparator at a time and provide bit-level diagnostic resolution.

  off — Specifies that multiplexers are not inserted at the input of GO_ID registers.

  auto — Infers "on" when the local_comparators_per_go_id or shared_comparators_per_go_id property is set to a value greater than 1; otherwise, infers "off".

- go_status : auto | per_memory ;

  A property that specifies how a pass/fail status per individual memory is generated for the current controller.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The per-memory status register allows you to determine which memories failed without having to examine the individual compare status registers (GO_ID bits). The valid values are:

auto — A per-memory pass/fail status register is generated when memories with redundancy are present for the current MemoryBIST controller. If there are no memories with redundancy for the current controller, then the per-memory status register is not added. This is the default value.

per_memory — A per-memory pass/fail status register is added for all memories on the current MemoryBIST controller, even if there are no memories with redundancy.

- StopOnErrorOptions/failure_limit: *int* | off ;

  A property that specifies the maximum number of failures that the memory BIST controller can log in the Stop-On-Nth-Error mode of diagnosis. The default value becomes 4096 when the controller tests at least one memory of type SRAM or DRAM. It becomes “off” when the controller tests only memories of type ROM. The error counter is no longer used for ROM diagnosis as it was in the LV flow. Instead the functional debug mode is used to perform ROM diagnosis.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Step

Specifies the properties for memory BIST steps.

Usage

```
DftSpecification(module_name,id) {
  MemoryBist {
    Controller(id) {
      Step { // repeatable
        algorithm                       : algo_name ;          // *DefSpec
        operation_set                   : opset_name ;
                                          // default: from_library
        comparator_location             : shared_in_controller |
                                          per_interface ;     // *DefSpec
        bist_data_in_pipelining       : on | off | int ;       // *DefSpec
        bist_data_out_pipelining      : on | off | per_port ; // *DefSpec
        MemoryInterface(id) {                                 // repeatable
          generate_external_repair_logic : on | off ;
          instance_name                 : inst_name ;
          memory_library_name           : mem_lib_name ;
          repair_analysis_present       : auto | off ;         // *DefSpec
          repair_group_name             : none | group_name ;
          scan_bypass_logic             : async_mux | none | sync_mux |
                                          from_library ; // *DefSpec


          local_comparators_per_go_id   : int | all ;          // 1 *DefSpec
          rom_content_file              : file_path;
          output_enable_control         : always_on | system ;
          observation_xor_size          : auto | off ;
          data_bits_per_bypass_signal   : 1..MaxPosInt | all ;// 1 *DefSpec
        }
        ReusedMemoryInterface(id) {       // repeatable *DefSpec
          instance_name                 : inst_name ;
          reused_interface_id           : [ctrl_id:]mem_interface_id ;
          repair_group_name             : none | group_name ;
        }
      }
    }
  }
}
```

Description

A wrapper that specifies the properties for memory BIST steps. These properties are used to generate specific controllers' configurations. The Step and MemoryCluster wrappers are mutually exclusive. See also "Tessent Core Description" on page 4793 for Memory wrapper details.

Arguments

- algorithm : algo_name | from_library ;

  The algorithm property specifies the name of the algorithm that is applied to all memories tested in this step. If the "from_library" value is used, the algorithm specified in the Memory

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

library description of all memories tested by the controller in the given Step is used. An error is generated if the "from_library" value is used and the memories tested by the controller in the given Step do not have the same value for the algorithm property.

- operation_set : *opset_name* ;

  The operation_set property specifies the name of the operation set that is applied to all memories tested in this step. If the "from_library" value is used, the OperationSet specified in the Memory library description of all memories tested by the controller in the given Step is used. An error is generated if the "from_library" value is used and the memories tested by the controller in the given Step do not have the same value for the OperationSet property.

- comparator_location : shared_in_controller | per_interface ;

  The comparator_location property specifies where comparators for the memory BIST controller are placed, with the default being the placement of the comparators in the memory interface. Using the per_interface value greatly reduces the routing area needed to bring the BIST data signals from the memory interfaces to the controller. You also get per memory diagnostic resolution in go-nogo tests even when the controller has more than one step when the comparators are inside the interfaces. You can use the local_comparators_per_go_id property to reduce the area overhead.

- bist_data_in_pipelining : on | off | *int* ;

  The bist_data_in_pipelining property defines the number of pipeline stages to be added to all BIST inputs in the memory interface. Setting the value "on" is equivalent to 1 pipeline stage. You can use this property to facilitate timing closure on paths from the controller. The dedicated pipeline flip-flops allow the layout tool to place the flip-flops at the optimal distance for each memory. Typically, a single pipeline stage is sufficient. The MemoryBist/Controller/AdvancedOptions/pipeline_controller_outputs option is much less costly and is typically good enough to meet timing. The pipeline registers are added in the memory interface on the BIST signals from the controller. Some signals propagate to the memory while other signals are internal to the memory interface or controller and control the test execution. Examples of test signals that are pipelined when bist_data_in_pipelining is specified are address and control to the memory, write data pattern, expect data pattern, compare enable, and checkerboard pattern enable.

  When using the bist_data_in_pipelining property, you do not need to update any operation set because the pipelining is taken into account when writing to and reading from the memory.

- bist_data_out_pipelining : on | off | per_port ;

  A property that specifies whether the BIST data from the memory to the comparators is pipelined with a single pipelining stage. The pipelining stage is inserted in the memory interface.

  Note

  MemoryBIST does not create BIST data out pipeline registers for ROM memory types.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The valid values and their effects are as follows:

on — Pipelines the BIST data out to the comparators. You specify on if the BIST data output does not have time to propagate to the comparator. Pipelining might be required when the comparators are located in the memory BIST controller due to the physical distance between the memory controller and memory interface.

off — Does not pipeline the BIST data out.

per_port — Pipelines the BIST data out per memory output port. For multi-port memories, the data from each output port is multiplexed into a single BIST data out to the comparators. You specify per_port to insert separate pipelining flip-flops on each data output port before the multiplexer.

- MemoryInterface(*id*)/generate_external_repair_logic : on | off ;

  A property that specifies whether external IO memory repair logic is added to the memories associated with the specified memory interface. The default value is off. This property is automatically populated with a value of on by the create_dft_specification command if "set_memory_instance_options -generate_external_repair_logic on" was previously specified for a memory instance using the memory interface. For further information, refer to "External Repair" in *Tessent MemoryBIST User's Manual for use With Tessent Shell*.

- MemoryInterface(*id*)/instance_name : *inst_name* ;

  A property that specifies the instance name of the memory interface associated with the given memory. This property is required when running the process_dft_specification command without the -no_insertion option. In this case, the value of the tcd_memory_name Instance attribute attached to the instance defines the Memory library associated to that instance. When you use the -no_insertion flow, the elaborated design that contain the instantiated memories is optional. When the design is not present, the Memory library associated with the MemoryInterface is defined using the memory_library_name.

- MemoryInterface(*id*)/memory_library_name : *mem_lib_name* ;

  A property that specifies the name of the memory library associated with the memory tested by the memory interface. This property is only used and needed if you are running the process_dft_specification command with the -no_insertion option and the design is not elaborated in memory. Otherwise, the memory_library_name is extracted from the tcd_memory_name attribute attached to the instance referenced by the MemoryInterface(id)/instance_name property.

- MemoryInterface(*id*)/repair_analysis_present : auto | off ;

  A property that specifies whether repair analysis logic is generated for the memory tested by the memory interface. If set to auto and the memory library description associated to the memory tested by the memory interface contains at least one RedundancyAnalysis/ SpareElement wrapper, the repair analysis logic is generated. See "Tessent Core Description" on page 4793 for RedundancyAnalysis wrapper information.

- MemoryInterface(*id*)/repair_group_name : none | *group_name* ;

  A property that specifies the group name for the memory interfaces that share a BIRA/BISR module. This property is automatically filled in when the create_dft_specification command

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is run and repair sharing is enabled with set_memory_instance_options. The user can update this property to manually change the repair group assignments.

- MemoryInterface(*id*)/scan_bypass_logic : async_mux | none | sync_mux | from_library ;

  A property that specifies when and how memories are bypassed during scan testing. Bypassing memories enables testing of the user interface logic to and from these embedded memories as well as to and from the memory BIST controller and memory interface circuitry.

  If the specified value for MemoryInterface(id)/scan_bypass_logic equals “sync_mux”, “async_mux” or “none”, it overrides the TransparentMode property in the Memory library description. If the value is not specified or set to “from_library”, the value in the Memory library file takes effect.

  Valid values are as follows:

  async_mux — Inserts an additional multiplexer on the memory interface DataOut ports so that data is directly transferred from the DataIn ports to the DataOut ports. This setting enables combinational ATPG tools to test the interface between the user logic and the memory. It is not recommended for at-speed testing because the timing paths are not consistent with the functional mode.

  sync_mux — Inserts an additional multiplexer and flip-flop on the interface DataOut ports that enable ATPG tools to test the interface between the user logic and the memory without having to enable testing through the memory. This mode is very efficient for ATPG and provides very consistent timing paths between test and functional modes.

  none — Specifies that no bypass logic is added in the memory interface. Specify None if you have a bidirectional data bus or the memory implements internal bypass logic.

  from_library — Specifies that the bypass logic is generated according to the TransparentMode property setting of the Memory library file.

- MemoryInterface(*id*)/local_comparators_per_go_id : *int* | all ;

  A property that specifies the number of comparators to share with a given GO_ID register located in the interface. You can use this feature to manage the area overhead by reducing the number of registers used to log the comparator status. However, using a value other than 1 complicates and lengthens diagnosis with bit-level resolution as you need to repeat the test as many times as there are comparators sharing a common GO_ID register. For this diagnosis, you must modify the PatternsSpecification and create separate patterns for each possible integer value with the MemoryInterface/ property to manually iterate through the comparators. No automation is present for this process and you need to follow a similar manual diagnosis flow when using Tessent SiliconInsight Desktop.

  The value should be set to 1 to enable reading complete memory words in a single read operation. Also, the MemoryBist/Controller/AlgorithmResourceOptions/data_register_bits property should be set to an appropriate value based on the complexity of the patterns to be written to memory. For complete control of the write data, this property should be set to the number of bits of the widest memory tested by the controller.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For memories that implement IO/Column repair analysis or Row and IO/Column repair analysis, the number of comparators per GO_ID must be 1. The number is automatically adjusted whenever required and a warning is issued.

The default value is 1. Specifying the value "all" infers that there is one GO_ID register for all comparators associated with a memory.

- MemoryInterface(*id*)/rom_content_file : *file_path* ;

  A property that specifies the location of the ROM content file. The path is absolute or relative to the working directory. This property is mandatory when the memory tested by the memory interface is of type ROM. The ROM content file is used to compute the signatures associated with all algorithms applicable to ROMs and built into the controller. See the Step/Algorithm property and the AdvancedOptions/extra_algorithms to understand which algorithm gets built into the controller.

- MemoryInterface(*id*)/output_enable_control : always_on | system ;

  The output_enable_control property should only be specified when the design is not elaborated. Otherwise, it is recommended to let Tessent Shell analyze the design and infer the proper output_enable_control setting based on design connectivity. When the design contains many non-unique instances of the same memory interface with different output enable connection configurations, Tessent Shell applies the value that accommodates all memory instances.

  The output_enable_control property is used to determine the type of data output stage (2-state or 3-state) when generating the memory interface modules, as well as the logic that controls the output enable ports of the memory and interface modules. When set to always_on, a 2-state output multiplexer is generated. When set to system, a 3-state multiplexer is generated.

  Specifying The output_enable_control Based on Design:

  Set output_enable_control to always_on when the memory output enable port is tied to a constant value in the functional design. This maintains the memory data output drivers (tri-state) into an always on state. The memory BIST interface generated for this memory intercepts the data output with a standard 2-state mux and drives the values from the bypass registers during scan if the memory bypass is selected during scan. Else, the interface selects the memory output during functional mode or when applying multiple load scan patterns to memory. This does not cause any net contention with other memories in the design, and is the most common situation.

  Set output_enable_control to system when the memory output enable port is driven by functional logic. This is typically the case when many memories are connected to the same data bus. It is assumed that the functional logic enables a single memory output at any time and avoids bus contentions. The memory BIST interface generated for this memory must also have a 3-state output. During scan mode, the memory interface observes the memory output enable signal and either enables the output drivers from the memory interface (when memory bypass is enabled), or the memory output drivers (when applying multiple load scan patterns to memory).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- MemoryInterface(*id*)/observation_xor_size : auto | off ;

  A property that specifies whether to insert scan observation points for the address and control ports in the memory interface. The auto setting combines the signals based on the observation_xor_size property in the MemoryBist/Controller/AdvancedOptions wrapper. The off setting disables the observation logic insertion. The default setting is auto.

- data_bits_per_bypass_signal : 1..MaxPosInt | all ;

  A property that specifies the number of functional data input signals to be combined in an XOR tree, and captured into one memory bypass flop. The setting is meaningful when building bypass logic around a RAM. The MemoryInterface/scan_bypass_logic property must be different than none, or the TransparentMode setting in the Memory library file must be different than None. The minimum value is 1, which assigns one bypass signal per data signal. The maximum value is the memory word size. The all value combines all data input signals from a logical port into a 1-bit bypass path. The default value is 1.

- ReusedMemoryInterface(*id*)/instance_name : *inst_name* ;

  A property that is used to associate a memory interface module with another memory instance. You can use this wrapper when the memory instance has the same module name as the one for which the reused memory interface is for. The generation of this wrapper by the create_dft_specification command is controlled by the DefaultsSpecification MemoryBist/memory_interface_reuse_method property. By default, the create_dft_specification command uses this wrapper when the parent module of the memory instance is not unique to avoid having to uniquify the parent module during insertion. It also uses this wrapper when the memory instance is inside a generate loop to avoid having to unroll the loop during insertion. This property is optional when the process_dft_specification command is run with the -no_insertion option.

- ReusedMemoryInterface(*id*)/reused_interface_id : [ctrl_id:]mem_interface_id ;

  A property that specifies the memory interface that is to be reused by the given memory instance. The value is the id of the controller wrapper, a colon, and the id of the memory_interface wrapper that is reused. The id of the controller and the colon is optional when the reused MemoryInterface belongs to the same controller as the ReusedMemoryInterface wrapper.

- ReusedMemoryInterface(*id*)/repair_group_name : none | *group_name* ;

  A property that specifies the group name for the reused memory interfaces that share a BIRA/BISR module. This property is automatically filled in when the create_dft_specification command is run and repair sharing is enabled with set_memory_instance_options. The user can update this property to manually change the repair group assignments.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ReusedController

Specifies the insertion of the second instance of a memory BIST controller.

Usage

```
DftSpecification(module_name,id) {
  MemoryBist {
    ReusedController(id) {  // repeatable
      parent_instance      : inst_path ;
      leaf_instance_name   : inst_name ;
      clock_domain_label   : name ;
      reused_controller_id : id ;
      Step { //repeatable
        MemoryInterface(id) { // repeatable
          instance_name : name ;
        }
      }
      MemoryCluster(cluster_id) {
        instance_name : name ;
      }
    }
  }
}
```

Description

A wrapper that specifies the insertion of the second instance of a memory BIST controller. The create_dft_specification command uses this wrapper when it detects that two controllers test the exact same set of memory modules and that the common ancestor of each of those two sets are repeated instances of a single module. To avoid having to lift the controller into a unique location or having to uniquify the repeated module, the memory BIST controller module is reused. The ReusedController/Step and ReusedController/MemoryCluster wrappers are mutually exclusive.

Arguments

- parent_instance : *inst_path* ;

  An optional property that specifies where the memory BIST controller is instantiated in the design. When unspecified, it defaults to a null string which is interpreted as the common instance ancestor of all memories tested by the controller. See the description of the get_common_parent_instance command to see how the common ancestor is computed.

- leaf_instance_name : *inst_name* ;

  An optional property that specifies the memory BIST controller instance name explicitly. By default, an instance name is automatically generated by appending the "_inst" string to the module name. This property must be identical in all Controller/ReusedController wrappers which have their parent_instance specified or inferred to point to instances of the same repeated module. For controllers testing shared bus memory clusters, this property must be identical in all Controller/ReusedController wrappers.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- clock_domain_label : *name* ;

  A property that specifies the label (see add_clocks) of the clock domain used by the memory BIST controller and all the associated memories. Any memory tested by the controller that has a clock port sourced by a different clock domain has a multiplexer added in its interface module to inject the clock used by the controller when running memory BIST.

- reused_controller_id : *id* ;

  A property that specifies the id of a controller wrapper that is to be reused.

- Step/MemoryInterface(*id*)/instance_name : *name* ;

  A property that specifies the instance name of the memory to be tested by the given memory interface in the given Step.

- MemoryCluster(*cluster_id*)/instance_name : *name* ;

  A property that specifies the instance name of the memory cluster to be tested by the given controller.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BistAccessPort

A wrapper that configures the MemoryBIST BIST Access Port (BAP) module instantiated in the design.

Usage

```
DftSpecification(module_name,id) {
  MemoryBist {
    BistAccessPort {
      parent_instance      : inst_path ;
      leaf_instance_name   : inst_name ;
      AdvancedOptions {
        use_multicycle_paths : on | off ; // *DefSpec
      }
      DirectAccessOptions {  // *DefSpec
      }
      Connections {
      }
    }
  }
}
```

Description

The BistAccessPort wrapper provides the capability to locate the memory BIST BAP instance at a specified location in the design hierarchy. The memory BIST BAP provides control signals to memory BIST controllers, collects their GO and DONE status signals and provides access to their internal registers. A memory BIST BAP is inserted for each pass when doing multi-pass insertion.

Arguments

- parent_instance : *inst_path* ;

  An optional property that specifies the hierarchical path where the memory BIST BAP module is instantiated in the design.

  When unspecified, it defaults to a null string that is interpreted as the common instance ancestor of all memory controllers driven by the memory BIST BAP. See the description of the get_common_parent_instance command to see how the common ancestor is computed. When *inst_path* is specified as “.”, the memory BIST BAP is instantiated in the top module of the design.

  This property is not used or needed when the process_dft_specification command is run with the -no_insertion option.

- leaf_instance_name : *inst_name* ;

  An optional property that specifies the memory BIST BAP instance name explicitly. By default, an instance name is automatically generated by appending the “_inst” string to the memory BIST BAP module name, which itself defaults to the following:

  *design_name_design_id*_tessent_mbist_bap

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The *design_id* is specified with the set_context -design_identifier command for the current design.

This property is not used or needed when the process_dft_specification command is run with the -no_insertion option.

- AdvancedOptions/use_multicycle_paths : on | off ;

  A property that specifies whether the generated SDC and STA scripts include multi-cycle paths pertaining to the memory BIST BAP. The default value of on includes these multi-cycle paths in the SDC and STA scripts.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DirectAccessOptions

A wrapper that implements the direct access interface on the BAP module and configures the memory BIST controller to allow memory test operations outside of the standard IJTAG network.

Usage

Usage 1: MemoryBist BistAccessPort wrapper

```
DftSpecification(module_name,id) {
  MemoryBist {
    BistAccessPort {
      DirectAccessOptions {
        direct_access              : on | off ; // *DefSpec
        direct_access_clock_source : common | per_bist_clock_domain ;
                                                  // *DefSpec
        ExecutionSelections {
        }
      }
    }
  }
}
```

Usage 2: MemoryBist Controller wrapper

```
DftSpecification(module_name,id) {
  MemoryBist {
    Controller(id) {
      DirectAccessOptions {
        ExecutionSelections {
        }
      }
    }
  }
}
```

Description

The DirectAccessOptions wrapper, when specified within the BistAccessPort wrapper, enables the generation of the direct access interface on the BAP module and specifies the clocking structure to be used for synchronizing and initiating the in-system memory test. For both usages, the ExecutionSelections wrapper is used for specifying the controller features that are configurable through the direct access interface. Each controller can be individually configured to support any combination of step, memory, algorithm and operation_set selections.

For more information on the implementation and use of the BAP direct access interface, refer to the "Advanced BAP Memory Access" Appendix in *Tessent MemoryBIST User's Manual For Use With Tessent Shell*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- direct_access : on | off ;

  A property that implements the direct access interface on the BAP module and adds the signals necessary for operating the MemoryBIST controller and providing the status of the memory test. Optional signals are added to the interface to access specific controller features, as specified by the user in the ExecutionSelections wrapper.

- direct_access_clock_source : common | per_bist_clock_domain ;

  A property that specifies the clocking scheme for the direct access interface during in-system memory test. The default setting of common generates a single sequencer to manage the memory test for all controllers attached to the BAP. For this setting, the clock must be at least four times slower than the lowest memory BIST clock frequency. When set to per_bist_clock_domain, a sequencer is generated for each memory BIST clock domain to manage the controllers operating on the same clock domain. Certain direct access interface ports are uniquified for each memory BIST clock domain, as indicated by the Connections/DirectAccess/ClockDomain wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ExecutionSelections

A wrapper that configures the optional direct access interface signals on the BAP module and MemoryBIST controller.

Usage

Usage 1: MemoryBist BistAccessPort wrapper

```
DftSpecification(module_name,id) {
  MemoryBist {
    BistAccessPort {
      DirectAccessOptions {
        ExecutionSelections {
          controller              : off | auto ; // *DefSpec
          step                    : off | auto ; // *DefSpec
          memory                  : off | auto ; // *DefSpec
          algorithm               : off | auto ; // *DefSpec
          operation_set           : off | auto ; // *DefSpec
          retention_test_phase    : off | auto ; // *DefSpec
          configuration_data      : off | auto ; // *DefSpec
          test_port               : off | auto ; // *DefSpec
          data_inversion_with_address_bit : off | auto ;  // *DefSpec
          bira_enable             : off | auto ; // *DefSpec
          check_repair_needed     : off | auto ; // *DefSpec
          preserve_fuse_register_values   : off | auto ; // *DefSpec


        }
      }
    }
  }
}
```

Usage 2: MemoryBist Controller wrapper

```
DftSpecification(module_name,id) {
  MemoryBist {
    Controller(id) {
      DirectAccessOptions {
        ExecutionSelections {
          step                    : on | off ; // *DefSpec
          memory                  : on | off ; // *DefSpec
          algorithm               : on | off ; // *DefSpec
          operation_set           : on | off ; // *DefSpec
          configuration_data      : on | off ; // *DefSpec
          test_port               : on | off ; // *DefSpec
          data_inversion_with_address_bit : on | off ; // *DefSpec
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Description

The ExecutionSelections wrapper within the BistAccessOptions wrapper (Usage 1) specifies the controller features that are configurable from the system control logic through ports of the BAP direct access interface. The input port is created on the BAP when at least one controller implements the corresponding selection feature.

The ExecutionSelections wrapper within a Controller wrapper (Usage 2) specifies the controller features that are directly configurable from the BAP. The necessary controller ports are created and connected to the BAP. Each controller can be individually configured to support any combination of step, memory, algorithm and operation_set selections. The selection features are programmed using IJTAG TDRs added into the BAP. Additionally, they can be controlled through the BAP direct access interface, if present. The selection features at the controller level can be used for manufacturing test independent of the BAP direct access interface.

Refer to "BAP Direct Access Interface Pins" in *Tessent MemoryBist User's Manual for use With Tessent Shell* for further information on BAP direct access interface ports.

Arguments

For Usage 1:

- controller : off | auto ;

  A property that enables direct controller selection for in-system memory test. The direct access interface port sys_ctrl_select is added to the BAP module when enabled. The auto value resolves to on if BistAccessPort/DirectAccessOptions/direct_access is on.

- step : off | auto ;

  A property that enables direct controller step selection for in-system memory test. The direct access interface ports sys_step_select and sys_step_select_en are added to the BAP module when enabled. The auto value resolves to off if BistAccessPort/DirectAccessOptions/direct_access is off. The auto value resolves to on if direct_access is on and at least one memory BIST controller enables the direct step selection.

- memory : off | auto ;

  A property that enables direct memory selection within a controller step for in-system memory test. The direct access interface ports sys_memory_select and sys_memory_select_en are added to the BAP module when enabled. The auto value resolves to off if BistAccessPort/DirectAccessOptions/direct_access is off. The auto value resolves to on if direct_access is on and at least one memory BIST controller enables the direct memory selection.

- algorithm : off | auto ;

  A property that enables direct controller algorithm selection for in-system memory test. The direct access interface ports sys_algo_select and sys_select_common_algo are added to the BAP module when enabled. The auto value resolves to off if BistAccessPort/DirectAccessOptions/direct_access is off. The auto value resolves to on if direct_access is on and at least one memory BIST controller enables the direct algorithm selection.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- operation_set : off | auto ;

  A property that enables direct operation set selection for in-system memory test. The direct access interface ports sys_opset_select and sys_select_common_opset are added to the BAP module when enabled. The auto value resolves to off if BistAccessPort/DirectAccessOptions/direct_access is off. The auto value resolves to on if direct_access is on and at least one memory BIST controller enables the direct operation set selection.

- retention_test_phase : off | auto ;

  A property that enables direct test phase selection when performing retention test, using SMarchCHKB based library algorithms, for in-system memory test. The direct access interface ports sys_retention_test_phase[1:0] are added to the BAP module when enabled. The auto value resolves to off if BistAccessPort/DirectAccessOptions/direct_access is off and resolves to on if direct_access is on.

  Refer to the "Parallel Retention Testing" section in *Tessent MemoryBIST User's Manual for use With Tessent Shell* for further information and test phase selection encoding for sys_retention_test_phase[1:0].

- configuration_data : off | auto ;

  A property that enables direct setting of the configuration data to the memory cluster interface for in-system memory test. The direct access interface ports sys_mbistcfg_interface<*interface_id*> and sys_mbistcfg_en are added to the BAP module, when enabled. Dedicated sys_<*ctl_id*>_<*cluster_id*>_reset<*x*> and sys_<*ctl_id*>_<*cluster_id*>_request<*x*> inputs for each memory cluster are present on the direct access interface if direct_access is on. The auto value resolves to on if direct_access is on and at least one memory BIST controller enables the direct access of the configuration_data, otherwise it resolves to off.

- test_port : off | auto ;

  A property that enables direct test port selection for in-system memory test. The direct access interface ports sys_test_port_select and sys_test_port_select_en are added to the BAP module when enabled. The auto value resolves to off if BistAccessPort/DirectAccessOptions/direct_access is off. The auto value resolves to on if direct_access is on and at least one memory BIST controller enables the direct test port selection.

- data_inversion_with_address_bit : off | auto ;

  A property that enables direct test data inversion based on address for in-system memory test. The direct access interface input ports sys_data_inv_col_add_bit_select, sys_data_inv_col_add_bit_select_en, sys_data_inv_row_add_bit_select, and sys_data_inv_row_select_en are added to the BAP module when enabled. The auto value resolves to off if BistAccessPort/DirectAccessOptions/direct_access is off. The auto value resolves to on if direct_access is on and at least one memory BIST controller enables the data inversion selection.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- bira_enable : off | auto ;

  A property that allows direct enabling of repair analysis for the memory BIST controller from the BAP. The direct access interface port sys_bira_en is added to the BAP module when enabled. The auto value resolves to off if BistAccessPort/DirectAccessOptions/direct_access is off. The auto value resolves to on if direct_access is on and at least one memory BIST controller has repair analysis capability.

- check_repair_needed : off | auto ;

  A property that allows direct enabling from the BAP of repair analysis status register examination after BIST completion. The direct access interface port sys_check_repair_needed is added to the BAP module when enabled. The auto value resolves to off if BistAccessPort/DirectAccessOptions/direct_access is off. The auto value resolves to on if direct_access is on and at least one memory BIST controller has repair analysis capability.

- preserve_fuse_register_values : off | auto ;

  A property that allows direct enabling from the BAP of preservation of repair analysis register values from a previous test step. This inhibits the loading of the BISR register values into the repair analysis registers when a new test step begins. The direct access interface port sys_preserve_fuse_register is added to the BAP module when enabled. The auto value resolves to off if BistAccessPort/DirectAccessOptions/direct_access is off. The auto value resolves to on if direct_access is on and at least one memory BIST controller has repair analysis capability.

For Usage 2:

- step : on | off ;

  A property that enables direct selection of controller step from the BAP. The input ports STEP_SELECT and STEP_SELECT_EN are added to the controller when enabled. In cases where there is only one controller step, an on setting automatically becomes an off setting.

- memory : on | off ;

  A property that enables direct selection of memories within a controller step from the BAP. The input ports MEM_SELECT and MEM_SELECT_EN are added to the controller when enabled. In cases where there is only one memory per controller step, an on setting automatically becomes an off setting.

- algorithm : on | off ;

  A property that enables direct selection of the test algorithm from the BAP. The input ports ALGO_SELECT and SELECT_COMMON_ALGO are added to the controller when enabled. In cases where there is only one hard coded algorithm, an on setting automatically becomes an off setting.

- operation_set : on | off ;

  A property that enables direct selection of the operation set from the BAP. The input ports OPSET_SELECT and SELECT_COMMON_OPSET are added to the controller when

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

enabled. In cases where there is only one operation set, an on setting automatically becomes an off setting.

- configuration_data : on | off ;

  A property that enables the direct setting of cluster configuration data from the BAP. The input ports INTERFACE<*interface_id*>_MBISTCFG_IN and MBISTCFG_EN are added to the controller when enabled.

- test_port : on | off;

  A property that enables the direct selection of the test port from the BAP. The input ports TEST_PORT_SELECT and TEST_PORT_SELECT_EN are added to the controller when enabled.

- data_inversion_with_address_bit : on | off;

  A property that enables the direct selection of address-based data inversion from the BAP. The following input ports are added to the controller when enabled:

  - DATA_INV_COL_ADD_BIT_SELECT
  - DATA_INV_COL_ADD_BIT_SELECT_EN
  - DATA_INV_ROW_ADD_BIT_SELECT
  - DATA_INV_ROW_ADD_BIT_SELECT_EN

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Connections

A wrapper that automates the connections between system control logic and the BAP direct access interface.

Usage

```
DftSpecification(module_name,id) {
 MemoryBist {
  BistAccessPort {
   Connections {
    DirectAccess {
     controller_done              : port_pin_name ;
     controller_pass              : port_pin_name ;
     controller_select            : port_pin_name ;
     step_select                  : port_pin_name ;
     step_select_enable           : port_pin_name ;
     memory_select                : port_pin_name ;
     memory_select_enable         : port_pin_name ;
     algorithm_select             : port_pin_name ;
     algorithm_select_enable      : port_pin_name ;
     operation_set_select         : port_pin_name ;
     operation_set_select_enable : port_pin_name ;
     retention_test_phase         : port_pin_name ;
     preserve_test_inputs         : port_pin_name ;
     test_port_select             : port_pin_name ;
     test_port_select_enable      : port_pin_name ;
     data_inversion_column_address_bit_select        : port_pin_name;
     data_inversion_column_address_bit_select_enable : port_pin_name;
     data_inversion_row_address_bit_select           : port_pin_name;
     data_inversion_row_address_bit_select_enable    : port_pin_name;
     bira_enable                    : port_pin_name ;
     check_repair_needed            : port_pin_name ;
     preserve_fuse_register_values : port_pin_name ;
     incremental_test_mode        : port_pin_name ;

     ClockDomain (domain_label) {   // repeatable
       clock      : port_pin_name ;
       reset      : port_pin_name ;
       test_init  : port_pin_name ;
       test_start : port_pin_name ;
       test_done  : port_pin_name ;
       test_pass  : port_pin_name ;
     }
    }
   }
  }
 }
}
```

Description

A wrapper that automates the connections between the system control logic and the BAP direct access interface when implementing in-system testing using the advanced BAP memory access

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

feature. The direct access interface must be implemented in the hardware by specifying the BistAccessPort/DirectAccessOptions/direct_access property to on.

The connections for controller_done, controller_pass, test_done, and test_pass must be internal input pins or output ports of the current design. All other connections must be internal output pins or input ports of the current design. It is not allowed to duplicate usage for the specified design ports and pins. If a system control logic port or pin is wider than its counterpart on the BAP direct access interface, only the number of bits equaling the width of the direct access interface is connected to the control logic and the remaining control logic bits are unused.

Note

An error is issued if the system control logic port or pin is smaller in width than its counterpart on the BAP direct access interface.

Note

A warning is issued during DFT insertion if a BAP direct access interface output (such as controller_done, controller_pass, test_done, or test_pass) connects to a system control logic port or pin that already has a drive source. The tool disconnects the driver in this situation.

Arguments

- controller_done : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_ctrl_done[*msb:lsb*] output ports of the BAP direct access interface. Single or bussed connections are allowed. The BAP output port indicates when a controller has completed the test execution.

- controller_pass : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_ctrl_pass[*msb:lsb*] output ports of the BAP direct access interface. Single or bussed connections are allowed. The BAP output port indicates a controller's pass/fail status of the test execution.

- controller_select : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_ctrl_select[*msb:lsb*] input ports of the BAP direct access interface. Single or bussed connections are allowed. The BAP input port selects one or more controllers for test execution.

- step_select : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_step_select[*msb:lsb*] input ports of the BAP direct access interface. Single or bussed connections are allowed. The BAP input port designates the controller step for test execution.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- step_select_enable : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_step_select_en input port of the BAP direct access interface. The BAP input port enables the controller step selection.

- memory_select : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_memory_select[*msb:lsb*] input ports of the BAP direct access interface. Single or bussed connections are allowed. The BAP input port designates the memory within a controller step for test execution.

- memory_select_enable : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_memory_select_en input port of the BAP direct access interface. The BAP input port enables the memory selection.

- algorithm_select : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_algo_select[*msb:lsb*] input ports of the BAP direct access interface. Single or bussed connections are allowed. The BAP input port designates the test algorithm to be applied.

- algorithm_select_enable : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_select_common_algo input port of the BAP direct access interface. The BAP input port enables the test algorithm selection.

- operation_set_select : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_opset_select[*msb:lsb*] input ports of the BAP direct access interface. Single or bussed connections are allowed. The BAP input port designates the operation set to be applied.

- operation_set_select_enable : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_select_common_opset input port of the BAP direct access interface. The BAP input port enables the operation set selection.

- retention_test_phase : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_retention_test_phase[1:0] input ports of the BAP direct access interface. The BAP input ports select the test phase when performing retention test using the SMarchCHKB based library algorithms.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- preserve_test_inputs : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_preserve_test_inputs input port of the BAP direct access interface. The BAP input port maintains the memories under the control of the MBIST circuit.

- test_port_select : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_test_port_select input ports of the BAP direct access interface. The BAP input ports select the memory test ports for test execution.

- test_port_select_enable : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_test_port_select_en input port of the BAP direct access interface. The BAP input port enables the memory test port selection for test execution.

- data_inversion_column_address_bit_select : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_data_inv_col_add_bit_select input port of the BAP direct access interface. The BAP input port selects the column address bit that inverts the applied write and expect data registers for test execution. The select width is set by the AlgorithmResourceOptions/max_data_inversion_address_bit_index property, which defaults to 0 and corresponds to c[0].

- data_inversion_column_address_bit_select_enable : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_data_inv_col_add_bit_select_en input port of the BAP direct access interface. The BAP input port enables column address bit selection for write and expect data inversion.

- data_inversion_row_address_bit_select : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_data_inv_row_add_bit_select input port of the BAP direct access interface. The BAP input port selects the row address bit that inverts the applied write and expect data registers for test execution. The select width is set by the AlgorithmResourceOptions/max_data_inversion_address_bit_index property, which defaults to 0 and corresponds to r[0].

- data_inversion_row_address_bit_select_enable : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_data_inv_row_add_bit_select_en input port of the BAP direct access interface. The BAP input port enables row address bit selection for write and expect data inversion.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- bira_enable : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_bira_en input port of the BAP direct access interface. The BAP port enables repair analysis for the memory BIST controller.

- check_repair_needed : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_check_repair_needed port of the BAP direct access interface. The BAP port specifies that the repair analysis status registers need to be examined after the BIST completes.

- preserve_fuse_register_values : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_preserve_fuse_register port of the BAP direct access interface. The BAP port specifies if the repair analysis registers preserve their values from a previous test step, which inhibits the loading of the BISR register values into the repair analysis registers when a new test step begins.

- ClockDomain (*domain_label*) { }

  The *domain_label* property specifies the BAP direct access interface connections, described within the repeatable ClockDomain wrapper, for specific memory BIST controller clock domains. If the BistAccessPort/DirectAccessOptions/direct_access_clock_source property is set to common, the *domain_label* property should be specified as "-". The control and status signals are common for all controllers regardless of their clock domain and a single BAP sequencer services all controllers. The clock connecting to the sys_clock input of the BAP direct access interface must be at least four times slower than the lowest memory BIST clock frequency.

  When direct_access_clock_source is specified as per_bist_clock_domain, *domain_label* should match the domain name specified for the controller with the MemoryBist/Controller/clock_domain_label property. The signal connections defined in the ClockDomain wrappers for this configuration are applicable to the controllers operating on the specified clock domain. There must be two or more MBIST clock domains to use this configuration, otherwise implementation becomes the "common" case previously described.

- ClockDomain/clock : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_clock input port of the BAP direct access interface. When this property is unspecified and the direct_access_clock_source is per_bist_clock_domain, the BAP input clock port is automatically connected to the memory BIST clock identified by *domain_label*.

- ClockDomain/reset : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_reset input port of the BAP direct access interface. The BAP input port asynchronously resets the direct access interface.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ClockDomain/test_init : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_test_init input port of the BAP direct access interface. The BAP input port asynchronously resets the controllers operating on the same clock domain.

- ClockDomain/test_start : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_test_start input port of the BAP direct access interface. The BAP input port launches the test execution.

- ClockDomain/test_done : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_test_done output port of the BAP direct access interface. The BAP input port indicates when all controllers, operating on the same clock domain, have completed the test execution.

- ClockDomain/test_pass : *port_pin_name* ;

  An optional property and string pair that specifies the system logic port or pin name that connects to the sys_test_pass output port of the BAP direct access interface. The BAP input port indicates the global pass/fail status of all controllers operating on the same clock domain.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TCKInjectionPoints

Specifies the locations in the design where a mux is inserted to inject TCK onto the functional clock trees for TCK mode implementation.

Usage

```
DftSpecification(module_name,id) {
  MemoryBist {
    TCKInjectionPoints {
      ClockMux (mux_id) {
        leaf_instance_name : instance_name ;
        node : node_name ;
      }
    }
  }
}
```

Description

The TCKInjectionPoints wrapper defines the locations where muxes are inserted to inject TCK. The TCK that is injected is the one used by the specified MemoryBist/ ijtag_host_interface.These muxes are controlled by the DFT signal “tck_select”, which is asserted as needed during patterns steps by setting the Patterns tck_clock_only property to “on”.

Arguments

- leaf_instance_name : *instance_name* ;

  A string property that specifies the leaf instance name of the mux. A prefix is added to the specified *instance_name* as follows:

  *<persistent_clock_cell_prefix>*_tck_mux_*instance_name*

  If this property is not specified, it defaults to:

  *<persistent_clock_cell_prefix>*_tck_mux_*<design_name>*_*<dft_spec_id>*_*<mux_id>*_inst

  The “dft_spec_id” string is the “id” specified in the DftSpecification wrapper, and the “mux_id” is the string specified in the ClockMux wrapper. Tessent Shell checks for name conflicts before instantiating the mux and, if needed, applies the uniquification suffix specified by the set_insertion_options command.

- node : *node_name* ;

  A required property that specifies the location of a port, pin or net name to be intercepted with a mux instance.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

OCC

The OCC DftSpecification syntax defines the configuration of the OCC to be generated and inserted into the design.

Usage

```
DftSpecification(module_name,id) {
  OCC {
    ijtag_host_interface           : host_name | none ;
    capture_trigger                : auto | shift_en | capture_en ;
    static_clock_control           : auto | off | internal | external |
                                     both ;
    force_clock_gater_te_tied_off  : on | off ;
    capture_window_size            : integer ; // default: 3
    internal_clock_gater           : on | off | auto ;
    shift_only_mode                : on | off | auto ;
    kill_clock_mode                : on | off | auto ;
    include_clocks_in_icl_model    : on | off | auto ;
    leaf_instance_name             : instance_name ;
    upstream_parent_occ            : auto | off | require | allow ;
                                 // "require" creates a child-mode OCC
    parent_mode                    : on | off ;
    Interface {
    }
    Connections {
    }
    Controller(id) {
    }
  }
}
```

Description

In modern designs, OCC circuits are commonly used to manage clocks during test. Such clock controllers can generate slow-speed or at-speed clock sequences under the control of ATPG process. Tessent OCC is an implementation of a clock controller created by Tessent Shell that has been designed to meet the requirements of scan test for ATPG, Logic BIST, EDT, and Low Pin Count Test.

The OCC follows the use_rtl_synchronizer_cell property to decide if a real or an RTL cell synchronizer cell is used. The OCC also follows the use_rtl_synchronizer_cell property to use a synchronizer cell with or without a reset port. The use of the asynchronous reset removes the need to add extra shift cycles in the parallel load simulations even when you have capture clocks which are slower than the shift clock.

Note

The Sync cell is instantiated with the leaf instance prefix tessent_persistent_cell_ltest_ntc_ where tessent_persistent_cell_ is specified using the persistent_cell_prefix property of the DftSpecification wrapper. You can use this special prefix to locate them in the design if you want to disable their notification during logic test Verilog simulations.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Refer to "The Standard OCC" in the *Tessent Scan and ATPG User's Manual* for more information.

Note

The standard OCC is the recommended OCC for use in hierarchical cores. For more information, see "Recommendations" in the *Tessent Scan and ATPG User's Manual*.

The standard OCC has the ability to inject the shift clock and allows controlling which clock pulse goes through based on the control flip-flops. The fast_clock input can be a free-running asynchronous clock. See "Fast Capture Mode Operation" in the *Tessent Scan and ATPG User's Manual* for a schematic.

Note

The tool sets the ICL module attribute tessent_use_in_dft_specification = "false" for the Tessent OCC. This prevents unnecessary warning messages during subsequent create_dft_specification calls for instances that are connected correctly. If the following are true:

If you inserted a Tessent OCC without an ijtag_host_node, tied the StaticExternalControls and therefore require create_dft_specification to create a TDR to source the StaticExternalControls, you must first override the value of the tessent_use_in_dft_specification ICL attribute using "set_attribute_value <*OCC ICL instance*> -name tessent_use_in_dft_specification -value auto".

Arguments

- ijtag_host_interface : *host_name* | none ;

  A property that specifies the host scan interface to which the clock controllers are connected. The default value is "none" which means that the static controls are primary inputs on the control. With the "none" value, you can use the Connections/ StaticExternalControls wrapper to specify where to connect them. If you are using IJTAG with all its advantages with respect to the test_setup automation, use the create_dft_specification -sri_sib_list switch with a value of "Occ" to have the DftSpecification created with an Sib ready to host the OCC controllers. You then point to it using "Sib(Occ)" as the ijtag_host_interface value.

- capture_trigger : auto | shift_en | capture_en ;

  A property that specifies whether shift_en or capture_en is used as the trigger to generate programmable capture pulses. The capture_en version is used when some internal test logic generates a separate capture_en signal, such as the LPCT or LBIST controller.

  The default is auto.

  Choose one from the following:

  auto — Automatically determines whether shift_en or capture_en is used as the trigger. When the OCC wrapper is used with an LpctType3 wrapper in the same insertion

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

pass (that is, the wrappers are present in the same DftSpecification) auto resolves to the capture_en value; otherwise it resolves to shift_en. In the TK/LBIST independent insertion flow, auto resolves to shift_en.

capture_en — The OCC controllers infer the capture_en port.

shift_en — The OCC controllers infer the shift_en port.

- static_clock_control : auto | off | internal | external | both ;

  A property that specifies whether capture clock generation is available outside of ATPG control.

  Note

  The Tessent Core Description (TCD) flow does not support static clock control mode. When adding the Tessent OCC for ATPG using the add_core_instances command, the OCC is configured into dynamic mode with clock controls and does not support external, static clock controls.

  To use static clock control mode for ATPG, you must manually configure the OCC by adding the clocks, updating and supplying the appropriate test procedures, and supplying the named capture procedures so that ATPG can understand how to control the clocks. This bypasses the automation provided when using the add_core_instances command.

  There is a dependency between this property and the StaticExternalControls wrapper's static_clock_control_mode property. By default the OCC wrapper's static_clock_control property defaults to auto; you must set this property to a value other than off or auto in order to generate the static clock control pins on the OCC module (except when LogicBIST is present, in which case auto resolves to external).

  Choose one from the following:

  auto — When both OCC and LogicBIST are included in the same insertion pass (that is, the wrappers are present in the same DftSpecification), auto resolves to external when LogicBIST is present; otherwise, it resolves to off.

  off — Capture clock generation is only under ATPG control, specifically static programmability.

  internal — Non-ATPG control capture clock generation is available through an IJTAG TDR in the OCC (static programmability).

  external — Non-ATPG control capture clock generation is available through OCC input pins.

  both — Combination of internal and external.

- force_clock_gater_te_tied_off : off | on ;

  A property that specifies that the test enable pins of clock gaters inside OCC are tied off. By default, the property is off, which results in connecting the test enable and functional enable pin of the same clock gater to the same internal enable signal. When you enable this

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

property, you may benefit from easier timing closure, but make sure that third party tools will leave the test enable pins inside OCC untouched.

- capture_window_size : *integer* ;

  A property that specifies the maximum possible number of clock pulses the OCC can generate. During pattern generation, the OCC can be configured to produce a smaller number of pulses than this specified value. The default is 3, and the limit is 64.

  Note

  Using a large capture window for OCC creation and using all the bits during pattern generation could result in lower test coverage (EDT Aborts) and/or higher pattern count.

- internal_clock_gater : on | off | auto ;

  A property that specifies if a child-mode OCC has the clock gater inside the OCC module or if it is to only output the enable signal for a clock gater already present at the base of the clock tree. The value can only be off when the OCC operates in child mode. By default, auto resolves to off when the clock_intercept_node property is empty and the clock_enable_pin property refers to the name of a pin.

- shift_only_mode : on | off | auto ;

  A property that specifies the addition of a new port on the OCC controller that is used to activate the shift only mode in the OCC. The default is auto.

  This property is "on" if you added the ext_ltest_en DFT signal using the add_dft_signals command. Otherwise, this property is "off".

  For more information, refer to "Shift Mode" in the *Tessent Scan and ATPG User's Manual*.

- kill_clock_mode : on | off | auto ;

  A property that specifies the addition of a new port on the OCC controller that activates the clock gater on the fast_clock path such that the clock gates off during functional mode. This means that the output of an OCC with kill_clock_en = 1 is held in an off state whenever it propagates the fast_clock input to the clock_out output port. This includes all scenarios when test_mode signal is low, including the special case of the capture phase of shift only mode (shift_only_mode = 1, and scan_en = 0).

  Turning off the functional clock of disabled OCCs can reduce toggle activity to the portion of the design not being tested and can speed up pattern simulation. The tool ignores the value of the kill_clock_en port when the OCC is active.

  The following truth table characterizes the behavior of the OCC in kill clock mode:

Table 13-13. Kill Clock Mode Truth Table

test_mode	kill_clock_en	shift_only_mode	scan_en	clock_out
0	0	0	X	fast_clock
0	0	1	0	fast_clock

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-13. Kill Clock Mode Truth Table (cont.)

test_mode	kill_clock_en	shift_only_mode	scan_en	clock_out
0	0	1	1	slow_clock
0	1	0	X	0
0	1	1	0	0
0	1	1	1	slow_clock
1	X	X	1	slow_clock
1	X	X	0	chopped_fast_clock / chopped_slow_clock

For more information including a schematic, refer to Kill Clock Mode in the *Tessent Scan and ATPG User's Manual*.

When the DFT signal occ_kill_clock_en is added, the auto value is interpreted as On, and the kill_clock_en port is automatically sourced by the occ_kill_clock_en DFT signal making it easy to activate this feature using the set_static_dft_signal_values command.

Tip

Use this property to re-use the clock gater inside the OCC during functional mode to turn off the functional clock. You can also use this property during hierarchical test mode. When running the internal logic test modes on a core, you can save power inside the other cores that are not running by turning off their clocks using the following command and switch:

```
set_static_dft_signal_values occ_kill_clock_en 1 \
    -instances inactive_core_instances
```

To achieve the same results during MemoryBIST patterns, you use the DftControlSettings wrapper.

- include_clocks_in_icl_model : on | off | auto ;

  An optional property that specifies whether the input and output clock ports as well as the optional TCK mux are to be described in the ICL model of the OCC. The tool inserts a TCK mux inside the OCC when the ijtag_host_interface property is not set to "none". You must describe the TCK mux in ICL if you want to use the tck_clock_only property in the patterns file and the TCK clock is injected with the OCC controller.

  The default for this property is auto. The use of the icl_extraction_port_trigger_list attribute in the fast_clock ICL port declaration ensures backward compatibility. Only if a trace from another ICL module reaches the clock_out port, is the fast_clock input traced during ICL extraction. The auto setting ensures that the TCK mux, which is present within the OCC when the ijtag_host_interface property is not set to "none", is visible within the ICL view and can thus be used to run MemoryBIST with TCK.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If you set the property to "on", the tool omits the icl_extraction_port_trigger_list from the fast_clock ICL port declaration.

- leaf_instance_name : *instance_name* ;

  A property that specifies the leaf instance name for the clock controller.

- upstream_parent_occ : auto | off | require | allow ;

  A property that specifies whether the OCC supports an active upstream parent-mode OCC to drive its fast clock input. The default is "auto". See Parent-Mode Operation, Child-Mode Operation, and Active Upstream Parent Mode in the *Tessent Scan and ATPG User's Manual* for schematics of the OCCs.

  auto — The value resolves to "allow" when an IJTAG interface is present or "off" when an IJTAG interface is not present.

  off — The OCC has no support for an upstream parent-mode OCC.

  require — The OCC operates in child mode and requires an upstream parent-mode standard OCC.

  A child-mode OCC does not include a multiplexer along the clock path to inject the shift clock. The child-mode OCC lets the clock go through when scan_en is high. When scan_en is low, the clock input pulses for a constant number of pulses and the child-mode OCC lets zero or more pulses go through based on the value of its control flip-flops. Typically, a parent-mode OCC is at the base of the clock sourcing the clock input of the child-mode OCC. See "Child-Mode OCC Logic Schematic" in the *Tessent Scan and ATPG User's Manual* for a schematic.

  allow — The OCC supports an active upstream parent-mode OCC to drive its fast clock input. You can use this value for a standard OCC or a parent-mode OCC.

  When you set the core instance parameter active_upstream_parent_occ to "on" to support active upstream parent-mode OCCs, you bypass the fast clock synchronizer cell and only the parent-mode OCC's synchronizer cell is in use. The core-level patterns use the fast clock from the parent-mode OCC, while the slow clock is supplied locally. The benefit is to simplify the timing closure of the shift mode because all shift clocks and EDT clocks derive from a local source.

- parent_mode : on | off ;

  A property that specifies whether to implement the parent-mode features in a standard OCC. The default is "off". When the value is "on" and the ijtag_host_interface property is "none", the tool creates the parent_mode port on the OCC interface. When the value is "on" and the ijtag_host_interface property is a host_name, the tool creates an IJTAG TDR.

  In parent mode, the OCC injects the shift clock at the base of the clock in shift mode. When scan_en is low, it lets a programmable constant set of clock pulses go through. The parent-mode OCC typically feeds the clock input of a child-mode OCC. It is the child-mode OCCs that enable the scan-based test pattern generation tool to control which clock pulses go through. See "Parent-Mode OCC Logic Schematic" in *Tessent Scan and ATPG User's Manual* for a schematic.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example shows the insertion of OCC modules configured to enable upstream parent-mode OCCs. See Parent-Mode Operation, Child-Mode Operation, and Active Upstream Parent Mode in the *Tessent Scan and ATPG User's Manual* for schematics of the OCCs.

```
set_context dft -rtl -design_id rtl2
read_cell_library ../prerequisites/techlib_adk.tnt/current/tessent/ \
                                                        adk.tcelllib
read_verilog ../data/mems/SYNC_1R1W_16x8.v -interface_only -
exclude_from_file_dictionary
read_design coreb -design_id rtl1
set_current_design coreb

set_dft_specification_requirements -logic_test on
add_clocks clkc -period 8ns

## DFT Signals
add_dft_signal memory_bypass_en tck_occ_en
add_dft_signal ltest_en int_ltest_en ext_ltest_en int_mode ext_mode
add_dft_signal scan_en test_clock edt_update -source_node {scan_en
test_clock edt_update}
add_dft_signal edt_clock shift_capture_clock -create_from_other_signals

check_design_rules
set spec [create_dft_spec -sri_sib_list {occ edt}]
read_config_data -in_wrapper $spec -from_string {
  Occ {
    ijtag_host_interface : Sib(occ);
    include_clocks_in_icl_model : on;
    upstream_parent_occ : allow;
    Controller(clka) {
      clock_intercept_node : clka;
    }
    Controller(clkb) {
       clock_intercept_node : clkb;
    }
    Controller(clkc) {
      clock_intercept_node : clkc;
    }
  }

  Edt {
    ijtag_host_interface : Sib(edt);
    Controller(c1) {
      longest_chain_range            : 10, 50;
      scan_chain_count               : 15;
      input_channel_count            : 2;
      output_channel_count           : 1;
    }
  }
}
report_config_data $spec
process_dft_specification
extract_icl
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set spec [create_patterns_spec]
process_patterns_specification
set_simulation_library_sources -v ../prerequisites/techlib_adk.tnt/ \
                        current/verilog/adk.v -y ../data/mems -extension v
run_testbench_simulations
write_design_import_script -use_relative_path_to . -replace
```

Example 2

The following example shows how to insert parent-mode OCC modules for each block:

```
read_config_data -in_wrapper $spec -from_string {
  Occ {
    ijtag_host_interface : Sib(occ);
    capture_window_size : 4;
    parent_mode : on;
    Controller(clka) {
      clock_intercept_node : clka_buf/A;
    }
    Controller(clkb) {
      clock_intercept_node : clkb_buf/A;
    }
    Controller(clkc) {
      clock_intercept_node : clkc_buf/A;
    }
  }

  Edt {
    ijtag_host_interface : Sib(edt);
    Controller(c1) {
      longest_chain_range                  : 10, 50;
      scan_chain_count                     : 40;
      input_channel_count                  : 2;
      output_channel_count                 : 1;
      connect_bscan_segments_to_lsb_chains : on;
      Connections {
        EdtChannelsIn(1) {
        }
        EdtChannelsIn(2) {
        }
        EdtChannelsOut(1) {
          Pipelinestage {
          }
        }
      }
    }
  }
}
```

Example 3

The parent-mode OCC operates as a standard OCC unless you configure it to function as a parent. In this example, the top-level OCC operates in parent mode and the cores enable the upstream parent-mode OCC during retargeting.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

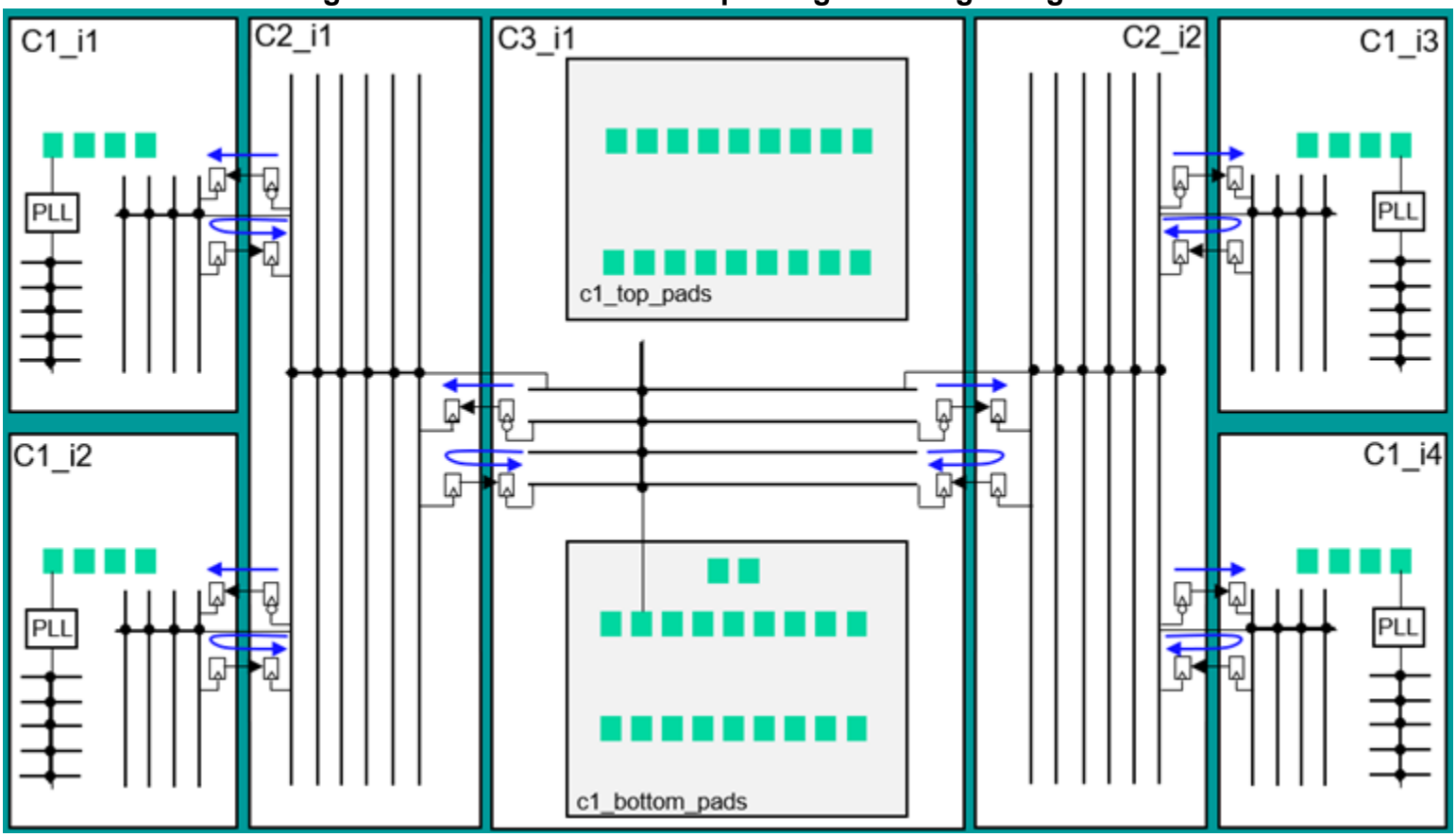
```
set_context patterns -scan_retargeting
read_cell_library ../prerequisites/techlib_adk.tnt/current/tessent/
adk.tcelllib
read_design corea -view graybox
read_design coreb -view graybox
read_design top
set_current_design top
add_core_instances -instances {corea_i1 coreb_i1} -mode int_mode

import_clocks
add_core_instances -inst [get_inst top_rtl2_tessent_occ_parent_*] \
                        -parameter {parent_mode on fast_capture_mode on}
set_static_dft_signal_values retargeting3_mode 1
//set_drc_handling R27 warning

set_core_instance_parameters -instances \
            coreb_i1/coreb_rtl2_tessent_occ_clka_inst -parameter_values \
            {active_upstream_parent_occ on}
set_core_instance_parameters -instances \
            coreb_i1/coreb_rtl2_tessent_occ_clkb_inst -parameter_values \
            {active_upstream_parent_occ on}
set_core_instance_parameters -instances \
            coreb_i1/coreb_rtl2_tessent_occ_clkc_inst -parameter_values \
            {active_upstream_parent_occ on}
//setenv SET_ACTIVE_UPSTREAM_PARENT_OCC_TO_ON 1

check_design_rules
report_procedures test_setup
read_patterns tsdb_outdir/logic_test_cores/corea_gate.logic_test_core/ \
           corea.atpg_mode_int_mode/corea_int_mode_transition.patdb
read_patterns tsdb_outdir/logic_test_cores/coreb_gate.logic_test_core/ \
           coreb.atpg_mode_int_mode/coreb_int_mode_transition.patdb
set_external_capture_options -pll_cycles 5 [lindex [get_timeplate_list] 0]
write_pattern top_retargeting3_parallel.v -parallel -v -replace -
param_list {SIM_TOP_NAME TB}
write_pattern top_retargeting3_serial.v   -serial   -v -replace -end 16 \
           -param_list {SIM_TOP_NAME TB}
```

Example 4

This example shows the DFT insertion process used on a hierarchical design that is implemented using the tiling design flow (TDF). Figure 13-77 shows a reference design composed of seven physical layout regions that are abutted to each other with only wire in the chip-level module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-77. Hierarchical Chip Using the Tiling Design Flow



The chip is symmetrically built, where two rows of the C1 modules are mirrored on both sides of the chip. A left- and right-mirrored copy of C2 are placed on either side of C3. All DFT ports are pre-created on each tile and pre-connected in the chip module. See the process_top_module_connections command for a way to automate this within the tool, and see the TopModuleConnections wrapper matching the example shown above.

In a tiling flow, the clock trees are typically localized to each tile and a leaf node of the localized clock tree is used to source the clock tree in the adjacent tiles, as shown in Figure 13-77. In this example, there are three levels of cascaded clock trees by the time the clock reaches the C1 tiles. The functional clocking is reused by the OCC controllers during the scan-based test modes.

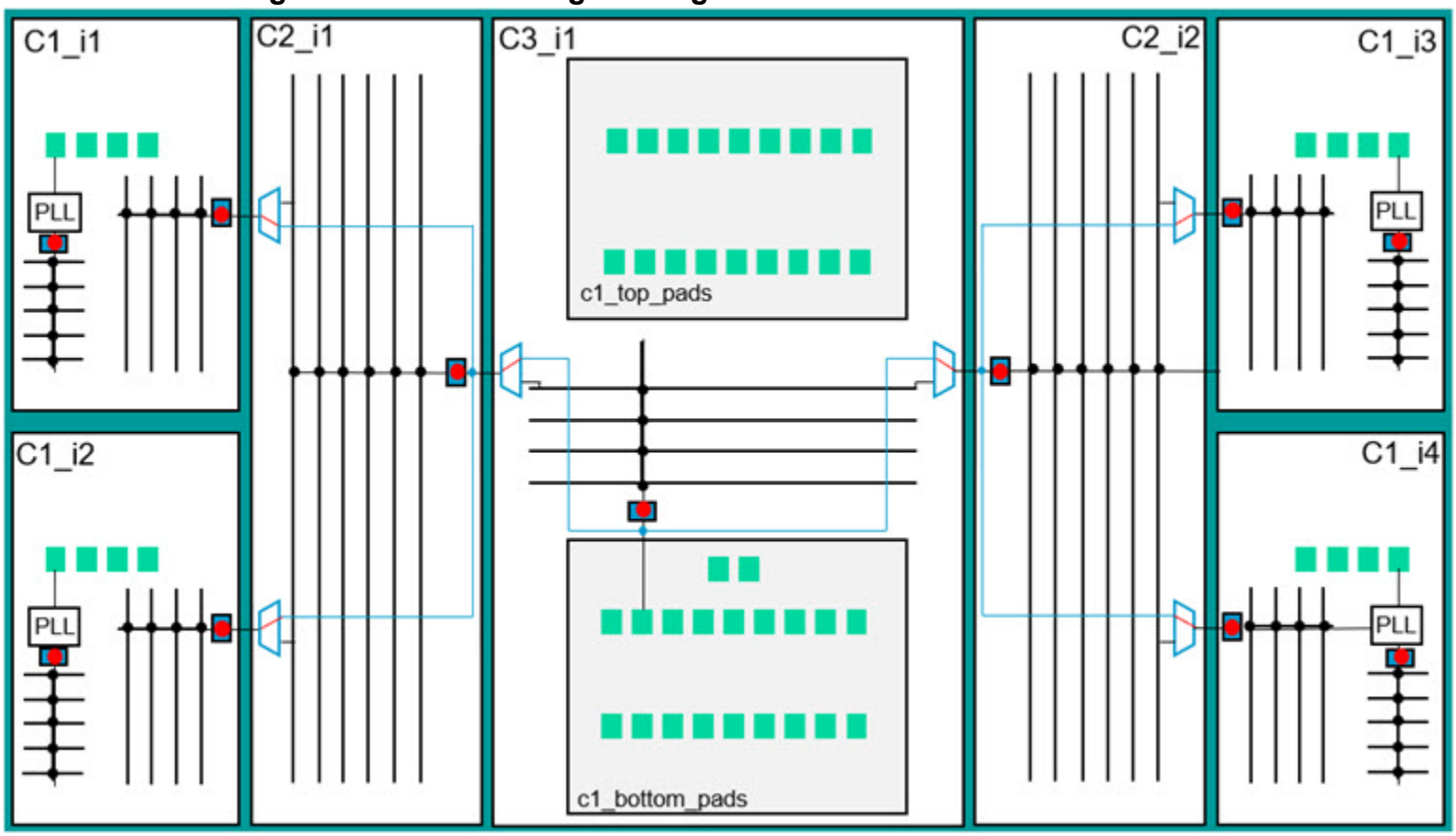
- The following examples show you how to insert the different parts of the flow into a tiled design:
  - Top-Level Ports — Example 1 in the process_top_module_connections command reference.
  - Boundary Scan — Example 1 in the HostBScanInterface wrapper section.
  - IJTAG — Example 3 in the IjtagNetwork wrapper section.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Memory BISR — Example 3 in the MemoryBISR/SecondaryHostChainInterface wrapper section.
- SSN — Example 3 in SSN wrapper section.

Each tile in the design is wrapped during scan insertion. An OCC controller is inserted in each tile to be used during their respective internal test modes. Figure 13-78 shows the location of the OCC controllers.

Figure 13-78. OCC Usage During the Internal Test Modes of All Tiles



Internal Test Modes

The red dots show that the OCCs are all active during the internal test of the tiles. If you want to run any combination of tiles in parallel, you must inject OCC bypass multiplexers in the C3 and C2 tiles, so that the OCCs in C1 and C2 can receive a free-running clock, even when the OCCs in C3 and C2 are busy gating the clock for their respective internal test modes. If you do not inject the multiplexer shown in Figure 13-78, you cannot run the internal TDF test mode of C1 in parallel with C2 and C3, nor will you be able to run the internal TDF test mode of C2 in parallel with C3. This condition is verified by the R27 DRC check. The easiest way to insert such OCC bypass multiplexers is to use the add_dft_clock_mux command, as shown in the code block below. Run the command in the DFT insertion pass where the OCCs are inserted. The add_dft_clock_mux command correctly connects to the specified -test_clock_source after the OCC did its interception of the same specified clock source. The int_ltest_en DFT signal

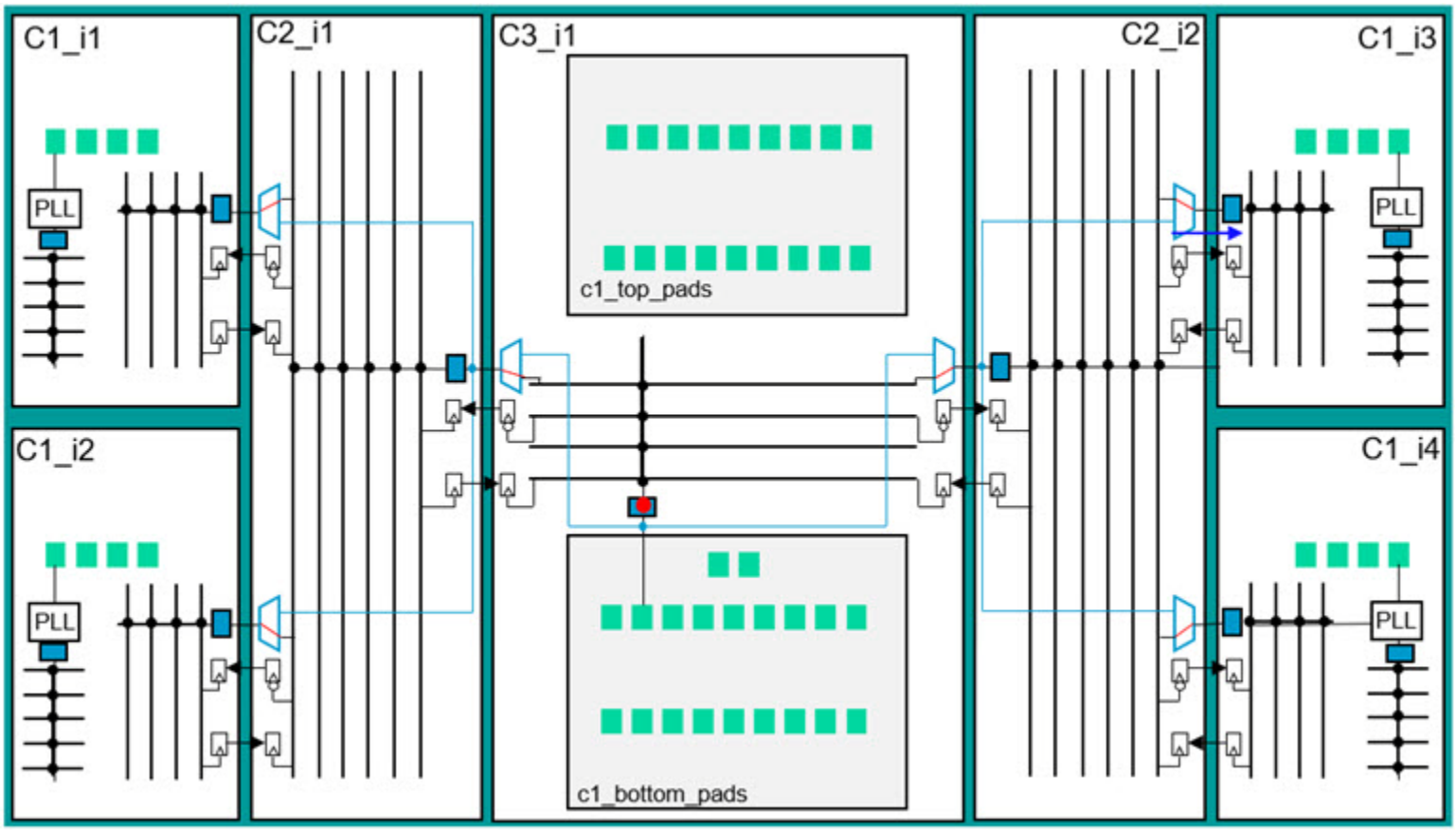
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

controls the select of the multiplexers. This DFT signal is automatically set high whenever a scan_mode of type internal is turned on, ensuring that the OCC bypass clock multiplexing is activated during the internal test modes.

External Test Modes

Figure 13-79 shows the active OCC when the tiles are placed in their external test mode.

Figure 13-79. OCC Usage During the External Test Modes of the Tiles



The OCC bypass clock multiplexers are returned to select the functional clock sources of each tile ensuring that the inter-tile logic paths are tested at-speed using their native functional clocking. The OCC in C3 is active during its external test mode because its clock source comes from a primary input of the chip, and there is no OCC at the top level to handle the gating of the clock. The OCC is promoted to the external test mode of C3 during scan insertion with the "add_scan_mode -include_elements" command, as shown in the dofile below.

C3 OCC Insertion

Below are the dofile commands used to insert the OCC into the C3 tile. The add_dft_clock_mux command injects the OCC bypass clock multiplexer.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_context dft -rtl -design_id rtl3
read_cell_library ../../library/cells/adk.tcelllib

read_design c3 -design_id rtl2
read_design c3_top_pads -design_id rtl1 -view ijtag_graybox
read_design c3_bottom_pads -design_id rtl1 -view ijtag_graybox
set_current_design c3

set_design_level physical_block

set_dft_specification_requirements -logic_test on -design_type tile

## DFT Signals
add_dft_signals int_edt_mode ext_edt_mode
add_dft_signals memory_bypass_en tck_occ_en
add_dft_signals ltest_en int_ltest_en ext_ltest_en
add_dft_signals ssn_en

add_dft_clock_mux clkc_to_left  -dft_signal int_ltest_en \
                                -test_clock_source clkc;
add_dft_clock_mux clkc_to_right -dft_signal int_ltest_en \
                                -test_clock_source clkc;

check_design_rules

set spec [create_dft_spec -sri_sib_list {occ edt ssn}]

# Prevents ports to be created in tiling flow and
# makes sure all DFT ports were properly added and
# connected in the die netlist.
set_config_value allow_port_creation_on_current_design -in $spec off


read_config_data -in_wrapper $spec -from_string {
  OCC {
    ijtag_host_interface : Sib(occ);
    include_clocks_in_icl_model : yes;
    Controller(clkc) {
       clock_intercept_node : clkc;
    }
  }
}

process_dft_specification
extract_icl -create_ijtag_graybox on
```

C3 Scan Chain Insertion

Below are the dofile commands to insert scan chains into C3. The OCC is promoted to the external scan mode explicitly using the “add_scan_mode -include_elements” command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
analyze_wrapper_cells

report_wrapper_cells

add_scan_mode int_edt_mode \
              -edt_instances c3_rtl3_tessent_edt_c1_int_inst
# Promote clkc OCC to external mode as there is no OCC outside the tile.
add_scan_mode ext_edt_mode \
              -edt_instances c3_rtl3_tessent_edt_c1_ext_inst \
              -include_elements [add_to_collection \
                   [get_scan_element -class wrapper] \
                   [get_scan_element *tessent_occ_clkc*]]

analyze_scan_chains

insert_test_logic
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Interface

This wrapper specifies the port names on the interface of the OCC module. When a particular property is not used, the tool default port naming is used.

Usage

```
DftSpecification(module_name,id) {
  OCC {
    Interface {
      scan_en          : port_name ; // default: scan_en
      capture_en       : port_name ; // default: capture_en
      slow_clock       : port_name ; // default: slow_clock
      fast_clock       : port_name ; // default: fast_clock
      clock            : port_name ; // defualt: clock
      clock_out        : port_name ; // default: clock_out
      clock_en_out     : port_name ; // default: clock_en_out
      scan_in          : port_name ; // default: scan_in
      scan_out         : port_name ; // default: scan_out
      clock_sequence   : port_name ; // default: clock_sequence[%d]
      pulse_to_align   : port_name ; // default: pulse_to_align[%d]
      IjtagScanInterface {
      }
      StaticExternalControls {
      }
    }
  }
}
```

Description

This wrapper specifies the port names on the interface of the OCC module. When a particular property is not used, the tool default port naming is used instead.

Arguments

- scan_en : *port_name* ;

  Scan enable port name. The default is scan_en.

- capture_en : *port_name* ;

  Optional capture enable port name, used only when capture_trigger is set to capture_en. The default is capture_en. The default connection for capture_en is tie-0.

  In LPCT usage, this tied pin is driven exclusively by the LPCT controller. In LBIST usage, this pin is multiplexed between LBIST capture enable during LBIST mode and inverted scan enable during ATPG mode.

- slow_clock : *port_name* ;

  Slow clock input. This port provides the shift clock during shift mode and clock source during slow capture mode. In LPCT or LBIST applications, this is connected to a free running clock. The default is slow_clock. This port is not present when the type property is set to child.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- fast_clock : *port_name* ;

  Fast clock input. This port is connected to the functional clock source and provides the source clock during fast capture mode. The default is fast_clock. This port is not present when the type property is set to child. See "Interface Port Naming Convention" in the *Tessent Scan and ATPG User's Manual* for more information.

- clock : *port_name* ;

  A clock input that only exist when the type property is set to "child". This clock carries the shift clock durign shift mode and the capture clock during capture mode. An OCC of type parent is typically used at the source of the clock feeding the OCC of type child.

- clock_out : *port_name* ;

  Test clock output pin that delivers shift clock during shift and programmable capture clock during capture. The default is clock_out. See "Interface Port Naming Convention" in the *Tessent Scan and ATPG User's Manual* for more information.

- clock_en_out : *port_name* ;

  A port that is only present for OCC of type "child". It is used to control the test_en port of a clock gater situated at the based of the clock tree. This output is typically only used when the internal_clock_gater property is off.

- scan_in : *port_name* ;

  Scan input for the shift register inside OCC. This pin is declared as a scan sub-chain input for scan insertion. The default is scan_in.

- scan_out : *port_name* ;

  Scan output for the shift register inside OCC. This pin is declared as a scan sub-chain output for scan insertion. The default is scan_out.

- clock_sequence : *port_name* ;

  Used for LogicBIST non-ATPG control of the OCC. The default is clock_sequence[%d].

- pulse_to_align : *port_name* ;

  An optional property that specifies a name for the pulse_to_align bus port on the OCC boundary that the tool creates when the static_clock_control's property value is "external" or "both", and the OCC/Controller/FrequencyRatio wrapper is specified. The default name is pulse_to_align[%d].

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

IjtagScanInterface

This wrapper specifies the names of ijtag interface ports of the OCC module. This wrapper is used only when the OCC uses IJTAG.

Usage

```
DftSpecification(module_name,id) {
  OCC {
    Interface {
      IjtagScanInterface {
        tck             : port_name       ; // default: ijtag_tck
        reset           : port_name       ; // default: ijtag_reset
        select          : port_name       ; // default: ijtag_sel
        capture_en      : port_name       ; // default: ijtag_ce
        shift_en        : port_name       ; // default: ijtag_se
        update_en       : port_name       ; // default: ijtag_ue
        scan_in         : port_name       ; // default: ijtag_si
        scan_out        : port_name       ; // default: ijtag_so
        reset_polarity  : active_polarity ; // legal: active_high
                                            // active_low
      }
    }
  }
}
```

Description

This wrapper specifies the names of ijtag interface ports of the OCC module. This wrapper is used only when the OCC uses IJTAG.

Arguments

- tck : *port_name* ;

  IJTAG control signal. The default is ijtag_tck.
- reset : *port_name* ;

  IJTAG control signal. The default is ijtag_reset.
- select : *port_name* ;

  IJTAG control signal. The default is ijtag_sel.
- capture_en : *port_name* ;

  IJTAG control signal. The default is ijtag_ce.
- shift_en : *port_name* ;

  IJTAG control signal. The default is ijtag_se.
- update_en : *port_name* ;

  IJTAG control signal. The default is ijtag_ue.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- scan_in : *port_name* ;

  Scan interface pins for the IJTAG scan path that are connected to the specified IJTAG host scan interface.The default is ijtag_si.

- scan_out : *port_name* ;

  Scan interface pins for the IJTAG scan path that are connected to the specified IJTAG host scan interface.The default is ijtag_so.

- reset_polarity : *active_polarity* ;

  Active state of the ijtag reset signal, to be used during IJTAG network insertion. This specifies only the active state inside the OCC. The default is active_low.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

StaticExternalControls

This wrapper specifies the names of static signals when ijtag is not used. When using IJTAG, these signals are available as TDR bits inside the OCC.

Usage

```
DftSpecification(module_name,id) {
  OCC {
    Interface {
      StaticExternalControls {
        test_mode                  : port_name ; //test_mode
        fast_capture_mode          : port_name ; //fast_capture_mode
        parent_mode                : port_name ; //parent_mode
        capture_cycle_width        : port_name ; //capture_cycle_width[%d]
        static_clock_control_mode  : port_name ;
                                                 //static_clock_control_mode
        shift_only_mode            : port_name ; //shift_only_mode
        kill_clock_en              : port_name ; //kill_clock_en
      }
    }
  }
}
```

Description

This wrapper specifies the names of static signals when ijtag is not used. When using IJTAG, these signals are available as TDR bits inside the OCC.

Arguments

- test_mode : *port_name* ;

  Signal to specify non-Functional mode. The default value is test_mode. This port is only created when the ijtag_host_interface property is set to “none”.

- fast_capture_mode : *port_name* ;

  Signal to choose between fast and slow capture modes. The source of the programmable clock output is the fast_clock OCC input when this signal is 1 or the slow_clock OCC input when this signal is 0. The default value is fast_capture_mode. The tool creates the port when the ijtag_host_interface property is “none”, and the OCC operates in standard or parent-mode (which means the OCC/upstream_parent_occ property is “allow” or “off”.)

- parent_mode : *port_name* ;

  An optional property to name the parent_mode port. The default name is parent_mode. The tool creates the port when the ijtag_host_interface property is “none” and the OCC operates in parent-mode, which means the OCC/parent_mode property is “on”.

- capture_cycle_width : *port_name* ;

  Provides pattern generation time configurability of the capture width size up to the maximum specified. This is a vector port with log2(capture_width_size) bits. The default value is “capture_cycle_width”. This port is only created when the ijtag_host_interface property is set to “none”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- static_clock_control_mode : *port_name* ;

  Control signal to decide whether the programmable clock sequence is under ATPG control or not. When this pin is 0, the clock sequence is shifted in as part of scan loading. When this pin is 1, the clock sequence is parallel loaded from the OCC clock sequence. The default value is static_clock_control_mode. This port is only created when the ijtag_host_interface property is set to “none”.

- shift_only_mode : *port_name* ;

  Signal used to enable the shift_only_mode port on the OCC controller. The default value is “shift_only_mode”. The port is only created when the OCC wrapper shift_only_mode property is set to or interpreted as “on”.

- kill_clock_en : *port_name* ;

  Signal used to enable the kill_clock_mode port on the OCC controller that is used to activate the clock gater on the fast_clock path such that the clock can be gated off during functional mode. The default value is “kill_clock_en”. The port is only created when the OCC wrapper kill_clock_mode property is set to or interpreted as “on”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Connections

This wrapper specifies the design port or pin to which the OCC pins should be connected. When specified, this globally overrides the default connection specified for this port type. These are the global specifications.

Usage

```
DftSpecification(module_name,id) {
  OCC {
    Connections {
      scan_en        : port_pin_name ; // Legal: DftSignal(scan_en)
                                       // OptionalDftSignal(scan_en)
      capture_en     : port_pin_name ; // default: 0
      slow_clock     : port_pin_name ;
                                 // Legal: DftSignal(shift_capture_clock)
                                 // OptionalDftSignal(shift_capture_clock)
      clock_sequence : port_pin_constant_name, ... ; // default: 0
      pulse_to_align : port_pin_constant_name, ... ; // default: 0
      StaticExternalControls {
      }
    }
  }
}
```

Description

This wrapper specifies the design port or pin to which the OCC pins should be connected. When specified, this globally overrides the default connection specified for this port type. These are the global specifications.

Arguments

- scan_en : *port_pin_name* | DftSignal(scan_en) | OptionalDftSignal(scan_en);

  An optional property used to specify a source node to connect the scan_en port to.

  The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the set_design_level command, is not “chip” and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and is created as an input port during insertion.

  When design level is “chip”, specified port names must already be connected to a pad cell. The connection is automatically made to the internal side of the input pad buffer. You can specify OptionalDftSignal(scan_en) or DftSignal(scan_en) as the value, and it is interpreted to request a connection to the DFT signal scan_en as it was added using the add_dft_signals command. When using OptionalDftsignal(scan_en), you do not get an error if the DFT signal scan_en was not added; a connection to the port called “scan_en” is made instead and the port is created if not present unless design level is equal to “chip” in which case you get an error. When the property is unspecified, its value defaults to OptionalDftSignal(scan_en).

  When you use SSN with a ScanHost wrapper, you must leave this property unspecified otherwise you will get an error. The scan_en port of the OCC will be sourced by the scan_en output port of the ScanHost node.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- capture_en : *port_pin_name* ;

  Optional capture enable port name, used only when capture_trigger is set to capture_en. The default is 0.

- slow_clock : *port_pin_name* ;

  An optional property used to specify a source node to connect the slow_clock port to.

  The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the set_design_level command, is not "chip" and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it is created as an input port during insertion.

  When design level is "chip", specified port names must already be connected to a pad cell. The connection is automatically made to the internal side of the input pad buffer. You can specify OptionalDftSignal(shift_capture_clock) or DftSignal(shift_capture_clock) as the value, and it is interpreted to request a connection to the DFT signal shift_capture_clock as it was added using the add_dft_signals command. When using OptionalDftsignal(shift_capture_clock), you do not get an error if the DFT signal shift_capture_clock was not added; a connection to the port called "slow_clock" is made instead and the port is created if not present unless design level is equal to "chip," in which case you get an error. When the property is unspecified, its value defaults to OptionalDftSignal(shift_capture_clock).

  When you use SSN with a ScanHost wrapper, you must leave this property unspecified otherwise you will get an error. The slow_clock port of the OCC will be sourced by the shift_capture_clock output port of the ScanHost node.

- clock_sequence : *port_pin_constant_name*, ...; // default: 0

  An optional property to use for LogicBIST non-ATPG control of the OCC. The default is 0. You should provide a sequence of bit values to connect for a constant value. For example, when capture_window_size is equal to 3, provide a value in the form '111', '110', '001'.

- pulse_to_align : *port_pin_constant_name*, ... ; // default: 0

  An optional property to use for LogicBIST non-ATPG control of the OCC. The tool makes this connection during OCC insertion when the static_clock_control's property value is "external" or "both", and the OCC/Controller/FrequencyRatio wrapper is specified. The default is 0, which means making a constant connection. You should provide a sequence of bit values to connect for a constant value. You can override this connection for a particular OCC using the OCC/Controller/Connections wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

StaticExternalControls

This wrapper specifies the names of static signals when IJTAG is not used. When using IJTAG, these signals are available as TDR bits inside the OCC.

Usage

```
DftSpecification(module_name,id) {
  OCC {
    Connections {
      StaticExternalControls {
        test_mode                 : port_pin_name ; // Legal: 0 1
        fast_capture_mode         : port_pin_name ; // Legal: 0 1
        parent_mode               : port_pin_name ; // Legal: 0 1
        capture_cycle_width       : port_pin_name ; // Legal: 0 1, ...
        static_clock_control_mode : port_pin_constant_name ; // Legal:
                                                              // 0 1
        shift_only_mode           : port_name ;
                          // default: DftSignalOrTiedLow(ext_ltest_en)
        kill_clock_en             : port_name ;
                              // default: DftSignal(occ_kill_clock_en)
      }
    }
  }
}
```

Description

This wrapper specifies the names of static signals when the ijtag_host_interface property is set to “none”. When the ijtag_host_interface property refers to an IJTAG host node, these signals are available as TDR bits inside the OCC.

Arguments

- test_mode : *port_pin_name* | 0 | 1 ;

  Signal that is used to activate the OCC. When this signal is 0, the OCC is not activated and simply lets the fast_clock/clock input go through unaltered. The default is 0.

- fast_capture_mode : *port_pin_name* | 0 | 1 ;

  Signal to choose between fast and slow capture modes. The source of the programmable clock output is the fast_clock OCC input when this signal is 1 or the slow_clock OCC input when this signal is 0. The default value is 0. This port is not present for OCC of type “child”.

- parent_mode : *port_pin_name* | 0 | 1 ;

  A signal connection that activates or deactivates parent mode. When equal to 0, the OCC operates in standard mode. When equal to 1, the OCC operates in parent mode. The default value is 0.

- capture_cycle_width : *port_pin_name* | 0 | 1, ... ;

  Provides pattern generation time configurability of the capture width size up to the maximum specified. Constant throughout the test, that is ATPG cannot change it on a per pattern basis.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- static_clock_control_mode : *port_pin_constant_name* | 0 | 1 ;

  Control signal to decide whether the programmable clock sequence is under ATPG control or not. When this pin is 0, the clock sequence is shifted in as part of scan loading. When this pin is 1, the clock sequence is parallel loaded from the OCC external_clock_control pins. The default is 0. This port only exists when the static_clock_control property is set to external or both.

- shift_only_mode : *port_name* ;

  DFT signal used to activate the shift_only_mode. The default connection is DftSignalOrTiedLow(ext_ltest_en).

- kill_clock_en : *port_name* ;

  DFT signal used to activate the kill_clock_mode such that the clock gater on the fast_clock path such that the clock can be gated off during functional mode. The default is DftSignal(occ_kill_clock_en).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Controller

This wrapper specifies the connections to the controller.

Usage

```
DftSpecification(module_name,id) {
  OCC {
    Controller(id) {
      clock_intercept_nodes      : port_pin_name, ... ;
      clock_port_count           : integer | auto ; // default: auto
      FrequencyRatio( 2 | 4 | 8 ) {
        clock_intercept_nodes    : port_pin_name, ... ;
        clock_port_count         : integer | auto ; // default: auto
      }
      clock_enable_pin           : pin_name ;
      clock_enable_pin_polarity  : active_high | active_low | auto ;
      parent_instance            : instance_name ;
      capture_window_size        : integer ; // default: 3
      leaf_instance_name         : instance_name ;
      internal_clock_gater       : on | off | auto ;
      shift_only_mode            : on | off | auto ;
      kill_clock_mode            : on | off | auto ;
      upstream_parent_occ        : auto | off | require | allow ;
                                 // "require" creates a child-mode OCC
      parent_mode                : on | off ;
      Connections {
      }
    }
  }
}
```

Description

This wrapper specifies the connections to the controller.

Arguments

- clock_intercept_nodes : *port_pin_name*, ... ;

  A property that specifies a list of design ports or pins that attach to the OCC clock output pin. The value of the “has_functional_source” attribute of the ports or pins specified by this property impacts the connectivity of the OCC fast clock port. See the fast_clocks property description in “OCC/Controller/Connections”.

- clock_port_count : *integer* | auto ;

  A property that specifies the number of clock ports. Use the clock_port_count property in the no-insertion flow only. The default value is auto, which resolves to the number of nodes specified in the clock_intercept_nodes list.

- FrequencyRatio(2 | 4 | 8) {}

  A wrapper that specifies a divided-clock ratio of 2, 4, or 8 for a list of divided clocks and their *clock_intercept_nodes*. You can define more than one FrequencyRatio wrapper inside

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

the Connections wrapper. Define a wrapper for each divided clock ratio and its list of clock and intercept nodes.

- clock_enable_pin : *pin_name*;

  A property that specifies the test_en pin of a clock gater situated at the base of the clock tree which the OCC of type child uses to control the clock during scan-based test pattern generation. This property can only be used when the type property is set to "child". When the clock_enable_pin property is used and the clock_intercept_node property is left blank, the internal_clock_gater is inferred to off when it has a value of "auto".

- clock_enable_pin_polarity : active_high | active_low | auto;

  This property controls the active polarity of the clock_en_port. You only need to specify this property when you are creating the OCC without doing the insertion process_dft_specification is used with the -no_insertion switch) and the clock_enable_pin is not pointing to the test_en pin of an integrated clock gater cell. When it is pointing to the test_en pin of an integrated clock gater cell, the active polarity is automatically extracted from the cell information.

- parent_instance : *instance_name* ;

  This property specifies the design instance in which the OCC is instantiated. By default, the OCC is placed at the same hierarchical level as the clock_intercept_node.

- capture_window_size: *integer* ;

  A property that specifies the maximum possible number of clock pulses the OCC can generate. During pattern generation, the OCC can be configured to produce a smaller number of pulses than this specified value. The default is 3.

- leaf_instance_name : *instance_name* ;

  This property inherits its value from the OCC wrapper leaf_instance_name property if not specified locally.

- internal_clock_gater : on | off | auto;

  This property inherits its values from the ../internal_clock_gater property when it is not specified locally. Its meaning is identical to how it is described in the ../internal_clock_gater property section.

- shift_only_mode : on | off | auto;

  This property inherits its value from the ../shift_only_mode property when it is not specified locally.

- kill_clock_mode : on | off | auto;

  This property inherits its value from the ../kill_clock_mode property when it is not specified locally.

- upstream_parent_occ : auto | off | require | allow ;

  A property that specifies whether the OCC supports an active upstream parent-mode OCC to drive its fast clock input. The default is "auto".

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

auto — The value resolves to “allow” when an IJTAG interface is present or “off” when an IJTAG interface is not present.

off — The OCC has no support for an upstream parent-mode OCC.

require — The OCC operates in child mode and requires an upstream parent-mode standard OCC.

A child-mode OCC does not include a multiplexer along the clock path to inject the shift clock. The child-mode OCC lets the clock go through when scan_en is high. When scan_en is low, the clock input pulses for a constant number of pulses and the child-mode OCC lets zero or more pulses go through based on the value of its control flip-flops. Typically, a parent-mode OCC is at the base of the clock sourcing the clock input of the child-mode OCC. See “Child-Mode OCC Logic Schematic” in the *Tessent Scan and ATPG User’s Manual* for a schematic.

allow — The OCC supports an active upstream parent-mode OCC to drive its fast clock input. You can use this value for a standard OCC or a parent-mode OCC.

When you set the core instance parameter active_upstream_parent_occ to “on” to support active upstream parent-mode OCCs, you bypass the fast clock synchronizer cell and only the parent-mode OCC’s synchronizer cell is in use. The core-level patterns use the fast clock from the parent-mode OCC, while the slow clock is supplied locally. The benefit is to simplify the timing closure of the shift mode because all shift clocks and EDT clocks derive from a local source.

- parent_mode : on | off;

  This property inherits its value from the ../parent_mode property when it is not specified locally.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Connections

This wrapper specifies the design port or pin to which the OCC pins should be connected.

Usage

```
DftSpecification(module_name,id) {
  OCC {
    Controller(id) {
      Connections {
        scan_en        : port_pin_name ; // DftSignal(scan_en)
                                         // OptionalDftSignal(scan_en)
        capture_en     : port_pin_name ; // default: 0
        slow_clock     : port_pin_name ;
                                 // DftSignal(shift_capture_clock)
                                 // OptionalDftSignal(shift_capture_clock)
        fast_clocks    : port_pin_name, ... ;
        FrequencyRatio( 2 | 4 | 8 ) {
          fast_clocks  : port_pin_name, ... ;
        }
        clock          : port_pin_name ;
        clock_sequence : port_pin_constant_name, ... ; // default: 0
        pulse_to_align : port_pin_constant_name, ... ; // default: 0
        StaticExternalControls {
        }
      }
    }
  }
}
```

Description

This wrapper specifies the design port or pin to which the OCC pins should be connected.

Arguments

- scan_en : *port_pin_name* ;

  An optional property used to specify a source node to connect the scan_en port to.

  The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the set_design_level command, is not “chip” and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it is created as an input port during insertion.

  When design level is “chip”, specified port names must already be connected to a pad cell. The connection is automatically made to the internal side of the input pad buffer. You can specify OptionalDftSignal(scan_en) or DftSignal(scan_en) as the value, and it is interpreted to request a connection to the DFT signal scan_en as it was added using the add_dft_signals command. When using OptionalDftsignal(scan_en), you do not get an error if the DFT signal scan_en was not added; a connection to the port called “scan_en” is made instead and the port is created if not present unless design level is equal to “chip,” in which case you get an error.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When the property is unspecified, its value is inherited from the ../Connections/scan_en property which itself defaults to OptionalDftSignal(scan_en) when unspecified.

When you use SSN with a ScanHost wrapper, you must leave this property unspecified otherwise you will get an error. The scan_en port of the OCC will be sourced by the scan_en output port of the ScanHost node.

- capture_en : *port_pin_name* ;

  Optional capture enable port name, used only when capture_trigger is set to capture_en. The default is 0.

- slow_clock : *port_pin_name* ;

  An optional property used to specify a source node to connect the shift_capture_clock port to.

  The node name can be a port or a pin name. The node must already exist in the design with the exception of one case. If the design level, as specified by the set_design_level command, is not "chip" and the name corresponds to the name of a scalar port, it is allowed to not exist in the design and it is created as an input port during insertion.

  When design level is "chip", specified port names must already be connected to a pad cell. The connection is automatically made to the internal side of the input pad buffer. You can specify OptionalDftSignal(shift_capture_clock) or DftSignal(shift_capture_clock) as the value, and it is interpreted to request a connection to the DFT signal shift_capture_clock as it was added using the add_dft_signals command. When using OptionalDftsignal(shift_capture_clock), you do not get an error if the DFT signal shift_capture_clock was not added; a connection to the port called "slow_clock" is made instead and the port is created if not present unless design level is equal to "chip," in which case you get an error.

  When the property is unspecified, its value is inherited from the ../Connections/shift_capture_clock property which itself defaults to OptionalDftSignal(shift_capture_clock) when unspecified.

  When you use SSN with a ScanHost wrapper, you must leave this property unspecified otherwise you will get an error. The slow_clock port of the OCC will be sourced by the shift_capture_clock output port of the ScanHost node.

- fast_clocks : *port_pin_name*, ... ;

  This property specifies a list of connections for the fast_clock input port of the OCC.

Note

You can use a "-" in your list to skip a position for those cases with more than one (N) clock_intercept_nodes and one or more (M) fast_clocks (M<N) at a given frequency. See "Synchronous DftSpecification Definition Rules" in the *Tessent Scan and ATPG User's Manual* for details.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When making these connections, the tool considers the value of the has_functional_source attribute of the ports or pins specified by the clock_intercept_node, and the value of the fast_clock property. There are four cases:

- Case 1 — The specified clock_intercept_node has a functional source, meaning its "has_functional_source" attribute set to true, and the fast_clock property is not specified. The OCC intercepts the specified clock_intercept_node so that the specified port/pin drives the OCC fast_clock port and the OCC clock_output drives all the sinks previously driven by the specified pin.
- Case 2 — The port or pin specified by clock_intercept_node has no functional source and the fast_clock property is specified. The OCC is inserted such that its fast_clock pin is connected to the port/pin specified by the fast_clock property and its clock_out pin is connected to the port/pin specified by the clock_intercept_node property.
- Case 3 — The specified clock_intercept_node has its has_functional_source attribute set to true, and the fast_clock property is specified. The OCC is inserted so that its fast_clock pin is connected to the port/pin specified by the fast_clock property and the OCC clock_out pin drives the logic previously connected to the port/pin specified by the clock_intercept_node property.
- Case 4 — The clock_intercept_node port does not have a functional source and fast_clock is not specified. In this case, the tool reports an error.

- FrequencyRatio(2 | 4 | 8) {}

  A wrapper that specifies a divided-clock ratio of 2, 4, or 8 for a list of divided clocks and their *clock_intercept_nodes*. You can define more than one FrequencyRatio wrapper inside the Connections wrapper. Define a wrapper for each divided clock ratio and its list of clock and intercept nodes.

- clock : *port_pin_name* ;

  This property is used to specify a connection for the clock input port of the OCC which only exists on OCC of type "child". This port is connected to a functional clock source and provides the source clock during both shift and capture mode. Used only when the clock_intercept_node property points to a pin that has no functional source (specifically, its has_functional_source attribute is false).

- clock_sequence : *port_pin_constant_name*, ... ;

  You can override the ../../Connections/clock_sequence connection for a particular OCC using this property.

- pulse_to_align : *port_pin_constant_name*, ... ; // default: 0

  You can override the ../../Connections/pulse_to_align connection for a particular OCC using this property.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

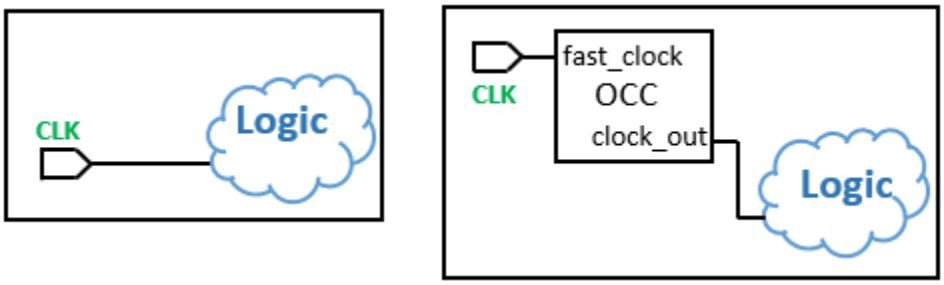
This example shows the case when the specified clock_intercept_node has a functional source and the fast_clock option is not specified for the OCC controller.

The OCC/Controller property clock_intercept_node specifies the CLK port (highlighted in green). The OCC/Controller/Connections fast_clock property is not used.

```
DftSpecification(design1, gates)  {
  OCC {
      Controller(NX1) {
         clock_intercept_node: CLK;
      }
  }
}
```

Figure 13-80 shows a design before and after OCC insertion. The OCC is inserted so that port specified by clock_intercept_node, CLK (highlighted in green), is connected to the OCC fast_clock port and the OCC clock_out port is connected to the logic previously connected to CLK.

Figure 13-80. Case 1 — clock_intercept_node Has Functional Source and fast_clock Not Specified



Example 2

This example shows case 2 when the specified controller_intercept_node does not have a functional source and the fast_clock option is specified for the OCC controller. One example for this case is when you are using a MUX outside the OCC to switch between the functional clock and the test clock.

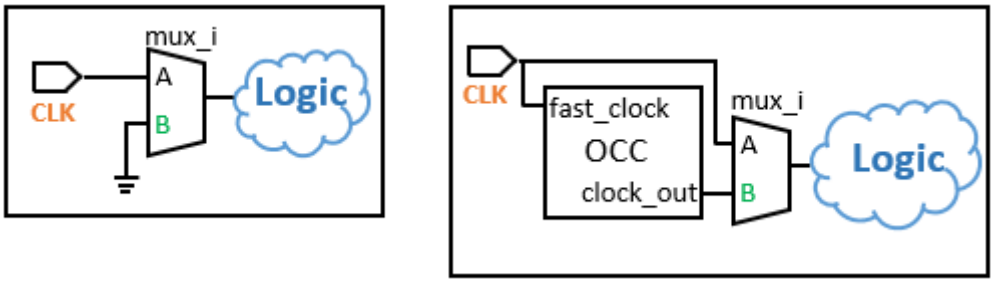
The specification shows that the clock_intercept_node is a pin on MUX, mux_i/B (highlighted in green), which is tied low. It also shows that the OCC/Controller/Connections fast_clock property is set to CLK (highlighted in orange).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(design1, gates)  {
  OCC {
    Controller(NX1) {
      clock_intercept_node: mux_i/B;
      Connections {
        fast_clock : CLK;
      }
    }
  }
}
```

Figure 13-81 shows that, before OCC insertion, CLK was connected to mux_i/A and mux_i/B is tied low. After insertion, the CLK port (highlighted in orange) is still connected to mux_i/A. It is also connected to the OCC fast_clock port. The OCC clock_out pin is connected to mux_i/B (highlighted in green).

Figure 13-81. No Functional Source and fast_clock Specified



Note

An error is reported if the clock_intercept_node does not have a functional source and fast_clock is not specified.

Example 3

This example shows the case when the specified clock_intercept_node has a functional source and the fast_clock option is specified for the OCC controller.

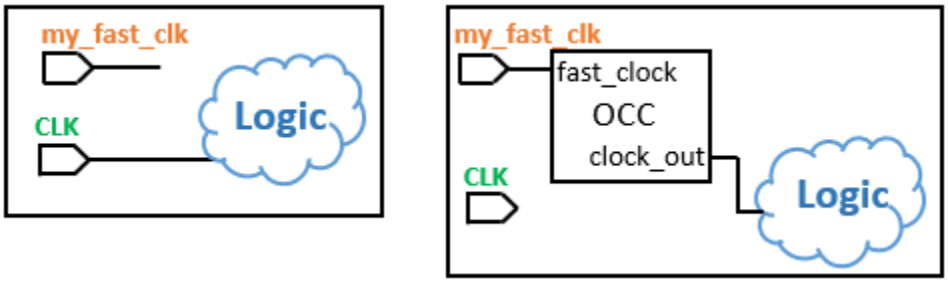
The OCC/Controller property clock_intercept_node specifies the CLK port (highlighted in green). The OCC/Controller/Connections fast_clock property specifies the my_fast_clock port (highlighted in orange).

```
DftSpecification(design1, gates)  {
  OCC {
    Controller(NX1) {
      clock_intercept_node: CLK;
      Connections {
        fast_clock : my_fast_clock;
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-82 shows that before OCC insertion the clock_intercept_node, CLK (highlighted in green) is connected to logic. It also shows the fast_clock specified port, my_fast_clock (highlighted in orange). After OCC insertion my_fast_clock is connected to the fast_clock input of the OCC and the clock_out port of the OCC drives the logic previously connected directly to CLK.

Figure 13-82. Case 3 - Functional Source and fast_clock Specified



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

StaticExternalControls

This wrapper specifies the names of static signals when ijtag is not used.

Usage

```
DftSpecification(module_name,id) {
  OCC {
    Controller(id) {
      Connections {
        StaticExternalControls {
          test_mode                 : port_pin_name | 0 | 1 ;
          fast_capture_mode         : port_pin_name | 0 | 1 ;
          parent_mode               : port_pin_name | 0 | 1 ;
          capture_cycle_width       : port_pin_name | 0 | 1, ... ;
          static_clock_control_mode : port_pin_constant_name | 0 | 1 ;
          shift_only_mode           : port_name;
                                // default: DftSignalOrTiedLow(ext_ltest_en)
          kill_clock_en             : port_name ;
                                    // default: DftSignal(occ_kill_clock_en)
        }
      }
    }
  }
}
```

Description

This wrapper specifies the names of static signals when ijtag is not used.

Arguments

- test_mode : *port_pin_name* ;

  This property inherits its value from the ../../Connections/StaticExternalControls/test_mode property when unspecified. It has the exact same usage as described in the ../../Connections/StaticExternalControls/test_mode section. You specify it here if you want a different connection for this controller.

- fast_capture_mode : *port_pin_name* ;

  This property inherits its value from the ../../Connections/StaticExternalControls/fast_capture_mode property when unspecified. It has the exact same usage as described in the ../../Connections/StaticExternalControls/fast_capture_mode section. You specify it here if you want a different connection for this controller.

- parent_mode : *port_pin_name* ;

  This property inherits its value from the ../../Connections/StaticExternalControls/parent_mode property when unspecified. It has the exact same usage as described in the ../../Connections/StaticExternalControls/parent_mode section. You specify it here if you want a different connection for this controller.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- capture_cycle_width : *port_pin_name*, ... ;

  This property inherits its value from the ../../Connections/StaticExternalControls/capture_cycle_width property when unspecified. It has the exact same usage as described in the ../../Connections/StaticExternalControls/capture_cycle_width section. You specify it here if you want a different connection for this controller.

- static_clock_control_mode : *port_pin_constant_name* ;

  This property inherits its value from the ../../Connections/StaticExternalControls/static_clock_control_mode property when unspecified. It has the exact same usage as described in the ../../Connections/StaticExternalControls/static_clock_control_mode section. You specify it here if you want a different connection for this controller.

- shift_only_mode : port_name ;

  This property inherits its value from the ../../Connections/StaticExternalControls/shift_only_mode property and has the exact same usage. You specify it here if you want a different connection for this controller. The default is DftSignalOrTiedLow(ext_ltest_en).

- kill_clock_en : *port_name* ;

  This property inherits its value from the ../../Connections/StaticExternalControls/kill_clock_en property and has the exact same usage. You specify it here if you want a different connection for this controller. The default DftSignal(occ_kill_clock_en).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SSN

A wrapper that specifies the Streaming Scan Network to be built and, optionally, inserted into the design.

Usage

```
DftSpecification(module_name,id) {
  SSN {
    DefaultChildConfiguration {
    }
    ijtag_host_interface : host_node_leaf_name; // default: Sib(ssn)
    scan_host_sourcing_dynamic_dft_signals : scan_host_id | none;
    clock_sources_with_no_local_occs : port_pin_name, ...;
    ScanHost(id) {
      // Only instantiate the ScanHost wrapper here if you do not want
      // a parallel bus datapath (Streaming-Through-IJTAG access only).
      // The ScanHost wrapper is typically added inside the Datapath
      // wrapper so that it supports a parallel SSN bus.
    }
    Datapath(id) {
    }
  }
}
```

Description

Use this wrapper to define one or more Streaming Scan Network (SSN) data paths and their associated nodes. Typically, you use a single Datapath wrapper at the lower levels of the chip. Under certain circumstances, however, you may need to define more than one Datapath wrapper at the higher level for different sections of the chip.

When SSN is present, you must provide an IJTAG network, which the tool uses to program the ScanHost nodes.

Planning the SSN architecture is relatively easy and does not require that you know the number of cores you will run in parallel in advance, and their respective channel count as the SSN can supply an infinite amount of channel data with as little as one bus data pair. However, the wider the SSN datapath, the shorter the test time for a given SSN bus data rate.

Follow these steps to architect your SSN:

1. Using your high-level chip floor plan, evaluate where your GPIOs that you can use for test are located in the chip.

   It is best to have a localized group of GPIOs that you use as the bus data input and another equal size localized group of GPIOs that you use as the bus data output.

2. Evaluate the speed at which you can shift through the GPIOs.

   As shown in Figure 13-88 and Figure 13-89 in the BusFrequencyDivider wrapper description, the critical path is the loop timing path from clock-in to data-out. When

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

using an OutputPipeline stage that is on its mini-clock tree, you can easily achieve a data rate of 200 MHz at the I/Os.

3. Evaluate the rate you can operate the SSN bus inside the chip when on a synchronous clock tree region.

   The SSN bus is a synchronous data protocol that typically uses no negative edge flip-flops. It is typically easy to operate at 400 MHz within a synchronous clock tree region, which means that the internal bus width you need to route across the chip is typically half the width of your I/O bus while maintaining a consistent throughput.

4. Plan the synchronous clock tree regions for the SSN bus clock.

   You have lots of flexibility here. You can use a single hierarchical clock tree for the SSN bus clock in the entire chip at one extreme or use localized clock trees that are not balanced across physical regions at the other extreme. You can also use an architecture that is closer to your functional modes and creates hierarchical clock tree regions for groups of localized physical regions. When the SSN bus crosses from one synchronous clock tree region to the next, you use a BusFrequencyDivider node at the source and a BusFrequencyMultiplier node at the destination to synchronize the data path. How much you need to slow down the data rate and widen the data bus is a function of the maximum skew you need to tolerate between the synchronous clock tree regions and is explained in detail in the "Using the BusFrequencyDivider to Cross CTS Regions" section of the BusFrequencyDivider wrapper description.

5. Plan how you will route the SSN bus to reach all the physical regions.

   Again, use your floor plan and your synchronous clock tree grouping to plan the distribution of the SSN bus. Routing a straight bus from one physical region to the next is very cheap when planned correctly. Avoid needed top-level routing resources to make the connection by making sure the pins on one physical block align well with the pins on the next physical block.

Refer to Example 1 to see how the SSN datapath is inserted inside a block and how it is designed to connect between repeated instances of the block and to other block types above and below it in Example 2.

Refer to Example 3 to see how to build the SSN datapaths when you are using a tiling physical design flow, or even when you are using near abutment of cores and want to minimize the routing area at the top level and maintain your SSN clock tree local to each core.

See "Example 3" on page 4529 in the ScanHost wrapper description to see how to see the use of the scan_host_sourcing_dynamic_dft_signals property when inserting ScanHost nodes to drive the scan signals on a legacy block that was not build with SSN inside it.

By default, ScanHost nodes have a parallel bus datapath and an IJTAG interface. It is possible to have an SSN with no parallel bus datapath. This limits access to the ScanHost nodes to the single-bit IJTAG interface. Example 4 demonstrates how to build an SSH with no parallel SSN bus.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

You can use the Streaming-Through-IJTAG feature as a more flexible replacement for LPCT-2 in devices with only a JTAG TAP interface. You are no longer limited to only being able to drive a single EDT controller with a single channel pair when using the IEEE 1149.1 protocol.

Debug Using IJTAG

The bus register in and out of the ScanHost node is accessible through the IJTAG network, as shown in Figure 13-106 on page 4489. Those registers are exercised automatically during the ICL verify pattern. This IJTAG access mode is used for debug in combination with the SSN continuity test. You can use the feature by creating IJTAG patterns from a PDL that iReads the bus register values explicitly, and then run it at an intermediate point during the continuity test.

Instantiated Nodes Section in the SSN Instrument Dictionary

The SSN instrument dictionary contains a section called “InstantiatedNodes”. This section includes the leaf wrapper name for all SSN nodes specified in this wrapper as keys, where the value is a collection handle that points to the associated instance that the tool created during insertion (such as during process_dft_specificaton). You can use this information in a process_dft_specification.post_insertion callback to access the SSN instantiated nodes with the script. (See “register_callback” on page 1586 for more information about the callback procedure.) The following command fetches the dictionary into a variable:

```
set InstantiatedNodesDict [get_instrument_dictionary mentor::ssn::DftSpecification
InstantiatedNodes]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For example, assume the following DftSpecification/SSN wrapper has been processed:

```
DftSpecification(mod,rtl) {
  SSN {
    ijtag_host_interface : Sib(ssn);
    Datapath(1) {
      output_bus_width : 4;
      Pipeline(1) {
      }
      ScanHost(1) {
        OnChipCompareMode {
          present : on;
        }
        ChainGroup(edt) {
          output_chain_count : same_as_input_chain_count;
        }
        Connections {
          ChainGroup(edt) {
          }
        }
      }
      Pipeline(2) {
      }
    }
  }
}
```

The following command:

get_instrument_dictionary mentor::ssn::DftSpecification InstantiatedNodes ScanHost(1)

returns a collection with the instance of the ScanHost module.

Arguments

- ijtag_host_interface : *host_node_leaf_name* ;

  This property specifies the IJTAG network node where the tool attaches the SSN nodes to the IJTAG ports. The default is Sib(ssn).

  The value you specify is inherited downward by all the nodes in the Datapath wrappers. The IJTAG ScanInterface for each node is subsequently daisy chained below the *node_id*. You can, however, specify different IJTAG host nodes for different SSN nodes. You can also control the ordering of the daisy chaining using the ijtag_connection_order property within each node. See the ijtag_connection_order property description in the BusFrequencyDivider, BusFrequencyMultiplier, Multiplexer, OutputPipeline, Pipeline, Receiver1xPipeline, and ScanHost wrappers for more information on how to control the ordering and the location of the IJTAG attach points to the IJTAG network.

  Use the following formats to specify the *host_node_leaf_name* value:

  - Sib(*id*)
  - Tap(*id*)/HostIjtag(*id*)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- HostScanInterface(*id*)
- ScanMux(*id*)/Input(0|1)

The *id* must match the leaf name of a wrapper in the IjtagNetwork wrapper.

You can create the IjtagNetwork with a Sib node that the tool will use for the SSN node by using the -sri_sib_list switch with the create_dft_specification command. For example, use -sri_sib_list {ssn edt occ} when inserting the logic test hardware.

- scan_host_sourcing_dynamic_dft_signals : *scan_host_id* | none;

  A property that is mandatory when the SSN wrapper includes more than one ScanHost wrapper. One of the ScanHost nodes is used for the current-level scan logic, including the EDT and OCC inserted at the current level, as well as sourcing the scan_en, edt_update, edt_clock, and shift_capture_clock dynamic DFT signals. The other ScanHost nodes are used to connect to the scan signals on the child blocks instantiated below the current design but not already equipped with the SSN logic. In that case, those child blocks typically have a test_clock port instead of having distinct edt_clock and shift_capture_clock ports, and this is fully supported for those extra ScanHost nodes. "Example 3" on page 4529 in the ScanHost wrapper description shows how to perform the connection to the child blocks.

  Leaving this property unspecified means that the dynamic DFT signals are sourced from the single ScanHost node. Setting this property to "none" does not map the dynamic DFT signals to any ScanHost node.

- clock_sources_with_no_local_occs : *port_pin_name*, ...;

  A property used to specify one or more clock ports or pins where you want to inject a clock multiplexer such that the shift_capture_clock output of the local SSN ScanHost can source the clocks during shift mode and during low speed capture cycles. Normally, this is done automatically within the On-chip Clock Controllers (OCCs), but if you are inserting the ScanHost inside a physical block that is not wrapped, you typically do not have an OCC on input clock ports. You do not have a local OCC if your clock comes from the output of a child physical block, as in "corea" within the first discussion figure in "How to Handle Clocks Sourced by Embedded PLLs During Logic Test" in the *Tessent Shell User's Manual*. In either case, the local ScanHost still needs to source the shift clock, which is why you must use this property to insert the multiplexer.

Examples

Example 1

The following example shows the insertion of a SSN inside a block with the provision to make it easy to connect with the surrounding blocks at the next level up as shown in Example 2.

At the insertion level, only one Datapath wrapper is present even though the network has multiplexers to pickup nodes from the blocks below and above it. The ScanHost wrapper description describes how the tool associates the two EDT controllers to the ScanHost node. Additionally, the description illustrates how you use the DFT signals to multiplex the EDT controller (used in internal mode) onto the same ScanHost node used in external mode.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-83 shows the SSN inserted into the block. The IjtagNetwork is omitted for simplicity. The Datapath flow is from the bottom node to the top node within the wrapper as follows:

- The bottom ExtraOutputPath wrapper is used to branch out the clock and data to the bottom block.
- The bottom Multiplexer is used to insert the data from the bottom block back into the data path.
- The ExtraOutputPath inside the ScanHost wrapper is to branch out the data output of the ScanHost node through a Pipeline node to the top block.
- The top Multiplexer is used to insert the data from the top block through a Pipeline node back into the data path.

Notice that the Connections wrapper inside the Datapath wrapper only includes the bus_clock_out property. By default, the clock output port is not created. The example block, however, is to be used in a tiling physical flow that requires the clock output port to be sent the to the block on the left. The other three properties use their default values and are omitted from the example.

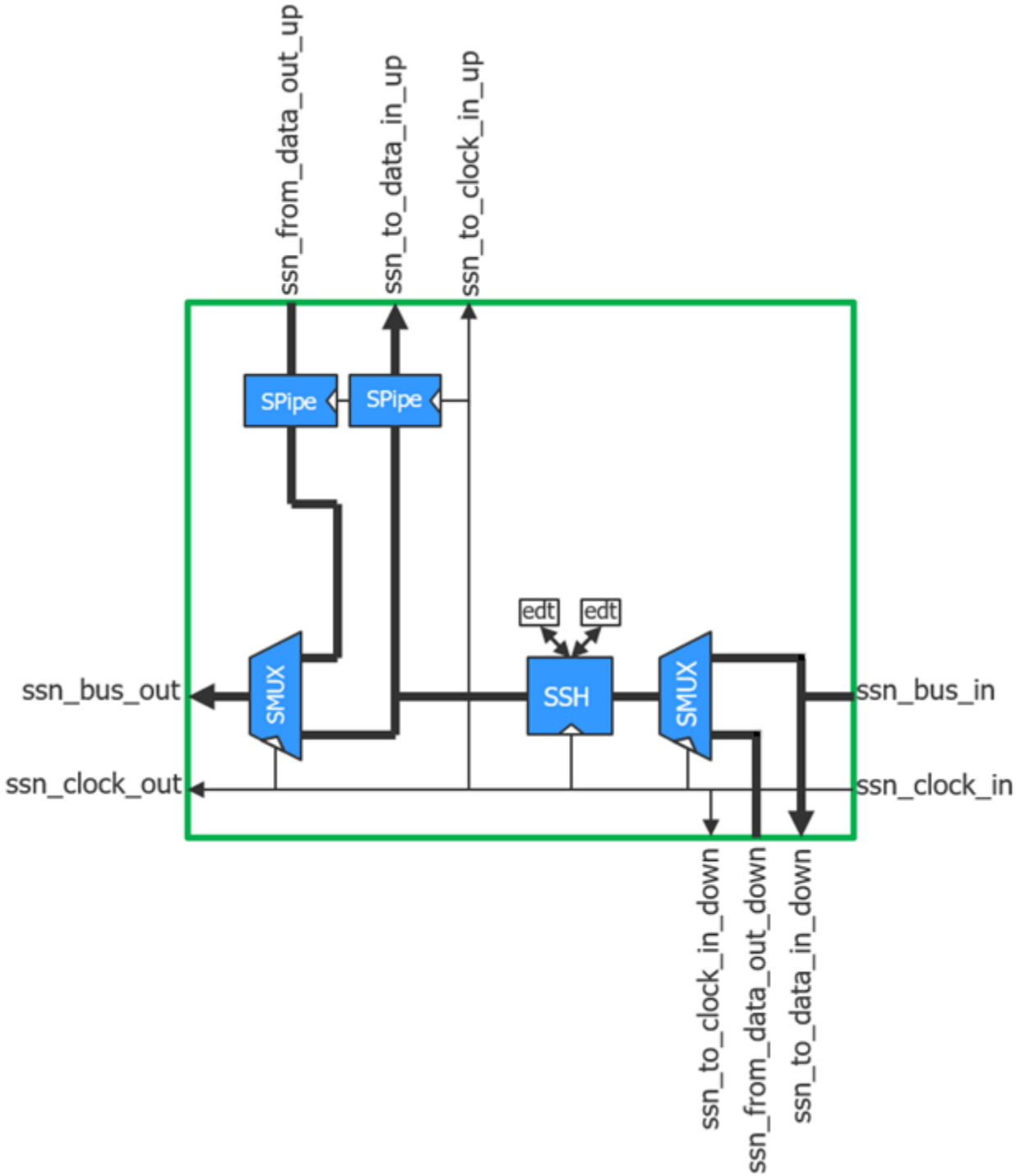
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set spec [create_dft_specification -sri_sib_list {ssn edt}]

read_config_data -in $spec -from_string {
  SSN {
    ijtag_host_interface : Sib(ssn);
    Datapath(1) {
      Connections {
        bus_clock_out : ssn_clock_out;
    }
      output_bus_width : 32;
      Multiplexer(up) {
        Connections {
          secondary_bus_data_in : ssn_from_data_out_up[%d];
        }
        Pipeline(from_up) {
        }
      }
      ScanHost(1) {
        ExtraOutputPath {
          Connections {
            bus_data_out  : ssn_to_data_in_up[%d];
            bus_clock_out : ssn_to_clock_in_up;
          }
          Pipeline(to_up) {
          }
        }
      }
      Multiplexer(down) {
        Connections {
          secondary_bus_data_in : ssn_from_data_out_down[%d];
        }
      }
      ExtraOutputPath {
        Connections {
          bus_data_out  : ssn_to_data_in_down[31:0];
          bus_clock_out : ssn_clock_in_down;
        }
      }
    }
  }
  Edt {
    ijtag_host_interface : Sib(edt);
    Controller(int) {
      longest_chain_range  : 100,200;
      scan_chain_count     : 800;
      input_channel_count  : 16;
      output_channel_count : 16;
      Connections {
        mode_enables       : DftSignal(int_edt_mode);
      }
    }
    Controller(ext) {
      longest_chain_range  : 100,200;
      scan_chain_count     : 40;
      input_channel_count  : 4;
      output_channel_count : 2;
      Connections {
        mode_enables       : DftSignal(ext_edt_mode);
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
            }
        }
    }
}
```

Figure 13-83. SSN Example One



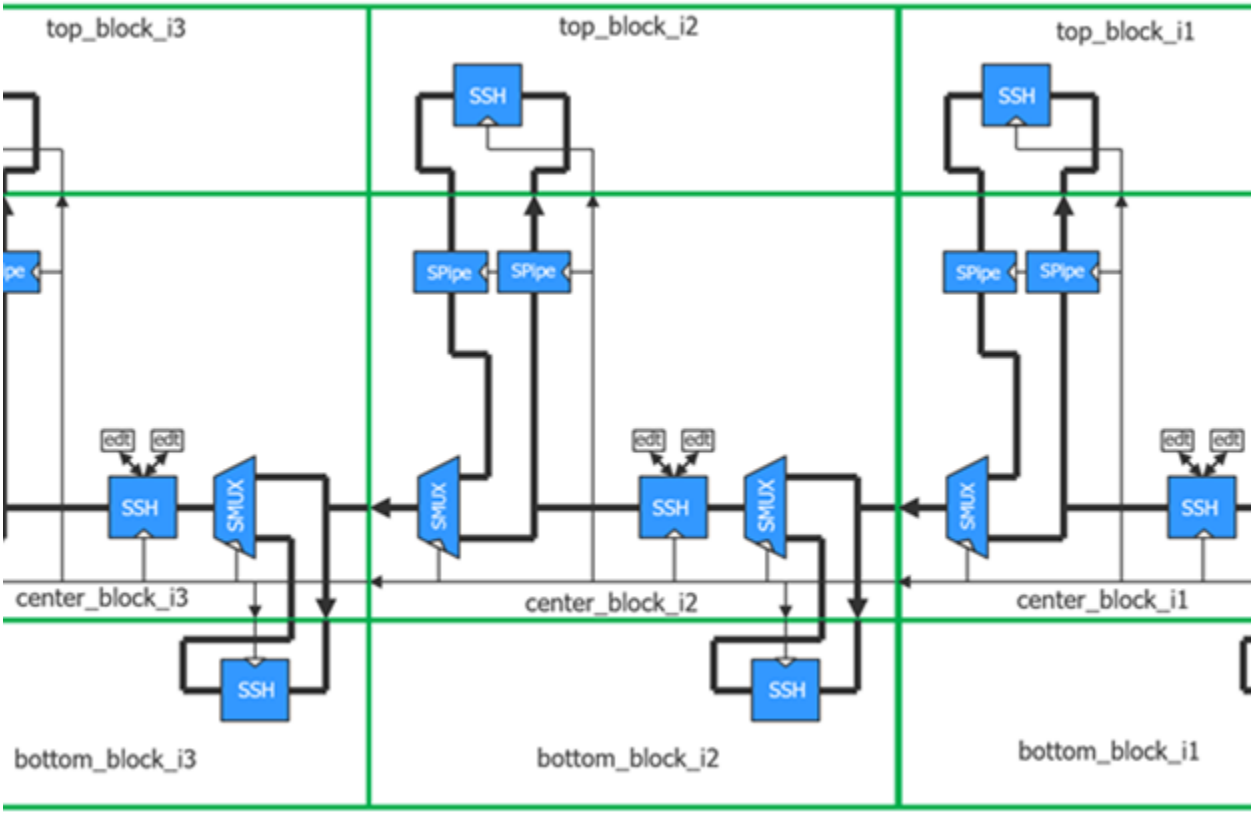
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

This example shows how the block created in Example 1 is connected to the surrounding blocks using a tiling methodology. Typically in a tiling flow, the ports used by SSN in Example 1 would have been pre-created manually ahead of time, and the connections in the top level would have also been made manually. There would be no need to run process_dft_specification at this level, and instead, only the extract_icl command would be used to extract the manually created connection and describe them in ICL. This example shows how complex datapath connections can be made automatically.

The ijtag_host_interface property is not used in this example because the Datapath wrappers only include DesignInstance wrappers. The IjtagNetwork wrapper handles the IJTAG network connections for the child blocks. The SSN data flow in a Datapath wrapper is from bottom to top. The DesignInstance(center_block_i3) is at the top; this means this DesignInstance is the one closer to the data output presented in Figure 13-84.

Figure 13-84. SSN Example Two With Connecting Tiles



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

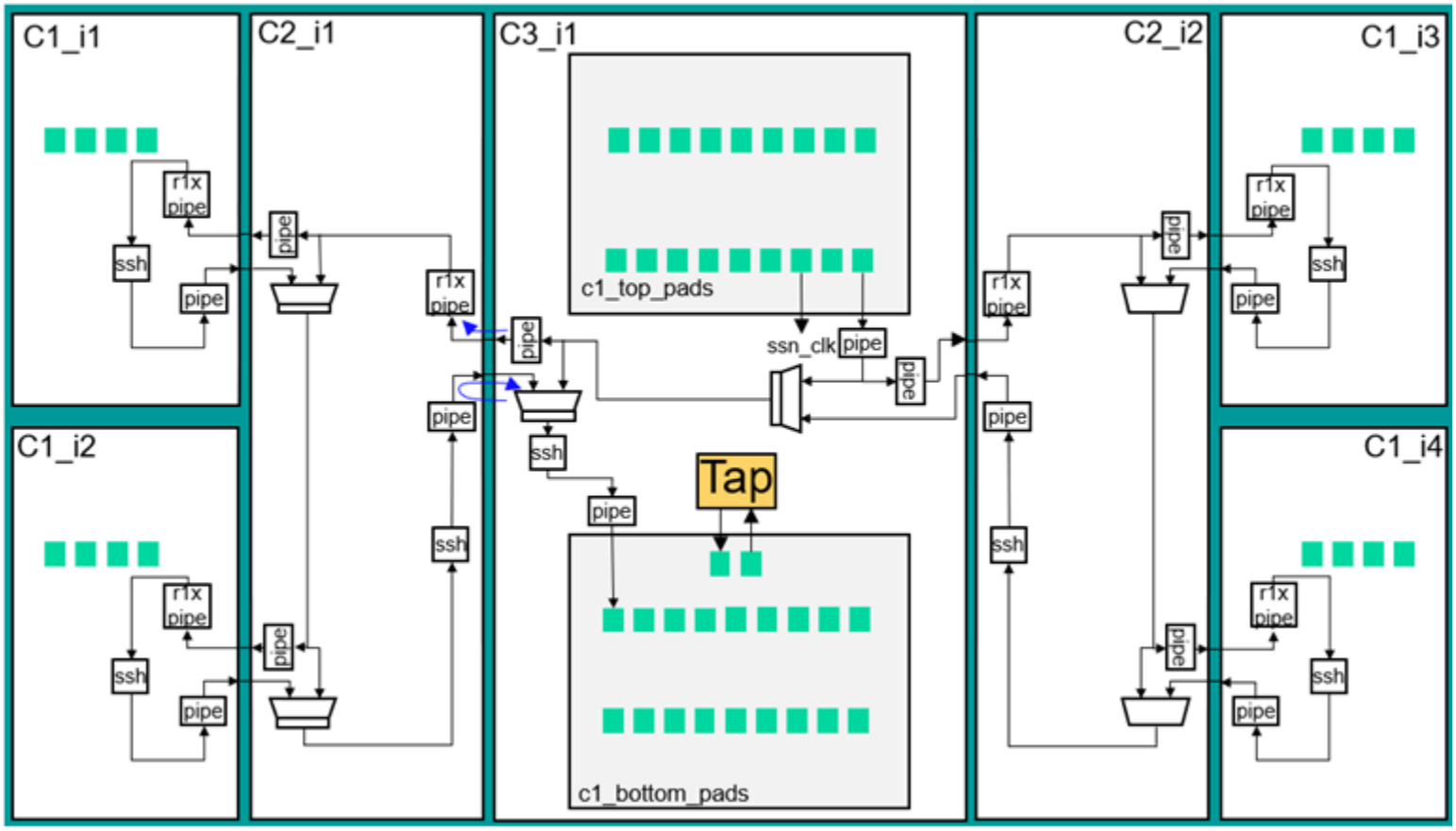
```
set spec [add_config_element DftSpecification(top,rtl2)]

read_config_data -in $spec -from_string {
  SSN {
    Datapath(main) {
      output_bus_width : 32;
      DesignInstance(center_block_i3) {
      }
      DesignInstance(center_block_i2) {
      }
      DesignInstance(center_block_i1) {
      }
    }
    Datapath(up1) {
      output_bus_width : 32;
      Connections {
        bus_clock_in : center_block_i1/ssn_to_clock_in_up;
        bus_data_in  : center_block_i1/ssn_to_data_in_up[%d];
        bus_data_out : center_block_i1/ssn_from_data_out_up[%d];
      }
      DesignInstance(top_block_i1) {
      }
    }
    Datapath(up2) {
      ...
    }
    Datapath(up3) {
      ...
    }
    Datapath(down1) {
      ...
    }

    Datapath(down2) {
      ...
    }
    Datapath(down3) {
      ...
    }
  }
}
```

Example 3

This example shows how to insert the SSN network in the tile-based design that is introduced in Example 4 of the OCC wrapper. From that example, you can follow the links to see how the Boundary Scan, IJTAG, and memory BISR parts are inserted. Just as with the functional clocking, boundary scan clock, IJTAG clock, and memory BISR clock, the SSN clocking is implemented with localized SSN clock trees, where a leaf of the localized clock tree is used to source the SSN clock tree of the adjacent tiles. Figure 13-85 shows the SSN network for the example design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-85. SSN Network in a Tiled Design



The boundary scan cells within the C3 tiles are equipped with auxiliary input and output logic to enable sharing the functional pins with the SSN bus. Refer to Example 1 of EmbeddedBoundaryScan/HostBScanInterface for the instructions on how to configure the boundary scan cells with the necessary auxiliary data logic.

C1

Below is an excerpt for the dofile to insert the SSN network in the C1 tile. There is a forward timing interface between C2 and C1, and a loop timing interface on the return path. In this example, the SSN clock trees are localized to each tile, requiring three clock trees in series to reach the C1 tiles on the outside.

The use of a Receiver1xPipeline node as the first node ensures we have a proper posedge-to-negedge forward timing path between C2 to C1. If your design had many more cascaded levels of SSN clock trees and the localized clock tree within C1 was large, you may need to use a Fifo node with input_retimed set to on instead of the Receiver1xPipeline, as you may collect too much duty cycle degradation to meet setup and hold on the forward timing interface. When the level of cascaded clock tree is not too high and the localized clock tree in the leaf tile is small enough, the use of a Receiver1xPipeline is adequate.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
read_design c1 -design_id rtl2
set_current_design c1

set_dft_specification_requirements -logic_test on

## DFT Signals
add_dft_signals int_edt_mode ext_edt_mode
add_dft_signals memory_bypass_en tck_occ_en
add_dft_signals ltest_en int_ltest_en ext_ltest_en
# To allow isolating the tile to tile testing from the PIs/POs of the
# device as those are tested during boundary scan test.
# Keeps SSN using internal capture mode
add_dft_signals bscan_input_isolation_enable output_pad_disable

check_design_rules

set spec [create_dft_spec -sri_sib_list {occ edt ssn}]
# Prevents ports to be created in tiling flow and
# makes sure all DFT ports were properly added and
# connected in the die netlist.
set_config_value allow_port_creation_on_current_design -in $spec off
read_config_data -in_wrapper $spec -from_string {
  SSN {
    ijtag_host_interface : Sib(ssn);
    DataPath(1) {
      output_bus_width :4;
      Pipeline(1) {
      }
      ScanHost(1) {
        OnChipCompareMode {
          present: on;
        }
      }
      Receiver1xPipeline(in) {
      }
    }
  }
}

process_dft_specification

extract_icl -create_ijtag_graybox on
```

C2

Below is an excerpt of the dofile to insert the SSN network within the C2 tile. Figure 13-85 shows that the SSN network enters the C2 tile from the center and provides two SSN host ports to service the SSN network within the C1 tiles on the outside. The host port from the C1 tile is constructed with a Multiplexer wrapper and an ExtraOutputPath wrapper in the previous node with a Pipeline node within it. A Receiver1xPipeline is used as the start of the network to build a proper posedge-to-negedge forward timing interface from the C3 tile. The pipeline(1) wrapper terminates the SSN network within the C2 tile and provides a full SSN bus clock period in the loop timing interface back to the C3 tile.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
read_design c2 -design_id rtl2
set_current_design c2

set_dft_specification_requirements -logic_test on

## DFT Signals
add_dft_signals int_edt_mode ext_edt_mode
add_dft_signals memory_bypass_en tck_occ_en
add_dft_signals ltest_en int_ltest_en ext_ltest_en

check_design_rules

set spec [create_dft_spec -sri_sib_list {occ edt ssn}]

# Prevents ports to be created in tiling flow and
# makes sure all DFT ports were properly added and
# connected in the die netlist
set_config_value allow_port_creation_on_current_design -in $spec off
read_config_data -in_wrapper $spec -from_string {
  SSN {
    ijtag_host_interface : Sib(ssn);
    DataPath(1) {
      output_bus_width :4;
      Pipeline(1) {
      }
      ScanHost(1) {
      }
      Multiplexer (from_r2) {
        Connections {
          secondary_bus_data_in : r2_ssn_from_bus_data_out[3:0];
        }
      }
      Multiplexer (from_r1) {
        Connections {
          secondary_bus_data_in : r1_ssn_from_bus_data_out[3:0];
        }
        ExtraOutputPath {
          Connections {
            bus_clock_out : r2_ssn_to_bus_clock;
            bus_data_out  : r2_ssn_to_bus_data_in[3:0];
          }
          pipeline (r2) {
          }
        }
      }
      Receiver1xPipeline (in) {
        ExtraOutputPath {
          Connections {
            bus_clock_out : r1_ssn_to_bus_clock;
            bus_data_out  : r1_ssn_to_bus_data_in[3:0];
          }
          pipeline (r1) {
          }
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

C3

Below is an excerpt from the dofile to insert the SSN network within the C3 tile, as shown in Figure 13-85. The SSN datapath starts from and ends on the functional GPIOs. The boundary scan cells were equipped with auxiliary input and output logic during the boundary scan insertion phase, as described in Example 1 of EmbeddedBoundaryScan/HostBScanInterface. The tool automatically maps the specified connections of the SSN bus that point to the GPIO port to pins on the c3_top_pads and c3_bottom_pads modules using the built-in auxiliary data logic information stored in the TSDB.

```
read_design c3 -design_id rtl2
read_design c3_top_pads -design_id rtl1 -view ijtag_graybox
read_design c3_bottom_pads -design_id rtl1 -view ijtag_graybox
set_current_design c3

set_dft_specification_requirements -logic_test on -design_type tile

## DFT Signals
add_dft_signals int_edt_mode ext_edt_mode
add_dft_signals memory_bypass_en tck_occ_en
add_dft_signals ltest_en int_ltest_en ext_ltest_en
add_dft_signals ssn_en
check_design_rules

set spec [create_dft_spec -sri_sib_list {occ edt ssn}]

# Prevents ports to be created in tiling flow and
# makes sure all DFT ports were properly added and
# connected in the die netlist.
set_config_value allow_port_creation_on_current_design -in $spec off
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
read_config_data -in_wrapper $spec -from_string {
  SSN {
    ijtag_host_interface : Sib(ssn);
    DataPath(1) {
      output_bus_width :4;
      Connections {
        bus_clock_in : core_data_in[4];
        bus_data_in  : core_data_in[3:0];
        bus_data_out : core_data_out[3:0];
      }
      Pipeline(1) {
      }
      ScanHost(1) {
      }
      Multiplexer (from_left) {
        Connections {
          secondary_bus_data_in : left_ssn_from_bus_data_out[3:0];
        }
      }
      Multiplexer (from_right) {
        Connections {
          secondary_bus_data_in : right_ssn_from_bus_data_out[3:0];
        }
        ExtraOutputPath {
          Connections {
            bus_clock_out : left_ssn_to_bus_clock;
            bus_data_out  : left_ssn_to_bus_data_in[3:0];
          }
          pipeline (left) {
          }
        }
      }
      pipeline (in) {
        ExtraOutputPath {
          Connections {
            bus_clock_out : right_ssn_to_bus_clock;
            bus_data_out  : right_ssn_to_bus_data_in[3:0];
          }
          pipeline (right) {
          }
        }
      }
    }
  }
}
process_dft_specification
extract_icl -create_ijtag_graybox on
```

Example 4

If you want to use a ScanHost node with no parallel bus datapath, using the Streaming-Through-IJTAG mode, declare the ScanHost without a Datapath wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
SSN {
   ijtag_host_interface : Sib(ssn);
   ScanHost(1) {
   }
}
```

Note

There is no Datapath wrapper.

See the section "Streaming-Through-IJTAG" on page 4489 for more details on why you might use this mode.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DefaultChildConfiguration

A wrapper used to override the default values for various properties of the child nodes.

Usage

```
DftSpecification(module_name,id) {
  SSN {
    DefaultChildConfiguration {
      Interface {
        bus_clock                      : port_name; // default: prop_name
        bus_data_in                    : port_name; // default: prop_name
        secondary_bus_data_in          : port_name; // default: prop_name
        bus_data_out                   : port_name; // default: prop_name
        select_in                      : port_name; // default: prop_name
        select_out                     : port_name; // default: prop_name
        ssh_is_active                  : port_name; // default: prop_name
        ChainGroup {
          scan_en                      : port_name; // default: prop_name
          scan_en_bypass_in            : port_name; // default: prop_name
          edt_update                   : port_name; // default: prop_name
          edt_update_bypass_in         : port_name; // default: prop_name
          edt_clock                    : port_name; // default: prop_name
          edt_clock_bypass_in          : port_name; // default: prop_name
          test_clock                   : port_name; // default: prop_name
          test_clock_bypass_in         : port_name; // default: prop_name
          shift_capture_clock          : port_name; // default: prop_name
          shift_capture_clock_bypass_in : port_name;
                                                    // default: prop_name
          shift_clock                  : port_name; // default: prop_name
          shift_clock_bypass_in        : port_name; // default: prop_name
          capture_clock                : port_name; // default: prop_name
          capture_clock_bypass_in      : port_name; // default: prop_name
          to_scan_in                   : port_name; // default: prop_name
          to_scan_in_bypass_in         : port_name; // default: prop_name
          from_scan_out                : port_name; // default: prop_name
          from_scan_out_bypass_out     : port_name; // default: prop_name
        }

        IjtagScanInterface {
          reset      : port_name; // default: ijtag_reset
          tck        : port_name; // default: ijtag_tck
          select     : port_name; // default: ijtag_sel
          capture_en : port_name; // default: ijtag_ce
          shift_en   : port_name; // default: ijtag_se
          update_en  : port_name; // default: ijtag_ue
          scan_in    : port_name; // default: ijtag_si
          scan_out   : port_name; // default: ijtag_so
        }
      }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
        bus_clock_period                : time; // default: 2.5ns
        bus_register_reset              : on | off ;
        input_retiming_cell_type        : dff | latch ; // default: dff
        max_capture_clock_pulses        : int;  // default: 7
        max_capture_to_shift_clock_period_ratio : int;  // default: 8
        max_initial_offset              : int;  // default: 2G
        max_scan_chain_length           : int;  // default: 256M
        max_scan_en_mcp                 : 2 | 4 | 8;
        max_edt_update_mcp              : 2 | 4 | 8;
        use_clock_dff_cell              : on | off;
        OnChipCompareMode {
          present                       : on | off;
          status_groups                 : int; // default: 128
        }
      }
    }
  }
```

Description

This wrapper overrides the default values for various properties of the child nodes. The properties with the same names in the child wrappers default to the values specified in this wrapper.

For example, if you want to change the bus_clock_in port on all child nodes to my_clock_name, you specify the bus_clock_in property in this wrapper, and the child nodes will inherit this new name. Each wrapper type that supports one or more child wrappers, such as the Multiplexer or ExtraOutputPath wrapper, also includes the DefaultChildConfiguration wrapper to maintain the inheritance through the wrapper hierarchy. You can specify any property at any level, and it will provide the value for the current level and any level below. Any property not locally specified inherits its value from the DefaultChildConfiguration wrapper in its ancestry.

Arguments

- Interface/*;

  A wrapper that includes properties to specify the port name of all node types. The ChainGroup wrapper only applies to the ScanHost node type. See the syntax summary in the "Usage" section to obtain the default name for each port.

  The secondary_bus_data_in and select_out properties apply only to the Multiplexer node. The remaining three properties apply to all node types.

- bus_clock_period : *time*;

  This property is used to set the bus clock period to be used by the run_synthesis command when synthesizing the SSN nodes in isolation. When synthesizing the SSN nodes with the rest of the functional logic, the SDC procs created by the extract_sdc command, which is automatically called by the extract_icl command, is used. The ssn_bus_clock_period for the entire physical block is configured using the set_load_unload_timing_options command. Refer to the section "ScanHost Scan Interface Operation and Timing Requirements" on page 4498 in the ScanHost page for a description of how to keep a common definition of those settings for both SDC and pattern generation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- bus_register_reset : on | off ;

  A property that enables all nodes to reset bus registers during iReset. If you disable this, iReset does not affect data in bus registers.

- input_retiming_cell_type : dff | latch ;

  A property that specifies the default input retiming cell type. The default value is dff.

- max_* : *int*;

  A set of properties that configure the ScanHost nodes. Refer to the ScanHost "Arguments" section for descriptions of these properties, what the property does, and when you should override the default values for these properties.

- OnChipCompareMode/*

  A set of properties that configure the ScanHost nodes. See the ScanHost "Arguments" section for a description of these properties, what they are used for, and when you should override their default values.

- use_clock_dff_cell : on | off;

  An optional Boolean property that specifies to use clock DFF cells with the ScanHost node. This property defaults to on. If your library does not define a clock DFF cell, the tool generates an RTL DFF cell; however, you may need to replace the RTL DFF cell with a true clock DFF cell later in your physical flow. When you set this property to off, the tool uses an always block in the resultant Verilog instead of the DFF cell.

  Turning on this property can enable better timing management on the clock path by keeping special cells on clock paths.

Examples

The following example overrides the default port name of the IJTAG scan interface for all the child nodes specified in the DftSpecification.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set spec [create_dft_specification -sri_sib_list ssn]
read_config_data -in $spec -from_string {
  SSN {
    DefaultChildConfiguration {
      Interface {
        IjtagScanInterface {
          reset      : my_reset;
          tck        : my_tck;
          select     : my_sel;
          capture_en : my_ce;
          shift_en   : my_se;
          update_en  : my_ue;
          scan_in    : my_si;
          scan_out   : my_so;
        }
      }
    }
    Datapath(1) {
      output_bus_width : 16;
      Pipeline(1) { //operates at 1/4 rate
      }
      BusFrequencyDivider(1) {
        frequency_ratio : 4;
      }
      Pipeline(2) { //operates at full rate
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Datapath

A wrapper used to describe the composition of a Streaming Scan Network datapath.

Usage

```
DftSpecification(module_name,id) {
  SSN {
    Datapath(id) {
      ijtag_host_interface : host_node_leaf_name; // default: Sib(ssn)
      bus_clock_period     : time;                // default: Inherited
      bus_register_reset   : on | off ;
      output_bus_width     : int;                 // mandatory
      output_data_rate     : auto | int | int/int;
      DefaultChildConfiguration {
      }
      Connections {
      }
      BusFrequencyDivider {
      }
      BusFrequencyMultiplier {
      }
      DesignInstance {
      }
      ExtraOutputPath {
      }
      Fifo {
      }
      Multiplexer {
      }
      OutputPipeline {
      }
      Pipeline {
      }
      Receiver1xPipeline {
      }
      ScanHost {
      }
    }
  }
}
```

Description

Specify this wrapper one or more times inside the SSN wrapper to describe the composition of a SSN datapath.

The flow of data goes from the bottom node to the top node within the Datapath wrapper. The BusFrequencyDivider, BusFrequencyMultiplier, Multiplexer, OutputPipeline, Pipeline, and ScanHost wrappers define new SSN nodes the tool creates.

The Connections wrapper specifies the end point connections of the data path. By default, these are the bus_clock_in, bus_data_in, and bus_data_out connections. You must specify the bus_clock_out property if you want to create an output clock port for a tiling flow as shown in Example 1 of the SSN wrapper description.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The ExtraOutputPath wrapper creates a branch out path that typically returns to the main data path as the secondary_bus_data_in connection of a Multiplexer node. The branch out path collects nodes in adjacent blocks as illustrated in Example 2 of the SSN wrapper description.

The DesignInstance wrapper references child block instances for inclusion in the data path.

Based on your physical flow, you may be using a standard bottom up assembly flow, or you may be using the tiling flow.

- If using the standard bottom-up flow, refer to Example 1 of the DesignInstance section that demonstrates constructing the data path at the top level.
- If using the tiling flow, refer to Example 2 of the SSN section that demonstrates constructing the data path at the top level.

Understanding the ICL Modeling of the SSN Network

The SSN network is composed of one or more data paths between a set of input and output ports. The tool checks the connectivity between the nodes and extracts the network using the extract_icl command.

If there are no connection errors, the tool models in ICL the SSN network using the following ICL elements with special custom attributes:

- DataInPort
- DataOutPort
- DataMux
- DataRegister

The data ports of the SSN data path are referenced within DataInPort and DataOutPort elements that have the function_modifier ICL port attribute set to tessent_ssn_bus_data. The DataInPort elements can only be referenced as the source of the DataInPorts on child instances with the same function_modifier value. The DataInPort elements can also be the source of DataRegister elements and the data source for DataMux elements. The select of the DataMux elements must be sourced by normal IJTAG data signals. DataInPort and DataOutPort elements with the function_modifier attribute set to tessent_ssn_bus_data cannot be the target of iWrite and iRead commands. See the I and ICL DRC rules I1 to I13, inclusive, and ICL62, ICL65, ICL120, and ICL121 for more precise definitions of these restrictions.

The SSN clocks are described with ClockPort and ToClockPort ICL elements with the function_modifier attribute set to tessent_ssn_bus_clock. In the ICL file for an SSN node, the bus port elements have the forced_high_dft_signal_list attribute set to ssn_en. The ssn_en DFT signal is automatically asserted active during ICL extraction, and it is automatically set high when the network is used within the ssn_continuity patterns (refer to the create_ssn_continuity_patterns command) and within the ATPG and scan retargeting patterns. The ssn_en DFT signal automatically enables the auxiliary datapath when the datapath

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

connections are made to ports connected to pad buffers equipped with auxiliary input or output logic. See the bus_clock_in property description in the Datapath/Connections wrapper description for more information.

The ScanHost nodes must be signed tessent instruments to be used by ATPG and scan pattern retargeting. They must have the tessent_instrument_type and tessent_instrument_subtype attributes set to mentor::ssn and scan_host, respectively. The tessent_signature attribute must also be present.

All other types of nodes in the network do not need to be signed tessent instruments as long as you described the nodes correctly with the DataInPort, DataOutPort, DataMux, and DataRegister ICL elements with special custom attributes as described above. For example, if you want to provide your own Pipeline node, it is best to use the "process_dft_specification -no_insertion" command to create an example and use this example to create the ICL model of your pipeline stage. Do not copy the tessent_instrument_type, tessent_instrument_subtype, and tessent_signature attributes in your ICL file.

Note

If you manually integrate the SSN datapath after using "process_dft_specification -no_insertion", you must ensure that there are no incidental inversions or bit permutations on the SSN bus. The bit sequence of the datapath must be maintained throughout the SSN bus.

Bus Width Scaling Limitations for Several Active SSHs With Different Frequencies in One Datapath

The following lists the limitations of scaling the bus width when there are several active SSHs with different frequencies in one datapath:

1. When there are several active ScanHost nodes with different frequencies in one datapath, the SSN is programmed in such way that the complete bus is used. This means, that small SSN packets may be enlarged by means of padding bits until the bus width is reached.
2. If the capture of several ScanHost nodes on different datapaths must be aligned, then the packet sizes of these datapaths must also match. Together with the restriction 1., this can result in situations in which no valid set of parameters for a combination of datapaths can be found. If a datapath is subject to the restriction 1. and if this datapath has ScanHost nodes that are capture aligned with ScanHost nodes on other datapaths, then all the involved datapaths must be compatible with respect to their bus widths and packet sizes.

Single-Bit Access Through the IJTAG Network Only

By default, ScanHost nodes have a parallel bus datapath and an IJTAG interface. If you are using SSN with access only through JTAG, you can build the ScanHost nodes without the parallel bus datapath by omitting the Datapath wrapper from your DftSpecification and include the ScanHost node directly. "Example 4" on page 4405 shows how to achieve this.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ijtag_host_interface : *host_node_leaf_name*;

  A property that inherits its value from the ijtag_host_interface property in the SSN wrapper. It uses the exact same format. The default is Sib(ssn). You specify the property here when you have more than one Datapath wrapper, and you want to use a different ijtag host node for each data path. To precisely control the ordering of the IJTAG network for the SSN nodes, specify the ijtag_connection_order property in each SSN hardware node.

- bus_clock_period : *time*;

  This property is used to set the bus clock period to be used by the run_synthesis command when synthesizing the SSN nodes in isolation. When unspecified, it inherits its value from the DefaultChildConfiguration wrapper in the SSN wrapper. When synthesizing the SSN nodes with the rest of the functional logic, the SDC procs created by the extract_sdc command, which is automatically called by the extract_icl command, is used. The ssn_bus_clock_period for the entire physical block is configured using the set_load_unload_timing_options command. Refer to the section "ScanHost Scan Interface Operation and Timing Requirements" on page 4498 in the ScanHost page for a description of how to keep a common definition of those settings for both SDC and pattern generation.

- bus_register_reset : on | off ;

  A property that enables all nodes to reset bus registers during iReset. If you disable this, iReset does not affect data in bus registers.

- **output_bus_width** : ***int***;

  A required property that specifies the width of the output bus. From this value, the output and input bus width of every node within the Datapath wrapper is computed from the top node down to the bottom node. The BusFrequencyDivider and BusFrequencyMultiplier nodes enlarge and reduce the bus width, respectively. The input bus width of the data path is extracted from the input bus width of the bottom node. When DesignInstance nodes are encountered, their built-in output bus width is verified to be consistent with the computed value, and an error is generated when there is an inconsistency.

- output_data_rate : auto | int | int/int;

  A property used to specify the data rate of the output data bus. The rate is expressed in terms of data values per tester period. When left to default to auto, it is inferred by using the ratio of input_data_width/output_data_width. You rarely need to specify an explicit value. It may happen when you use several Datapath wrappers in a single DftSpecification to interconnect different components together and they share the same clock sources. It not needed in standard data path configurations.

Examples

Refer to "Example 1" on page 4395 of the SSN wrapper description to see how an SSN data path is constructed into a block that is to be used in a tiling flow at the next level up. Refer to "Example 2" on page 4399 of the SSN wrapper description to see how the data path connections are made at the top level when using a tiling flow.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Refer to "Example 1" on page 4442 of the DesignInstance wrapper description to see how the data path connections are made at the top level when using the standard bottom up flow.

Refer to the "Example 1" on page 4523 in the ScanHost wrapper description for a detailed example showing how the SSN is used to carry the scan data to and from EDT controllers in both internal and external scan modes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Connections

A wrapper specifying the end point connections of a SSN data path.

Usage

```
DftSpecification(module_name,id) {
  SSN {
    Datapath(id) {
      Connections {
        bus_clock_in  : port_pin_name, ...;   // default: ssn_bus_clock
        bus_data_in   : {- | port_pin_name, ...};
                                   // default: ssn_bus_data_in[%d]
        bus_clock_out : port_pin_name, ...;
        bus_data_out  : {- | port_pin_name, ...};
                                   // default: ssn_bus_data_out[%d]
        delete_pre_existing_functional_source : on | off ;
      }
    }
  }
}
```

Description

This wrapper specifies the end point connections of a SSN data path. As shown in the "Usage" section, the wrapper provides end point objects for the data, and clock input and output connections. The data input and output connections as well as the clock input default to ports.

The output clock connection is null by default. As shown in "Example 1" on page 4395 in the SSN wrapper description, you specify the bus_clock_out property when you want to feed the clock through the block when you are using a tiling flow at the next level up. As shown in "Example 2" on page 4399 of the SSN wrapper description, you override the default port connection to pins of the DesignInstance node when creating secondary data path connections.

Arguments

- bus_clock_in : *port_pin_name*, ...;

  A property that specifies the source of the bus clock for the data path. When unspecified, it defaults to a port named ssn_bus_clock_in. The tool ignores the value if you specified the bus_clock_in property of the top node. The bus_clock_in connection of the Datapath wrapper also serves as the clock source for the ExtraOutputPath wrapper specified directly in the Datapath wrapper. Refer to "Example 1" on page 4395 in the SSN wrapper description to see an ExtraOutputPath wrapper specified directly in the Datapath wrapper where the specified bus_clock_out connection of the ExtraOutputPath wrapper is sourced by the bus_clock_in connection of the Datapath wrapper.

  The value you specify must refer to an existing pin, or to an input or in/out port that may or may not exist in the current design. The tool creates the connection if not already present when the *design_level* you specified with the set_design_level command is physical_block or sub_block.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When the design level is chip, the port must already exist and be connected to a pad buffer cell. You may also have a boundary scan cell equipped with AuxiliaryInputData logic. When you point to such a top-level port, the tool automatically maps the connection across the pad and optional boundary scan cell.

- bus_data_in : {- | *port_pin_name*, ...};

  A property used to specify the source of the data for the data path. When unspecified, the property defaults to a port named ssn_bus_data_in[%d]. When specified as a hyphen (-), the pin is tied low. Otherwise, the specified value is used as the data-in source for the bottom node specified in the Datapath wrapper. The bus_data_in connection of the Datapath wrapper also serves as the data source for ExtraOutputPath wrapper specified directly in the Datapath wrapper. See "Example 1" on page 4395 in the SSN wrapper description to see an ExtraOutputPath wrapper specified directly in the Datapath wrapper where the specified bus_data_out connection of the ExtraOutputPath wrapper is sourced by the bus_data_in connection of the Datapath wrapper.

  Note

  Generally, the tool makes a connection from an SSN instance to the ends of the SSN (such as bus_data_in and bus_data_out). When you specify "-" as a value to these properties, the tool skips making this connection. Instead, you can use a custom process_dft_specification.post_insertion callback procedure to customize the datapath connection. See "register_callback" on page 1586 for more information about the callback procedure.

  This custom procedure uses instance pointers for SSN nodes. For more information about retrieving the instance pointers for SSN nodes inserted during processing of the DftSpecification, refer to the section "Instantiated Nodes Section in the SSN Instrument Dictionary" on page 4393.

  When connecting to a bus port or pin, you can use [%d] as the index. The %d symbol will automatically be converted to "*n*-1:0" where *n* is the computed input bus width of the data path. See the description of the output_bus_width property in the Datapath section for a description of how the input and output bus width is computed for all nodes within a Datapath wrapper and how the computed bus width of the bottom node is used as the size of the input bus width of the data path. If you want to connect to several scalar or bus segments, specify the multiple object names separated by a comma. The first element corresponds to the most significant bit of the bus. The sum of the sizes of all the objects must match the computed bus input size.

  When the design level is chip, the port must already exist and be connected to a pad buffer cell. You may also have a boundary scan cell equipped with AuxiliaryInputData logic. When you point to such a top-level port, the tool automatically maps the connection across the pad and optional boundary scan cell.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- bus_clock_out : *port_pin_name*, ...;

  A property used to specify a destination connection for the bus clock. When unspecified, the property defaults to a null string, which means no output bus clock connection is to be made. The connection is created as follows:

  - If the top node within the Datapath wrapper is a DesignInstance with a bus_data_out port, it will be used to source the bus_clock_out connection.
  - If the top node within the Datapath wrapper is a DesignInstance with a bus_data_in port, it will be used to source the bus_clock_in connection.
  - If the top node is not a DesignInstance with a bus_data_out port, the source of its bus_clock_in will be used.
  - If there are no nodes inside the Datapath wrapper (like in the Datapath(up1)) wrapper in Example 2 of the SSN section, the bus_clock_in value of the Datapath is used to source the bus_clock_out connection.

  When the design level is chip, the port must already exist and be connected to a pad buffer cell. You may also have a boundary scan cell equipped with AuxiliaryOutputData logic. When you point to such a top-level port, the tool automatically maps the connection across the pad and optional boundary scan cell.

- bus_data_out : {- | *port_pin_name* ,...};

  A property used to specify the destination of the data for the data path. When unspecified, the data path defaults to a port named ssn_bus_data_out[%d]. When specified as a hyphen (-), the pin is open. Otherwise, the specified value is used as the data out destination for the top node specified in the Datapath wrapper. If there are no nodes inside the Datapath wrapper (like in the Datapath(up1)) wrapper in "Example 2" on page 4399 in the SSN wrapper description, the bus_data_in value of the Datapath is used to source the bus_data_out connection.

  Note

  Generally, the tool makes a connection from an SSN instance to the ends of the SSN (such as bus_data_in and bus_data_out). When you specify "-" as a value to these properties, the tool skips making this connection. Instead, you can use a custom process_dft_specification.post_insertion callback procedure to customize the datapath connection. See "register_callback" on page 1586 for more information about the callback procedure.

  This custom procedure uses instance pointers for SSN nodes. For more information about retrieving the instance pointers for SSN nodes inserted during processing of the DftSpecification, refer to the section "Instantiated Nodes Section in the SSN Instrument Dictionary" on page 4393.

  See the second paragraph of the bus_data_in property to see how connections to buses and scalars are specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When the design level is chip, the port must already exist and be connected to a pad buffer cell. You may also have a boundary scan cell equipped with AuxiliaryOutputData logic. When you point to such a top-level port, the tool automatically maps the connection across the pad and optional boundary scan cell.

- delete_pre_existing_functional_source : on | off ;

  A property that enables you to delete a preexisting functional source and make a connection to SSN logic for ssn_bus_clock_out and ssn_bus_data_out. If you use the same pin for both the SSN datapath and the DFT modal connection while this property is off, the tool reports an error. If this property is on, the SSN connection silently deletes the connection made by the DFT modal connection.

Examples

Refer to "Example 1" on page 4395 in the SSN wrapper description to see how the bus_clock_out property was used to specify an output port to be fed by the bus clock in port such that the block can be used in a tiling flow at the next level up. Refer to "Example 2" on page 4399 in the SSN wrapper description to see how the data path connections are made at the top level when using a tiling flow.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BusFrequencyDivider

A wrapper used to specify the creation of a bus frequency divider node

Usage

```
DftSpecification(module_name,id) {
  SSN {
    Datapath(id) {
      BusFrequencyDivider(id) {
        ijtag_host_interface   : host_node_leaf_name; // default: Sib(ssn)
        ijtag_connection_order : int | "";
        frequency_ratio        : int ;  // mandatory
        input_retimed          : on | off;
        input_retiming_cell_type :  dff | latch ; // default: Inherited
        bus_clock_period       : time ; // default: Inherited
        parent_instance        : instance_name;
        leaf_instance_name     : leaf_instance_name;
        Interface {
          bus_clock    : port_name; // default: Inherited
          bus_data_in  : port_name; // default: Inherited
          bus_data_out : port_name; // default: Inherited
          IjtagScanInterface {
            reset      : port_name; // default: Inherited
            tck        : port_name; // default: Inherited
            select     : port_name; // default: Inherited
            capture_en : port_name; // default: Inherited
            shift_en   : port_name; // default: Inherited
            update_en  : port_name; // default: Inherited
            scan_in    : port_name; // default: Inherited
            scan_out   : port_name; // default: Inherited
          }
        }
        Connections {
          bus_clock_in : port_pin_name;
        }
        ExtraOutputPath {
        }
      }
    }
  }
}
```

Description

This wrapper creates and optionally inserts an SSN node of type bus frequency divider. Use this node to slow down the bus data rate by a factor of *N* by making the data wider by the same factor.

The phase counter registers within the bus frequency divider node are reset asynchronously by the ijtag_reset signal. The update_en pulse also resets the phase counters, enabling you to run several SSN pattern sets in series without resetting the IJTAG network. You can choose to make the ijtag_clock and ijtag_ue ports inactive, as long as all the SSN clocks in the active network are kept in lockstep from the time the ijtag_reset is released.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

See also "Bus Width Scaling Limitations for Several Active SSHs With Different Frequencies in One Datapath" on page 4413 in the Datapath description.

The next two sections describe why you may want to slow down the data rate.

Using the BusFrequencyDivider to Cross CTS Regions

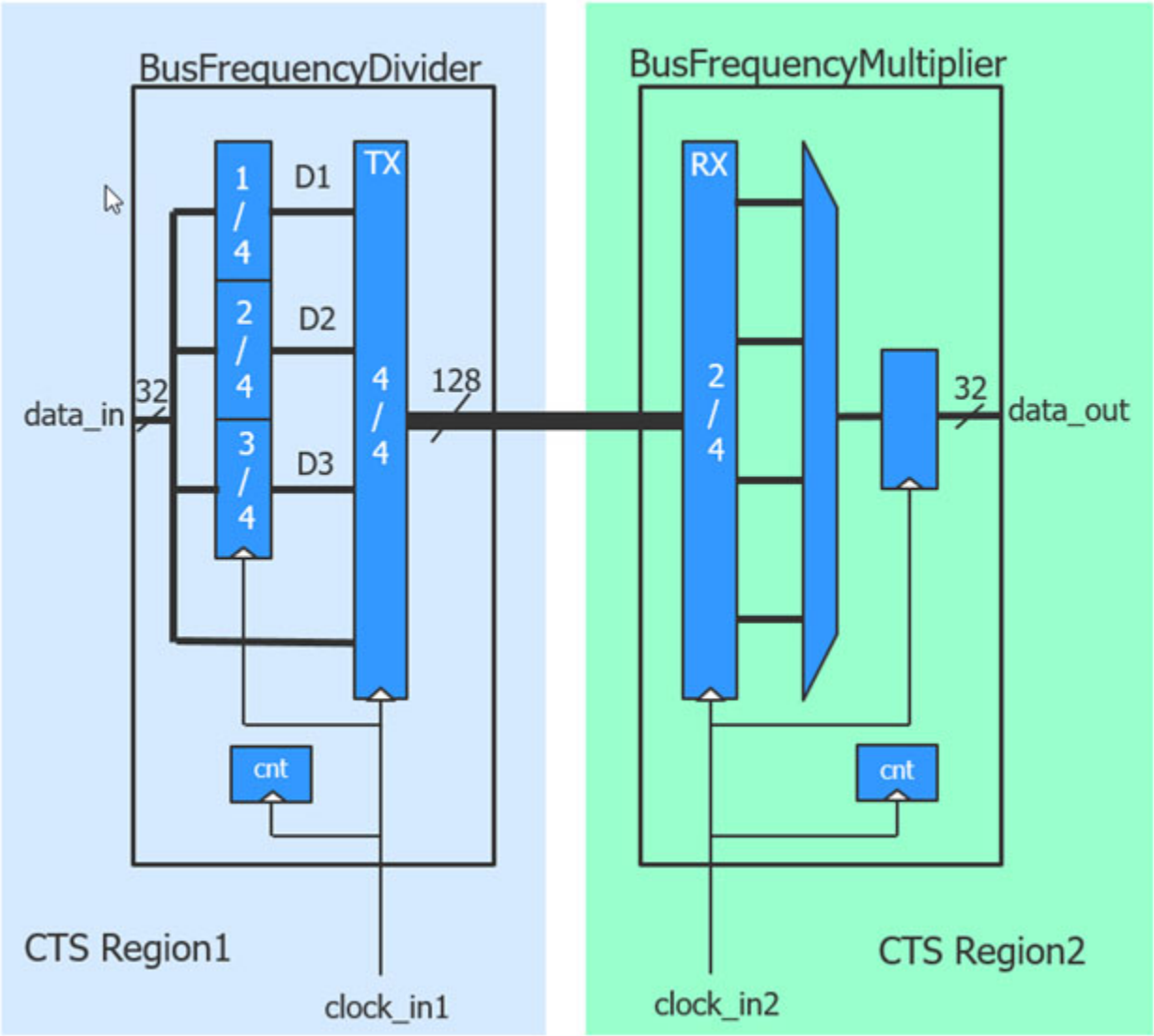
The first reason to use the BusFrequencyDivider node is when the SSN bus crosses a clock domain boundary. The SSN bus is a synchronous bus, and the data output launched from one node must meet setup and hold time requirements at the input of the next node. You may not, however, want to create a single Clock Tree Synthesis (CTS) region for the SSN bus clock across the entire Datapath at once. Instead, you may want to limit the CTS region to groups of cores and use a pair of BusFrequencyDivider/BusFrequencyMultiplier nodes to transfer the data synchronously from one CTS region to the next. To accomplish this, you need to slow down the bus data rate enough to allow sampling of the data on the other CTS region in the center of the data eye with enough timing margin to tolerate the skew between the two CTS regions.

Figure 13-86 shows a BusFrequencyDivider in one CTS region (blue) feeding data to a BusFrequencyMultiplier node on another CTS region (green). Figure 13-87 shows the timing diagram of the data going through the two nodes. The RX data is sampled in the middle of the TX eye opening, which means that the skew between the blue and green CTS region can be as much as two clock periods in either direction. You can select the frequency ratio for the two nodes as large as you want in order to tolerate any amount of skew between the CTS regions. A ratio of four is often enough.

You can implement the clock domain crossing at the boundary of the physical block, as shown in Example 2. The advantage of this method is that it localizes each clock domain to each physical block. However, this results in a wide data bus crossing the physical block boundary. You can also implement the clock domain crossing purely within the receiving physical block, as shown in Example 3. The advantage of this method is that you maintain a narrow data bus when crossing the physical block boundary. As a result, you must bring the sourcing clock into the receiving block, but you use a miniature localized clock tree for the BusFrequencyDivider node and use the reliable forward timing technique to send data from the sending physical block to the receiving physical block.

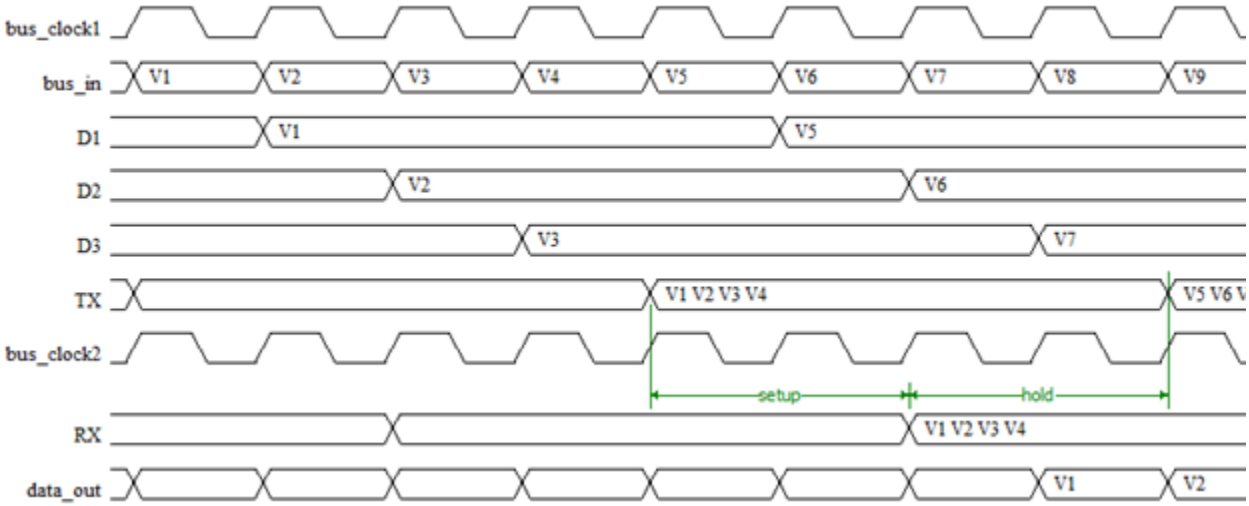
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-86. Schematic Showing Data Handoff Between Two CTS Regions



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

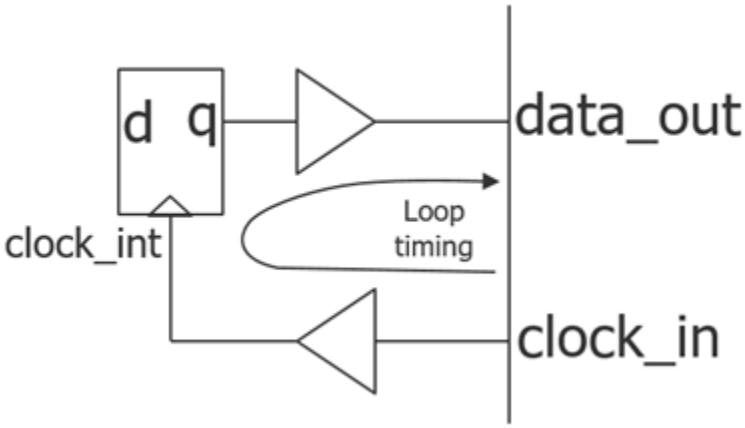
Figure 13-87. Timing Diagram Showing Data Handoff Between Two CTS Regions



Using the BusFrequencyDivider to Enable Faster Internal Bus Clock Frequency

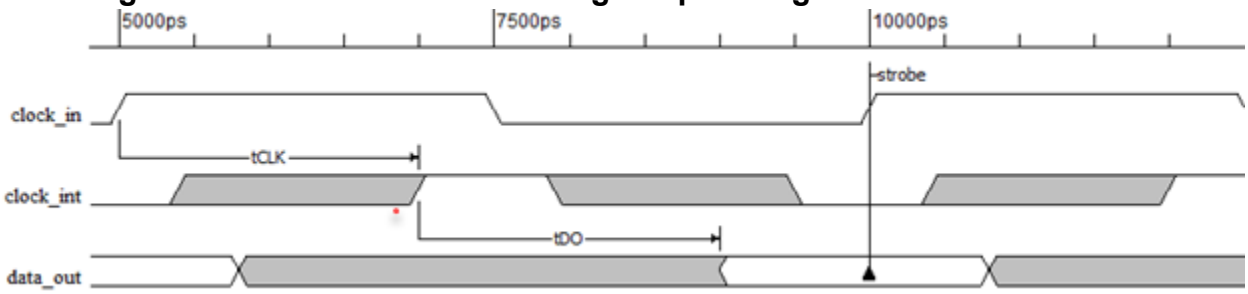
Another usage of the BusFrequencyDivider node is to slow down the data rate before the data exits the chip so as to meet the loop timing from clock_in to data_out in one bus clock period. Figure 13-88 shows the loop timing that must be met in one clock period with worst case timing. If that delay is more than a clock period in worst case timing, it would need to be strobed into the next tester period. With best case timing, however, the loop delay would reduce to about one third and would cross the strobe point. The only way to have a reliable strobe point that works for all device speeds and conditions is to make sure the loop delay is less than a clock period. This is illustrated in the waveform shown in Figure 13-89.

Figure 13-88. Schematic Showing Loop Timing Path for Bus Data Out



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-89. Waveforms Showing Loop Timing Path for Bus Data Out



In a typical chip, the best loop timing you can achieve with normal GPIO pads is 5 ns or a 200 MHz data rate. You can achieve this rate if the clock_in and data_out ports are in the same physical region, and you do not have a large clock tree feeding the last stage of flip-flops. The SSN bus inside the chip can operate faster than 200 MHz. It is easy to close timing at 400 MHz. Higher frequencies can be achieved if the bus clock CTS regions are well designed and not too large.

To run the internal bus at 400 MHz while maintaining a 200 MHz loop timing with the outside world, you use a BusFrequencyDivider node to slow down the data by a factor of two. You can route a 400 MHz 32-bit bus inside the chip while using a 200 MHz 64-bit bus to interact with the tester. You also use an OutputPipeline node between the BusFrequencyDivider node and the output pads to step down the bus clock distribution tree and avoid having the bus clock distribution tree delay in the loop timing path. The source of the bus clock distribution tree might even be in a different physical region, but it is important the clock input for the OutputPipeline node is local to the physical region containing the output pads and the OutputPipeline node.

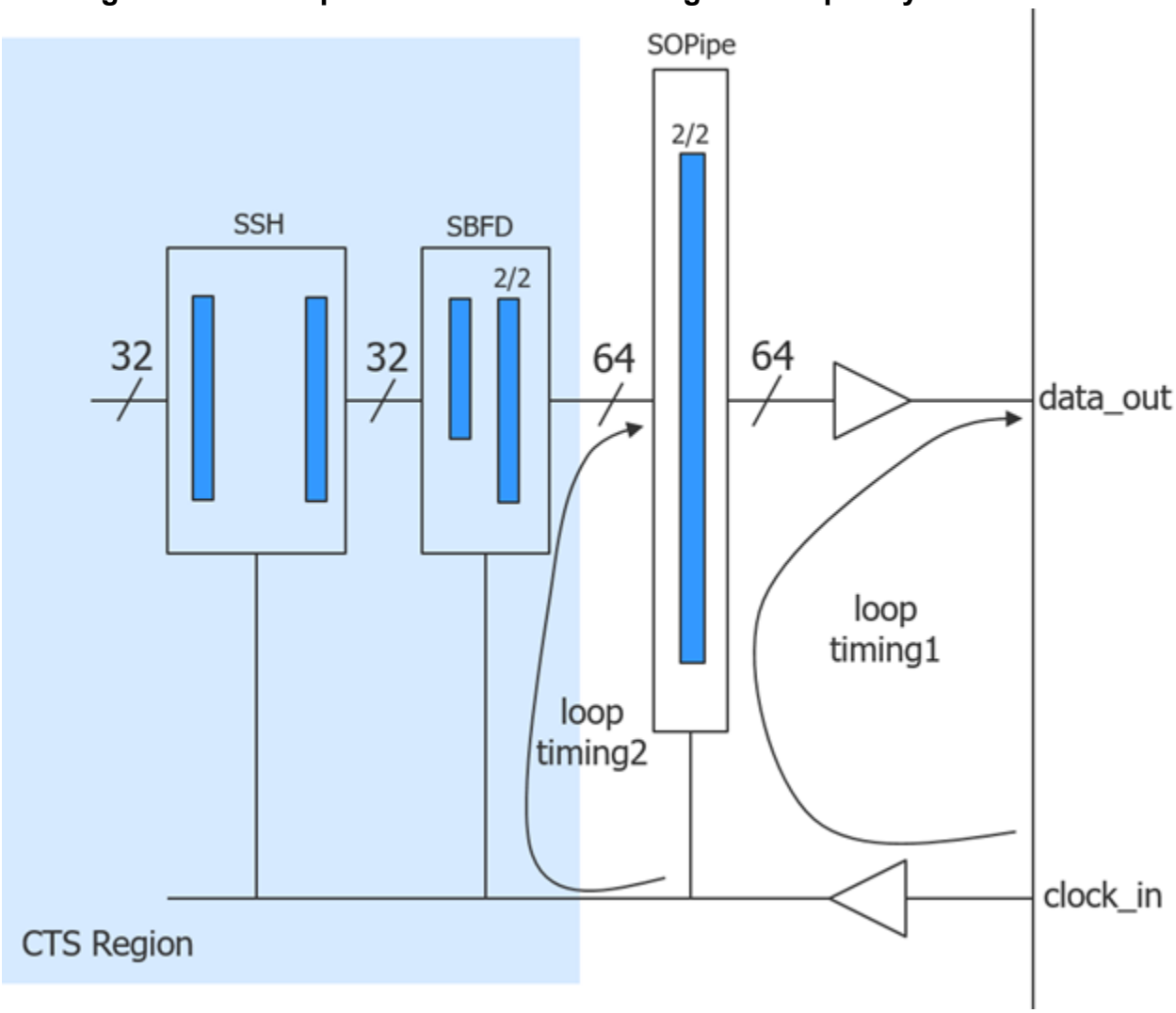
Figure 13-90 shows the schematic view of the slowed-down bus data before being re-timed by an OutputPipeline node using a clock branch that was not balanced with the complete bus clock distribution tree. A persistent buffer is included inside the OutputPipeline node to help define this CTS constraint. The loop timing identified as “loop timing1” in Figure 13-90 is identical to those shown in Figure 13-88 and Figure 13-89. The internal bus rate is 400 MHz, but the BusFrequencyDivider node slowed the rate down to 200 MHz. There is another loop timing path identified as “loop timing2” in Figure 13-90 that behaves identically to the other one. In this second timing loop, the clock tree delay is in the timing path, but no GPIO pad buffers are involved. This tolerates a clock tree insertion delay of up to 5 ns. If the clock tree delay is larger than that, which is unlikely given that the flop count in the bus clock domain is not large, you can insert a Pipeline node between the BusFrequencyDivider node and the OutputPipeline node. You would use a clock branch halfway between the full clock tree and the clock going to the OutputPipeline node. This enables splitting “loop timing2” into two loops, doubling the insertion delay for the complete clock tree in worst-case timing conditions.

When using this techniques, the data rate is 200 MHz, but the clock_in frequency is 400 MHz. You achieve this on the tester by having a 5 ns tester period (200 MHz) but using a 2 pulse clock waveform for the clock. The memory storage is still one location per data value but the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

clock is allowed to pulse twice per tester cycles. The clock will be automatically created as such by the tool when using the SSN network and BusFrequencyDivider nodes are present.

Figure 13-90. Output Data Slow Down Using BusFrequencyDivider Node



Arguments

- ijtag_host_interface

  ijtag_host_interface : *host_node_leaf_name*;

  A property that inherits its value from the ijtag_host_interface property in the Datapath wrapper. It has the exact same format. The default is Sib(ssn). You specify it here when you want to use a different IJTAG host interface for each node, but it is unlikely that you need to do that. To precisely control the ordering of the IJTAG network for the SSN nodes, specify the ijtag_connection_order property in each of the SSN hardware nodes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ijtag_connection_order : *int* | “”;

  An optional property that controls the connection order for the IJTAG scan interface of this node relative to the others sharing a common ijtag_host_interface value. When left unspecified, it defaults to a null string, which means that the connection order for the IJTAG scan interface matches the relative order of the node with respect to the others in the DFT specification. The IJTAG scan chain gets connected from the bottom node to the top node in the SSN wrapper. If you specify an integer value for one node, then you must specify it for all other nodes having the same ijtag_host_interface value and the integer value must be unique for each node. The node with the smallest ijtag_connection_order value will be closest to tdo.

- parent_instance : *instance_name*;

  A property that specifies the parent instance in which the given node should be instantiated into. When unspecified, the node is instantiated directly into the current design.

- leaf_instance_name : *leaf_instance_name*;

  A property that specifies the leaf instance name of the node. When unspecified, the leaf instance name defaults to the module name with the “_inst” suffix appended to it.

- frequency_ratio : *int* ;

  A mandatory property that specifies the maximum frequency ratio the node has between the input and output data bus. The value must be one of the following even integers:

  - 2
  - 4
  - 6
  - 8
  - 10
  - 12
  - 14
  - 16

- input_retimed : on | off;

  An optional property that specifies the insertion of a negative edge retiming flip-flop stage on the bus data inputs. Use this property when implementing a source synchronous timing interface to the BusFrequencyDivider node, as explained and illustrated in Example 3.

- input_retiming_cell_type : dff | latch ;

  A property that specifies the default input retiming cell type. The default value is inherited from the DefaultChildConfiguration wrapper in the ancestry of the BusFrequencyDivider node. This property has an effect only when the input_retimed property is set to on.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- bus_clock_period : *time*;

  This property is used to set the bus clock period to be used by the run_synthesis command when synthesizing the SSN nodes in isolation. When unspecified, it inherits its value from the DefaultChildConfiguration wrapper in its ancestry wrappers. When synthesizing the SSN nodes with the rest of the functional logic, the SDC procs created by the extract_sdc command, which is automatically called by the extract_icl command, is used. The ssn_bus_clock_period for the entire physical block is configured using the set_load_unload_timing_options command. Refer to the section "ScanHost Scan Interface Operation and Timing Requirements" on page 4498 in the ScanHost page for a description of how to keep a common definition of those settings for both SDC and pattern generation.

- Interface

  The properties in the Interface wrapper enable you to change the names of all the ports on the node. Refer to the Interface wrapper of the DefaultChildConfiguration wrapper to see the default value of each port name. If the property within the local Interface wrapper is not specified, the value is inherited from the Interface wrapper in the first DefaultChildConfiguration wrapper found in the ancestry of the current node.

- Connections

  bus_clock_in : *port_pin_name*;

  A property that can be used to specify the source of the bus clock for the BusFrequencyDivider node. This connection is typically inferred automatically. If unspecified and a DesignInstance wrapper is found below the BusFrequencyDivider wrapper and this instance was created with an output bus clock port, the bus clock input of the BusFrequencyDivider node will connect to it unless the ignore_bus_clock_out property is set to off in the DesignInstance/Connections wrapper. If a node below the BusFrequencyDivider wrapper has its Connection/bus_clock_in property specified, it will be used by the BusFrequencyDivider node also. Otherwise, it will connect to the object specified by the Datapath/Connections/bus_clock_in property, which defaults to "ssn_bus_clock". The only time you will specify this property is when you have a CTS boundary inside the core, and you are transferring the bus data from one domain to the other as shown in Example 1.

  In the node is specified directly above a DesignInstance wrapper having more than one bus_clock_in, then you must specify this property as its default source cannot be inferred.

  When the design level is chip, the port must already exist and be connected to a pad buffer cell. You may also have a boundary scan cell equipped with AuxiliaryInputData logic. When you point to such a top-level port, the tool automatically maps the connection across the pad and optional boundary scan cell.

Examples

Example 1

This example shows how to cross over between two CTS regions when the boundary is inside a single physical region. When the boundary is between the physical regions, which is the more common case, you use the flow shown in "Example 2" on page 4429.

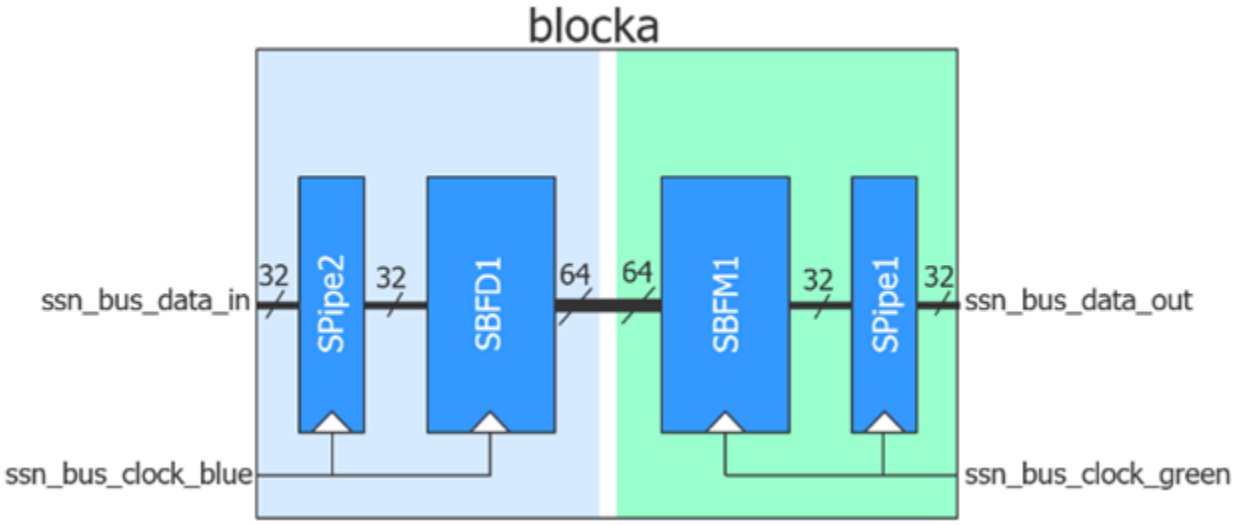
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-91 on page 4429 shows a schematic of the network built with the example DFT specification that follows this paragraph. The two clocks enter the block and each side of the network uses the appropriate clock. You only need to override the bus_clock_in connections of the BusFrequencyMultiplier node because the nodes above it automatically inherit the connection.

```
set_spec [create_dft_spec -sri_sib_list ssn]

read_config_data -in $spec -from_string {
  SSN {
    Datapath(1) {
      output_bus_width : 32;
      Connections {
        bus_clock_in : ssn_bus_clock_blue, ssn_bus_clock_green;
      }
      Pipeline(1) {
        Connections {
          bus_clock_in : ssn_bus_clock_green;
        }
      }
      BusFrequencyMultiplier(1) {
        frequency_ratio : 2;
        Connections {
          // This connection is inherited by nodes above
          bus_clock_in : ssn_bus_clock_green;
        }
      }
      BusFrequencyDivider(1) {
        frequency_ratio : 2;
        Connections {
          bus_clock_in : ssn_bus_clock_blue;
        }
      }
      Pipeline(2) {
        Connections {
          bus_clock_in : ssn_bus_clock_blue;
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-91. SSN Clock Domain Crossing Inside Physical Region



Example 2

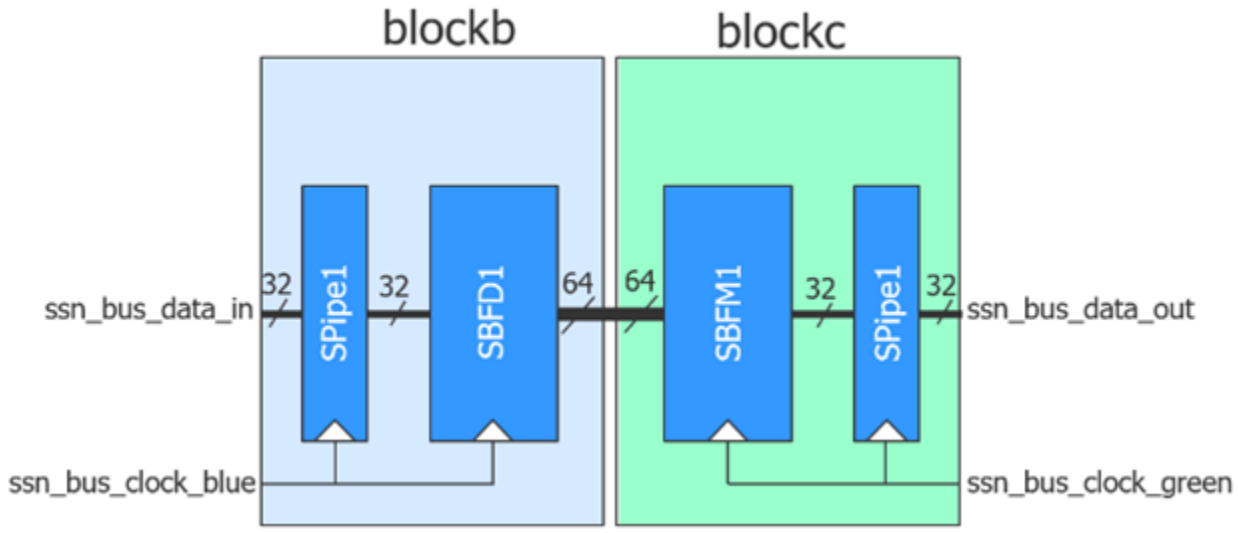
This example implements the same logic as in Example 1 but this time with the domain crossing at the boundary of two physical blocks as shown in Figure 13-92. The first dofile inserts the SSN in blockb, the second one in blockc, and the third one makes the connections between them in the parent block.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# Dofile for "blockb"

set spec [create_dft_specification -sri_sib_list ssn
read_config_data -in $spec -from_string {
  SSN {
    Datapath(1) {
      output_bus_width : 64;
      BusFrequencyDivider(1) {
        frequency_ratio : 2;
      }
      Pipeline(1) {
      }
    }
  }
}

# Dofile for "blockc"

set spec [create_dft_specification -sri_sib_list ssn
read_config_data -in $spec -from_string {
  SSN {
    Datapath(1) {
      output_bus_width : 32;
      Pipeline(1) {
      }
      BusFrequencyMultiplier(1) {
        frequency_ratio : 2;
      }
    }
  }
}

# Dofile for "parent block"

set spec [create_dft_specification]
read_config_data -in $spec -from_string {
  SSN {
    Datapath(1) {
      output_bus_width : 32;
      Connections {
        bus_clock_in : ssn_bus_clock_blue, ssn_bus_clock_green;
      }
      DesignInstance(blockc) {
        Connections {
          bus_clock_in : ssn_bus_clock_green;
        }
      }
      DesignInstance(blockb) {
        Connections{
          bus_clock_in : ssn_bus_clock_blue;
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-92. SSN Clock Domain Crossing Across Physical Regions

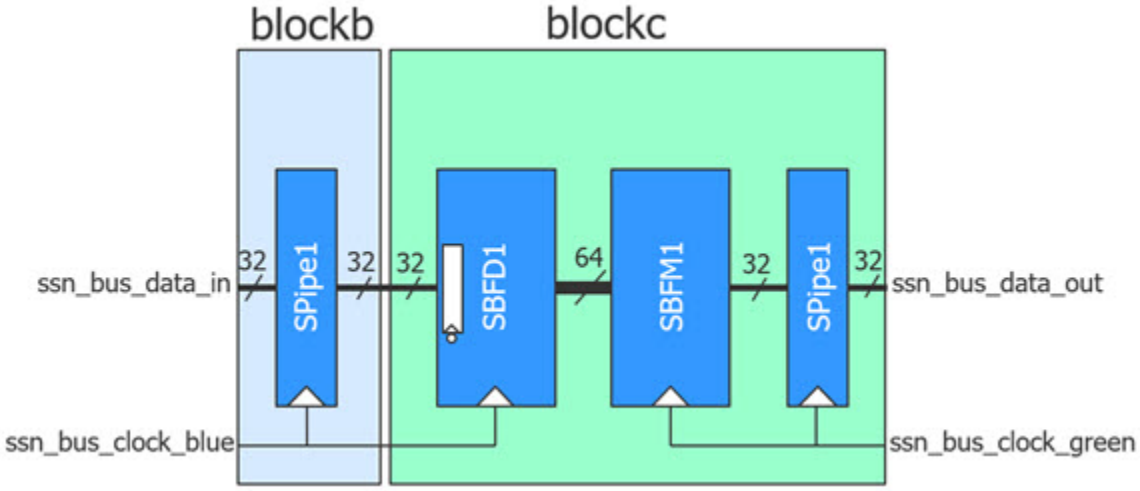


Example 3

This example implements the clock domain crossing shown in Example 2 but this time with the domain crossing localized into the receiving physical block. Compare the differences between Figure 13-92 from Example 2 and Figure 13-93 in this example. The data bus width when crossing the physical block region is a smaller width in this example. The BusFrequencyDivider node is placed in the receiving physical block, and it receives its clock from the physical block sourcing the data (blockb in the example). The BusFrequencyDivider node is built with the input_retimed property set to “on” so that a stage of negative edge retiming flip-flops gets added on the bus data input. This enables building a miniature local clock tree for the BusFrequencyDivider node and building a forward timing interface between it and the sourcing physical block. A forward timing interface (often referred to as a “source synchronous interface”) is one where the data and the clock are sourced together from a common block. The destination block strobes the data using the clock edge opposite from the one used at the source. This use of opposite clock edges results in the data and clock relative delay being able to tolerate as much skew as nearly 50% of the clock period. This kind of interface is effective even at very high frequencies because of the high correlation between the clock and datapath, which cancels the bulk of the uncertainty between them.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

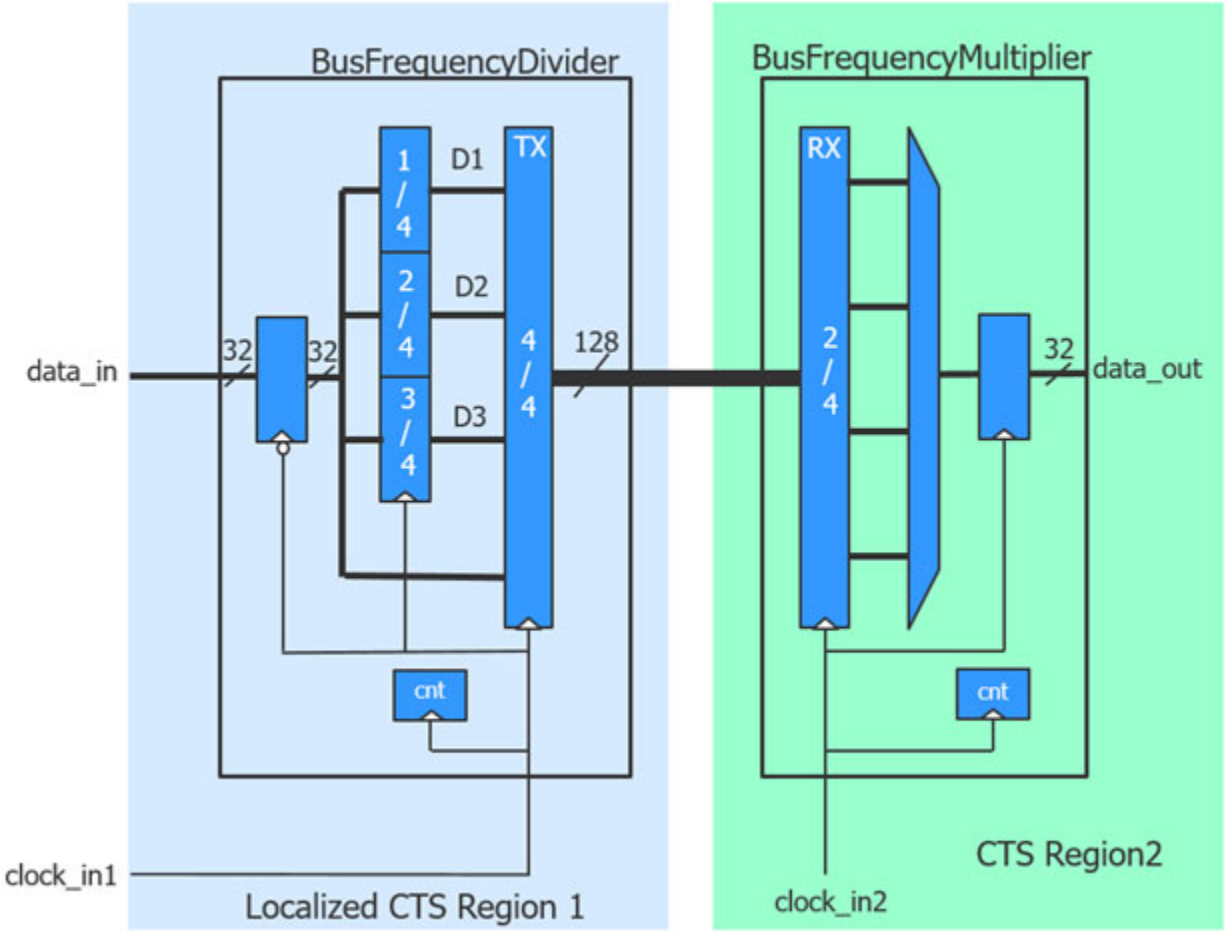
Figure 13-93. Clock Domain Crossing at the Input of the Receiving Block



The block diagram in Figure 13-94 shows the BusFrequencyDivider node sourcing data towards the BusFrequencyMultiplier node, similar to the sourcing in Figure 13-86. The timing of the data between them is identical and thus still explained with the timing diagram shown in Figure 13-87. The difference between these figures is the stage of retiming flip-flops added on the bus data_in before the node uses it. This retiming stage implements the source synchronous timing described previously by using the opposite phase of the clock to strobe the data as compared to the one used to source the data.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-94. BusFrequencyDivider With Input Retimed



The following is the DftSpecification used to create an SSN network like the one in blockc of Figure 13-93. The Datapath/Connections/bus_clock_in property lists the two clock inputs, and the Connections/bus_clock_in property within the nodes specifies which clock to use for each node. The Connections/bus_clock_in property is not specified within the Pipeline node, so it inherits its value from the node below it, as documented in the Connections/bus_clock_in property of each node.

```
SSN {
  DataPath(1) {
    output_bus_width :32;
    Connections {
      bus_clock_in : ssn_bus_clock_blue, ssn_bus_clock_green;
    }
    Pipeline(1) {
    }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
        BusFrequencyMultiplier(in) {
          frequency_ratio : 4;
          Connections {
            bus_clock_in : ssn_bus_clock_green;
          }
        }
        BusFrequencyDivider(in) {
          input_retimed   : on;
          frequency_ratio : 4;
          Connections {
            bus_clock_in : ssn_bus_clock_blue;
          }
        }
      }
    }
```

The following is the DftSpecification used to connect the ssn_bus_clock_blue input of blockc to the ssn_bus_clock_blue output on blockb at the next level up. As described in the DesignInstance/Connections/bus_clock_in property, the clocks ports are ordered alphabetically when deciding which connection goes with which pin on the instance.

```
      Datapath(1) {
        Connections {
          bus_clock_in : blue_i1/ssn_bus_clock_blue_out,
                         ssn_bus_clock;
          }
        }
        DesignInstance(green_i1) {
          Connections {
            bus_clock_in : blue_i1/ssn_bus_clock_blue_out,
                           ssn_bus_clock;
          }
        }
        DesignInstance(blue_i1) {
          Connections {
            bus_clock_in : ssn_bus_clock;
          }
        }
```

The clock input list of the datapath 1 must be specified to only contain ssn_bus_clock using the “set_icl_ssn_datapath_ports -clock_inputs” switch; otherwise, the tool reports an error during extract_icl that the clock pin “blue_i1/ssn_bus_clock_blue_out” cannot be traced to a top level port.

```
set_icl_ssn_datapath_ports -name 1 -clock_inputs ssn_bus_clock
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BusFrequencyMultiplier

A wrapper used to specify the creation of a bus frequency multiplier node

Usage

```
DftSpecification(module_name,id) {
  SSN {
    Datapath(id) {
      BusFrequencyMultiplier(id) {
        ijtag_host_interface    : host_node_leaf_name; // default: Sib(ssn)
        ijtag_connection_order : int | "";
        frequency_ratio        : int ;  // mandatory
        update_phase           : transmitter | receiver;
        bus_clock_period       : time ; // default: Inherited
        parent_instance        : instance_name;
        leaf_instance_name     : leaf_instance_name;
        Interface {
          bus_clock     : port_name; // default: Inherited
          bus_data_in   : port_name; // default: Inherited
          bus_data_out  : port_name; // default: Inherited
          IjtagScanInterface {
            reset       : port_name; // default: Inherited
            tck         : port_name; // default: Inherited
            select      : port_name; // default: Inherited
            capture_en  : port_name; // default: Inherited
            shift_en    : port_name; // default: Inherited
            update_en   : port_name; // default: Inherited
            scan_in     : port_name; // default: Inherited
            scan_out    : port_name; // default: Inherited
          }
        }
        Connections {
          bus_clock_in : port_pin_name;
        }
        ExtraOutputPath {
        }
      }
    }
  }
}
```

Description

This wrapper is used to create and optionally insert an SSN node of type bus frequency multiplier. Use this node to speed up the bus data rate by a factor of *N* by narrowing the data bus by the same factor.

The phase counter registers within the bus frequency multiplier node are reset asynchronously by the ijtag_reset signal. The update_en pulse also resets the phase counters, enabling you to run several SSN pattern sets in series without resetting the IJTAG network. You can choose to make the ijtag_clock and ijtag_ue ports inactive, as long as all the SSN clocks in the active network are kept in lockstep from the time the ijtag_reset is released.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

See also "Bus Width Scaling Limitations for Several Active SSHs With Different Frequencies in One Datapath" on page 4413 in the Datapath description.

You may want to speed up the data rate for two reasons.

Using the BusFrequencyMultiplier When the SSN Datapath Crosses a Clock Boundary

The first usage of the BusFrequencyMultiplier node is when the SSN datapath crosses a clock domain boundary. This usage is complementary to the BusFrequencyDivider node and is described in "Using the BusFrequencyDivider to Cross CTS Regions."

Using the BusFrequencyMultiplier to Allow Faster Internal Bus Clock Frequency

Another usage of the BusFrequencyMultiplier node is to speed up the data rate on the internal bus while keeping the Input data rate slow enough to meet the relative timing between clock_in and data_in in one bus clock period. The limiting factor for the data rate is the loop timing on the output side described in "Using the BusFrequencyDivider to Enable Faster Internal Bus Clock Frequency." Because the tester operates at the output data rate and can only provide one data value per tester cycle, you must align the input data rate to match the output data rate when you reach the chip level and want to use the SSN datapaths to apply the manufacturing test patterns. You use a double-pulse clock and a BusFrequencyMultiplier node to speed up the data rate internally.

Figure 13-95 shows the relative timing that must be met in one tester period. For simplicity, the holding multiplexer in front of the flip-flop is not shown but it is present in the hardware such that the data_in can be strobed once out of *N* clock cycles in the BusFrequencyMultiplier node. The propagation of the clock path is subtracted from the propagation delay of the data. The data_in values are launched aligned with the first falling edge of clock_in. The BusFrequencyMultiplier node samples the data on the first rising edge of the clock. The clock path is typically longer than the datapath as it includes the full CTS delay but the data_in path can always be slowed down with extra buffers to match more closely the clock_in delay when needed. Figure 13-96 and Figure 13-97 shows the relative timing waveforms with fast and slow timing. As you can see, the hold time reduces as the timing slows down but the layout tool will insert hold buffers on the datapath to keep the relative path matched enough to meet hold time in all conditions if not already correct by construction.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-95. Relative Timing Path for the Input Datapath

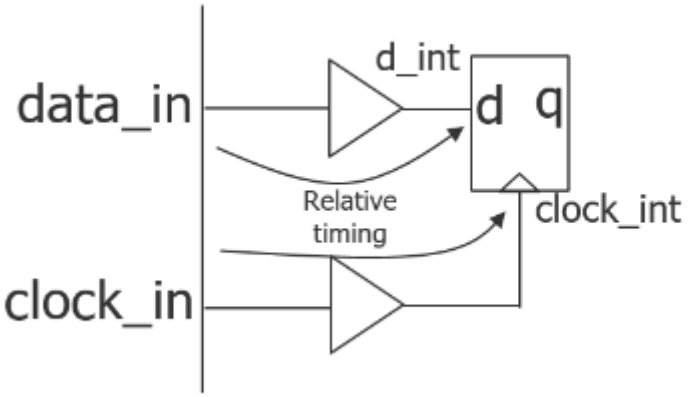


Figure 13-96. Waveform Showing Relative Timing Paths for Input Datapath (fast)

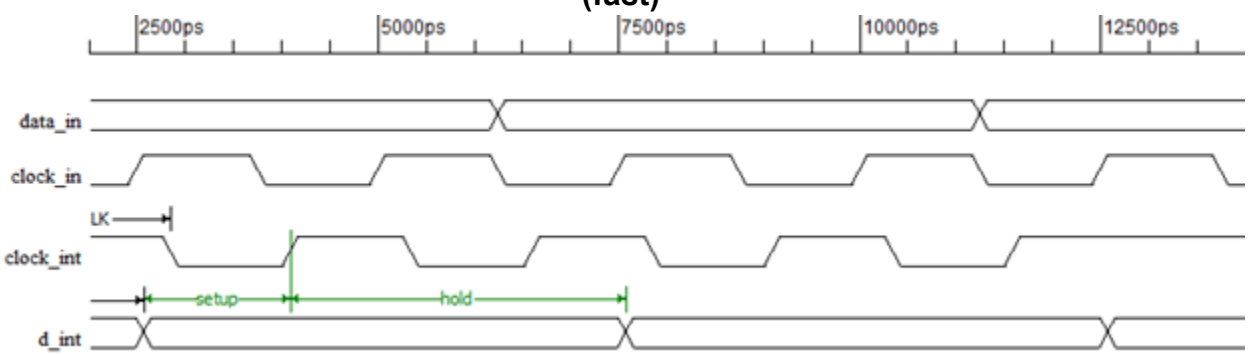
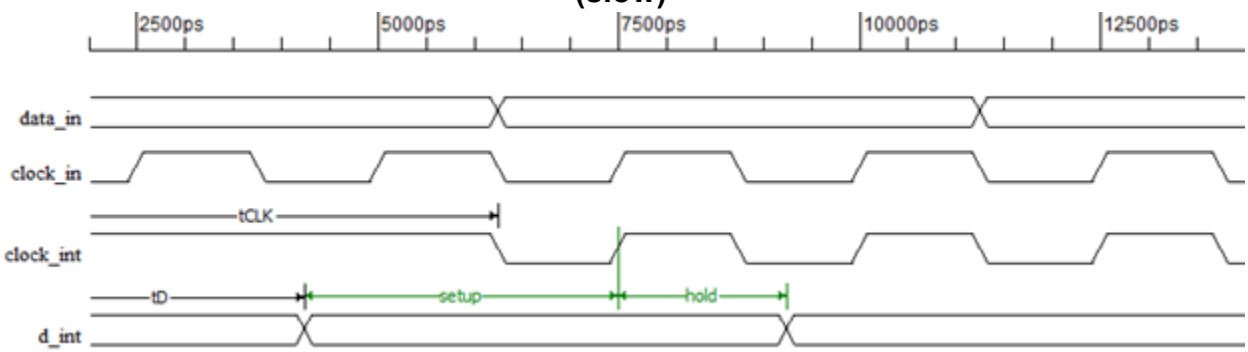


Figure 13-97. Waveform Showing Relative Timing Paths for Input Datapath (slow)



Arguments

- ijtag_host_interface

  ijtag_host_interface : *host_node_leaf_name*;

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

A property that inherits its value from the ijtag_host_interface property in the Datapath wrapper. It has the exact same format. The default is Sib(ssn). You specify the property when you want to use a different IJTAG host interface for each node, but it is unlikely that you need to do that. To precisely control the ordering of the IJTAG network for the SSN nodes, specify the ijtag_connection_order property in each of the SSN hardware nodes.

- ijtag_connection_order : *int* | “”;

  An optional property that controls the connection order for the IJTAG scan interface of this node relative to the others sharing a common ijtag_host_interface value. When left unspecified, it defaults to a null string, which means that the connection order for the IJTAG scan interface matches the relative order of the node with respect to the others in the SSN wrapper. If you specify an integer value for one node, then you must specify the integer value for all other nodes having the same ijtag_host_interface value, and the integer value must be unique for each node. The node with the smallest ijtag_connection_order value will be closest to tdo.

- parent_instance : *instance_name*;

  A property that specifies the parent instance in which the given node should be instantiated into. When unspecified, the node is instantiated directly into the current design.

- leaf_instance_name : *leaf_instance_name*;

  A property that specifies the leaf instance name of the node. When unspecified, the leaf instance name defaults to the module name with the “_inst” suffix appended to it.

- bus_clock_period : *time*;

  This property is used to set the bus clock period to be used by the run_synthesis command when synthesizing the SSN nodes in isolation. When unspecified, it inherits its value from the DefaultChildConfiguration wrapper in its ancestry wrappers. When synthesizing the SSN nodes with the rest of the functional logic, the SDC procs created by the extract_sdc command, which is automatically called by the extract_icl command, is used. The ssn_bus_clock_period for the entire physical block is configured using the set_load_unload_timing_options command. Refer to this “ScanHost Scan Interface Operation and Timing Requirements” on page 4498 in the ScanHost page for a description of how to keep a common definition of those settings for both SDC and pattern generation.

- frequency_ratio : *int* ;

  A mandatory property that specifies the maximum frequency ratio the node has between the input and output data bus. The value must be one of the following even integers:

  - 2
  - 4
  - 6
  - 8
  - 10

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

  - 12
  - 14
  - 16

- update_phase : transmitter | receiver;

  A property that enables you to change the value of the phase counter at which the input data bus is sampled by the BusFrequencyMultiplier node. The default value is "receiver" and is what you need to use in all normal cases. Those two normal cases are explained in the "Using the BusFrequencyDivider to Cross CTS Regions" on page 4421 and "Using the BusFrequencyDivider to Enable Faster Internal Bus Clock Frequency" on page 4423 sections. When you are following a tiling physical flow or a near abutment flow and want to localized the SSN clock tree with in each physical block, you use the value "transmitter" for the BusFrequencyMultiplier node that receives the data from the block to which the clock was sent to. Refer to "Example 3" on page 4400 in the SSN wrapper description for a complete explanation of this flow and why it is better to use the "transmitter" value in this specific case.

- Interface

  The properties in the Interface wrapper enable you to change the name of all the ports on the node. Refer to the Interface wrapper of the DefaultChildConfiguration wrapper to see the default value of each port name. If the property within the local Interface wrapper is not specified, the value is inherited from the Interface wrapper in the first DefaultChildConfiguration wrapper found in the ancestry of the current node.

- Connections

  bus_clock_in : port_pin_name;

  A property that can be used to specify the source of the bus clock for the BusFrequencyMultiplier node. This connection is typically inferred automatically. If unspecified and a DesignInstance wrapper is found below the BusFrequencyMultiplier wrapper and this instance was created with an output bus clock port, the bus clock input of the BusFrequencyMultiplier node will connect to it unless the ignore_bus_clock_out property is set to off in the DesignInstance/Connections wrapper. If a node below the BusFrequencyMultiplier wrapper has its Connection/bus_clock_in property specified, it will be used by the BusFrequencyMultiplier node also. Otherwise, it will connect to the object specified by the Datapath/Connections/bus_clock_in property, which defaults to a port named "ssn_bus_clock". You typically will never use this property as the default value is correct in all cases.

  If the node is specified directly above a DesignInstance wrapper having more than one bus_clock_in, then you must specify this property as the tool will be unable to infer the default source.

  When the design level is chip, the port must already exist and be connected to a pad buffer cell. You may also have a boundary scan cell equipped with AuxiliaryInputData logic.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When you point to such a top-level port, the tool automatically maps the connection across the pad and optional boundary scan cell.

Examples

Example 1

This example shows how to use a BusFrequencyMultiplier node as the first node closest to the chip inputs (bottom of the datapath) to speed up the internal data rate and decrease the size of the internal bus. The output_bus_width is specified as 32 but because the frequency_ratio property is specified as 2 in the BusFrequencyMultiplier node, the input bus width is inferred as 64.

```
set_spec [create_dft_spec -sri_sib_list ssn]


read_config_data -in $spec -from_string {
  SSN {
    Datapath(1) {
      output_bus_width : 32;
      Scanhost(1) {
      }
      BusFrequencyMultiplier(1) {
        frequency_ratio : 2;
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DesignInstance

A wrapper used to include an instance of a sub-block containing one or more SSN data paths into a higher level SSN data path.

Usage

```
DftSpecification(module_name,id) {
  SSN {
    Datapath(id) {
      DesignInstance(instance_name) {
        datapath : id ;
        Connections {
          ignore_bus_clock_out : on | off;
          bus_clock_in         : port_pin_name, ...;
        }
        ExtraOutputPath {
        }
      }
    }
  }
}
```

Description

This wrapper is used to include an instance of a sub-block containing one or more SSN data paths into a higher level SSN data path. You must specify the datapath property in the rare case when the child block has more than one data path; otherwise, the unique one is used automatically. The DesignInstance wrapper is only allowed when you have an elaborated design in memory, and the specified "*instance_name*" must point to an instance that exists *below* the current design.

After you have run the extract_icl command on a block containing SSN nodes, the ICL port elements of the extracted ICL module will include the necessary attributes to allow the Datapath to be properly described. If you inserted the SSN nodes manually, you use the <add_icl_ssn_datapaths> and the <set_icl_ssn_datapath_ports> commands to give the data paths a meaningful *id* name.

Arguments

- *instance_name*

  A string value that refers to the name of an existing instance below the current design. The referenced instance must have one or more SSN data paths, and the member of each data path must be identified using the icl_port tessent_ssn_datapath_ids attribute.

- datapath : *id* ;

  A optional property that is used when the referenced instance corresponds to a module having more than one SSN data path. Each data path of a design instance is connected to a data path at the higher level using one DesignInstance wrapper per data path.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Connections/bus_clock_in : *port_pin_name, ...*;

  A property that can be used to specify the sources of the bus clock ports associated to the specified Datapath ID of the DesignInstance node. This connection is typically inferred automatically when the data path only makes use of a single bus_clock_in port. You must explicitly specify the connections when there is more than one bus_clock_in port. Order the connections to match the alphabetical order of the bus_clock_in port names.

  If unspecified and the DesignInstance wrapper is found above another DesignInstance wrapper and this instance was created with a single output bus clock port, the bus clock input of the current DesignInstance node will connect to it unless the ignore_bus_clock_out property is set to on in the other DesignInstance/Connections wrapper. If a node below the current DesignInstance wrapper has its Connection/bus_clock_in property specified, it will also be used by the current DesignInstance node. Otherwise, it will connect to the object specified by the Datapath/Connections/bus_clock_in property, which defaults to "ssn_bus_clock".

  The only time you would specify this property is when you have a CTS boundary inside the core, and you are transferring data from one domain to the other using a BusFrequencyDivider/BusFrequencyMultiplier node pair. See the description of the BusFrequencyDivider wrapper for a complete description of these nodes and their purpose. See also "Example 1" on page 4427 in the BusFrequencyDivider section.

  If a clock port is used by two data paths of the current DesignInstance, the source must be the same in both DesignInstance wrappers having the datapath property referencing the data paths sharing the same clock port input.

  When the design level is chip, the port must already exist and be connected to a pad buffer cell. You may also have a boundary scan cell equipped with AuxiliaryInputData logic. When you point to such a top-level port, the tool automatically maps the connection across the pad and optional boundary scan cell.

- Connections/ignore_bus_clock_out : on | off;

  A property used to specify that the bus_clock_out port of the current DesignInstance is not to be used as the source of the clock for the nodes above it in the Datapath wrapper. See the description of the Connections/bus_clock_in property above and in the other SSN node types to understand how the bus_clock_in of the node above the DesignInstance will default to connecting its bus_clock_in port to the bus_clock_out port when it exists.

Examples

Example 1

Below is an example where the top-level SSN network is constructed by daisy chaining top level ScanHost and Pipeline nodes with the data path already constructed into child blocks. Notice the connections wrapper in the Datapath wrapper. The specified ports are connected to pad cells and those were equipped with auxiliary input and output logic using the AuxiliaryInputOutputPorts wrapper in a previous DFT insertion pass. The Datapath connection is specified on gpio[8:0]. Those ports are automatically mapped to their associated auxiliary

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

input/output pins, and the auxiliary_input/output_enable pins are automatically sourced by the ssn_en DFT signal.

Refer to "Example 2" on page 4399 of the SSN wrapper description for a more complex example where the child block module includes Multiplexer nodes. The secondary data paths for the instances of this child block are connected using an ExtraOutputPath wrapper.

```
set spec [create_dft_specification -sri_sib_list {ssn edt occ}]

read_config_data -in $spec -from_string {
   SSN {
    ijtag_host_interface : Sib(ssn);
    Datapath(1) {
      output_bus_width : 4;
      Connections {
        bus_clock_in : gpio[4];
        bus_data_in  : gpio[3:0];
        bus_data_out : gpio[8:5];
      }
      Pipeline(1) {
      }
      ScanHost(1) {
        output_chain_count : 4;
      }
      DesignInstance(corea_i1) {
      }
      DesignInstance(corea_i2) {
      }
      Pipeline(2) {
      }
    }
  }
  Occ {
    ijtag_host_interface : Sib(occ);
    type : parent;
    Controller(clka) {
      clock_intercept_node : clka_buf/A;
    }
    Controller(clkb) {
      clock_intercept_node : clkb_buf/A;
    }
    Controller(clkc) {
      clock_intercept_node : clkc_buf/A;
    }
  }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
    Edt {
      ijtag_host_interface : Sib(edt);
      Controller(c1) {
        longest_chain_range                  : 10, 50;
        scan_chain_count                     : 15;
        input_channel_count                  : 3;
        output_channel_count                 : 2;
        connect_bscan_segments_to_lsb_chains : off;
      }
    }
  }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ExtraOutputPath

A wrapper used to create a branch out of the SSN data path that typically ends back on the secondary bus data in the port of a Multiplexer node inside another block.

Usage

```
DftSpecification(module_name,id) {
  SSN {
    Datapath(id) {
      ExtraOutputPath {
        ijtag_host_interface : host_node_leaf_name; // default: Sib(ssn)
        output_bus_width     : int | inferred_from_source;
        Connections {
          bus_clock_in       : port_pin_name;      // default: inferred
          bus_clock_out      : port_pin_name;      // Optional
          output_datapath_id : id;                 // default: out<#>
          bus_data_out       : {- | port_pin_name, ... }; // Required
        }
        BusFrequencyDivider {
        }
        BusFrequencyMultiplier {
        }
        DesignInstance {
        }
        ExtraOutputPath {
        }
        Fifo {
        }
        Multiplexer {
        }
        OutputPipeline {
        }
        Pipeline {
        }
        Receiver1xPipeline {
        }
        ScanHost {
        }
      }
    }
  }
}
```

Description

A wrapper used to create a branch out of the SSN data path that typically ends back on the secondary bus data in port of a Multiplexer node inside another block. All child nodes that can be specified in the Datapath wrapper can be specified in the ExtraOutputPath wrapper. The bus_data_in source of the extra output path is the bus_data_out ports of the nodes in which the ExtraOutputPath wrapper is located unless the ExtraOutputPath wrapper is located directly inside the Datapath wrapper in which case the bus_data_in source of the extra output path is the bus_data_in of the Datapath wrapper itself.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The DefaultChildConfiguration wrapper is allowed here because the ExtraOutputPath wrapper supports child wrappers. The DefaultChildConfiguration wrapper is needed to maintain the inheritance through the wrapper hierarchy as explained in the DefaultChildConfiguration section.

Arguments

- ijtag_host_interface : *host_node_leaf_name*;

  A property that inherits its value from the ijtag_host_interface property in the Datapath wrapper. It has the exact same format. The default is Sib(ssn). You specify the property when you want to use a different IJTAG host interface for each node, but it is unlikely that you need to do that. To precisely control the ordering of the IJTAG network for the SSN nodes, specify the ijtag_connection_order property in each of the SSN hardware nodes.

- output_bus_width : *int* | inferred_from_source;

  A property that specifies the output bus width of the extra output path. When unspecified, it defaults to the width of the source of the ExtraOutputPath adjusted by the multiplication and division factors of the nodes found within the ExtraOutputPath wrapper. As in "Example 3" on page 4400 of the SSN wrapper description, the output_bus_width of the ExtraOutputPath wrapper found within the BusFrequencyMultiplier(from_bottom) wrapper defaults to 40 because its source is the 20 bit bus at the output of the BusFrequencyMultiplier(from_bottom) node multiplied by two because of the presence of the BusFrequencyDivider(to_top) node within it.

  You typically do not need to specify this property unless you need to specify a smaller value when you are branching out to a narrower portion of the network in order to include, for example, a legacy block that was built with a smaller bus width at a prior time.

- Connections/bus_clock_in : *port_pin_name*;

  A property used to specify the default source of the bus clock for the nodes within the ExtraOutputPath node. This bus clock is typically unspecified as it defaults automatically to the same source as the bus_clock_in of the node in which the ExtraOutputPath wrapper is located. To understand what that is, see the Connections/bus_clock_in property of the node in which the ExtraOutputPath is located.

  When the design level is chip, the port must already exist and be connected to a pad buffer cell. You may also have a boundary scan cell equipped with AuxiliaryInputData logic. When you point to such a top-level port, the tool automatically maps the connection across the pad and optional boundary scan cell.

- Connections/bus_clock_out : *port_pin_name*;

  An optional property used to specify a bus clock destination node. When unspecified, the bus_clock_out is not connected anywhere. You specify an output port name when using the tiling physical flow as in the "Example 1" on page 4395 of the SSN wrapper description.

  When the design level is chip, the port must already exist and be connected to a pad buffer cell. You may also have a boundary scan cell equipped with AuxiliaryOutputData logic.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When you point to such a top-level port, the tool automatically maps the connection across the pad and optional boundary scan cell.

- Connections/output_datapath_id : *id* ;

  An optional property that provides a unique label to the extra output data bus. Use this value when the Connections/bus_data_out property refers to top-level input ports. When this property is not specified, the ID for the value is automatically assigned as "out<#>", where <#> is a counter that starts at one and increments for each ExtraOutputData wrapper that the tool processes.

  Any ID you specify must be unique across all DataPath(<*id*>) wrappers, the Multiplexer/Connections/input_datapath_id properties, and the ExtraOutputPath/Connections/output_datapath_id properties.

- **Connections/bus_data_out** : {**-** | ***port_pin_name***,...};

  A required property used to specify the destination of the data for the data path. The specified value is used as the data out destination for the top node specified in the ExtraOutputPath wrapper. If there is no node specified in the ExtraOutputPath wrapper, the specified bus_data_out locations connect to the inferred bus_data_in location of the extra data path, which is described in the "Description" section above.

  When connecting to a bus port or pin, you can use [%d] as the index. The %d symbol will automatically be converted to "*n*-1:0" where *n* is the output bus width of the extra output path. If you want to connect to several scalar or bus segments, specify the multiple object names separated by a comma. The first element corresponds to the most significant bit of the bus. The sum of the sizes of all the objects must match the computed bus input size.

  When specified as a hyphen (-), the pin is open. Otherwise, the specified values must refer to existing pins, or to output or in/out ports that may or may not already exist in the current design. They are created if not already present when the *design_level* specified with the set_design_level command is physical_block or sub_block. When the design level is chip, the port must already exist and be connected to a pad buffer cell. You can also have a boundary scan cell equipped with AuxiliaryOutputData logic. When you point to such a top-level port, the tool automatically maps the connection across the pad and optional boundary scan cell.

  Note

  Generally, the tool makes a connection from an SSN instance to the ends of the SSN (such as bus_data_in and bus_data_out). When you specify "-" as a value to these properties, the tool skips making this connection. Instead, you can use a custom process_dft_specification.post_insertion callback procedure to customize the datapath connection. See "register_callback" on page 1586 for more information about the callback procedure.

  This custom procedure uses instance pointers for SSN nodes. For more information about retrieving the instance pointers for SSN nodes inserted during processing of the DftSpecification, refer to the section "Instantiated Nodes Section in the SSN Instrument Dictionary" on page 4393.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

Refer to "Example 1" on page 4395 of the SSN wrapper description. The ExtraOutputPath wrapper is used to branch out the data path to the blocks above and below the current block. The Multiplexer node is then used to merge those branched out paths back into the main data path.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Fifo

Controls creation of a Fifo node for cases where clock domain handoff can occur within a single physical partition.

Usage

```
DftSpecification(module_name,id) {
  SSN {
    Datapath(id) {
      Fifo(id) {
        ijtag_host_interface           : host_node_leaf_name;
                                                 // default: Sib(ssn)
        ijtag_connection_order         : int | "";
        bus_clock_period               : time;     // default: Inherited
        frequency_ratio                : int;      // required
        input_retimed                  : on | off;
        input_retiming_cell_type       : dff | latch ;
                                                 // default: Inherited
        in_clock_to_out_clock_skew     : early_or_delayed | delayed_only;
        in_clock_to_out_clock_skew_programmable : on | off;
        parent_instance                : instance_name;
        leaf_instance_name             : leaf_instance_name;

        Interface {
          bus_in_clock  : port_name; // default: Inherited
          bus_out_clock : port_name; // default: Inherited
          bus_data_in   : port_name; // default: Inherited
          bus_data_out  : port_name; // default: Inherited
          IjtagScanInterface {
            reset       : port_name; // default: Inherited
            tck         : port_name; // default: Inherited
            select      : port_name; // default: Inherited
            capture_en  : port_name; // default: Inherited
            shift_en    : port_name; // default: Inherited
            update_en   : port_name; // default: Inherited
            scan_in     : port_name; // default: Inherited
            scan_out    : port_name; // default: Inherited
          }
        }

        Connections {
          bus_in_clock_in  : port_pin_name;
          bus_out_clock_in : port_pin_name;
        }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
            BusFrequencyDivider(id) {
            }
            BusFrequencyMultiplier(id) {
            }
            DesignInstance(instance_name) {
            }
            ExtraOutputPath {
            }
            Fifo(id) {
            }
            Multiplexer(id) {
            }
            OutputPipeline(id) {
            }
            Receiver1xPipeline(id) {
            }
            Pipeline(id) {
            }
            ScanHost(id) {
            }
          }
        }
      }
    }
  }
```

Note

The frequency_ratio, input_retimed, in_clock_to_out_clock_skew, and in_clock_to_out_clock_skew_programmable properties have an impact on hardware generation. In addition, the Interface and Connections wrappers, and all properties and wrappers contained within them, have an impact on hardware generation. As a result, the Fifo wrapper also has an impact on hardware generation (in all cases, because the frequency_ratio property is required).

Description

The Fifo node is suitable for situations where BusFrequencyDivider and BusFrequencyMultiplier nodes can result in a high area overhead, particularly if you need to tolerate a high skew amount. Its design is based on the BusFrequencyDivider and BusFrequencyMultiplier nodes. It includes an input register to store input packets, similar to that of the BusFrequencyDivider, and it selects the output packet based on the phase counter value, similar to the operation of the BusFrequencyMultiplier.

The Fifo node includes the following components:

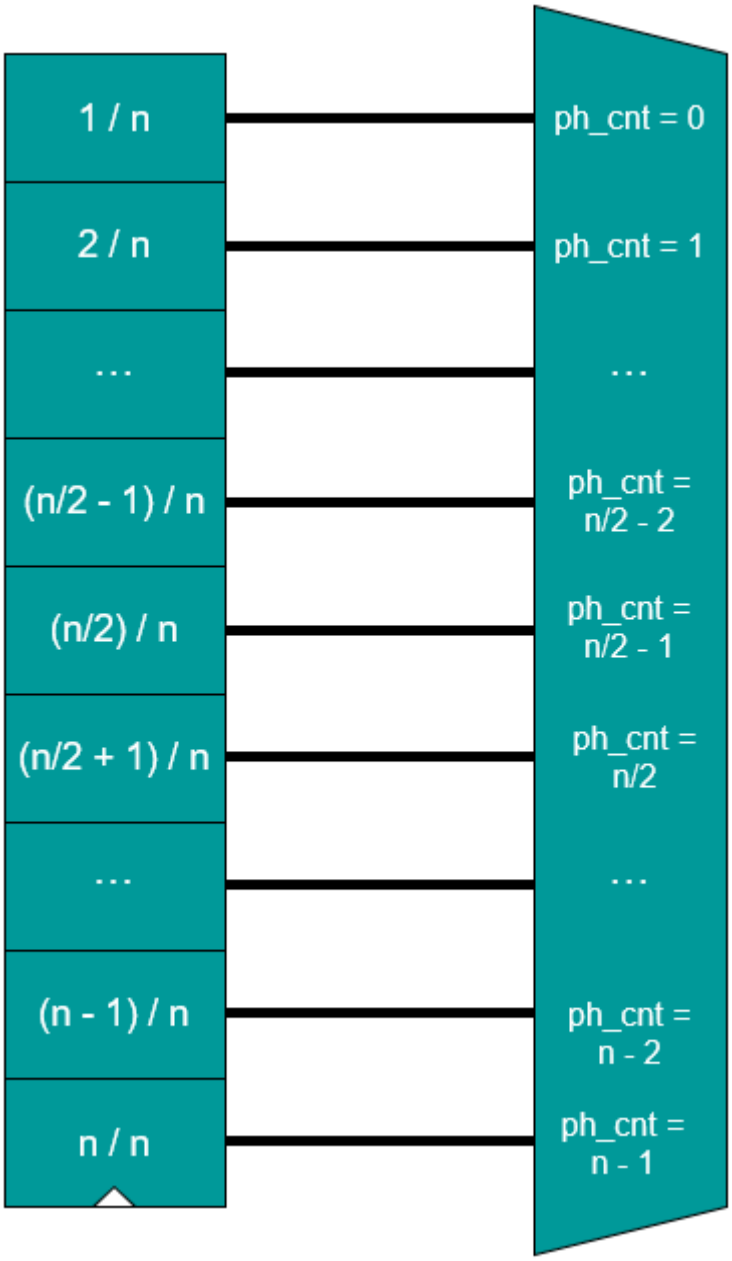
- An optional input retiming register with a size of *bus_width*.
- An input FIFO register.

  The FIFO stores the input packet in the register selected by the input phase counter.
- An output bus register with a size of *bus_width*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Two phase counters, one per clock region.
- A multiplexer that selects data from the FIFO register to send to the output ratio. The multiplexer has *frequency_ratio* inputs and selects the active input based on the value of *frequency_counter*. The multiplexer can be configured in one of two operation modes:
  - The first configuration tolerates only a delayed input clock (that is, the value of in_clock_to_out_clock_skew is delayed_only). In this configuration, it selects packets according to the phase counter value as follows:

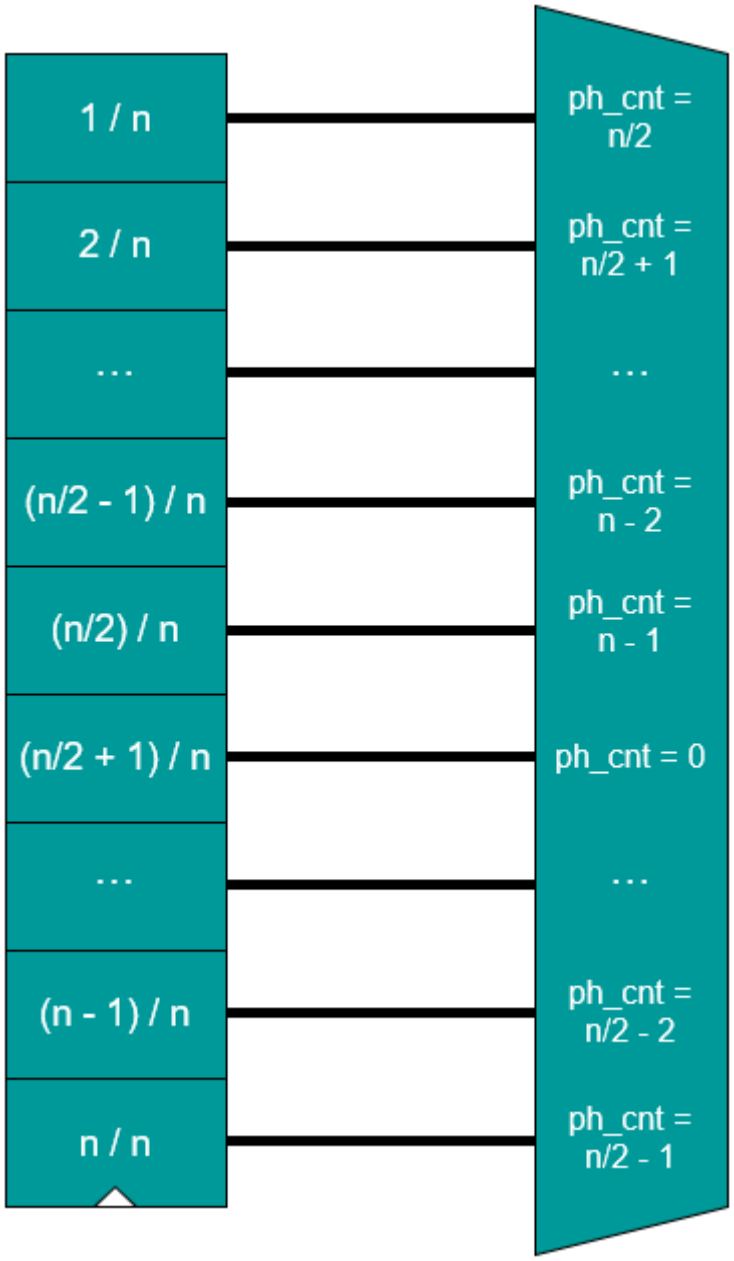
Figure 13-98. Packets per Phase Counter Value With delayed_only



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The second configuration can tolerate in_clock_to_out_clock_skew set to early_or_delayed. In this configuration, it selects packets according to the phase counter value as follows:

Figure 13-99. Packets per Phase Counter Value With early_or_delayed



- An additional multiplexer that selects the value from the preceding multiplexers, based on the TDR value, when deskew tolerance is set as programmable (default behavior). The default reset value is based on the in_clock_to_out_clock_skew property value.
- Two phase counter sync reset registers—one per clock region.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Two optional datapath register sync reset registers (generated by default)—one per clock region.
- An optional IJTAG interface with a one-bit TDR to program deskew behavior (generated by default).

The following signals are driven by the input and output clocks:

- **Input clock** — FIFO input register, input bus register reset, input phase counter reset, and input phase counter
- **Output clock** — FIFO output register, output bus register reset, output phase counter reset, and output phase counter

Note

The FIFO is designed for the situation where the input and output clock have the same period. If the clocks have different periods, it is recommended to use BusFrequencyDivider and BusFrequencyMultiplier nodes.

When a Fifo node is present, the datapath is connected with the order specified in the DftSpecification. This is the same as the connection order without the Fifo node. To enable the next nodes to infer the same clock, you must specify the Connections/bus_out_clock property. The Connections/bus_in_clock value is inferred from the previous node. If the Fifo is the first node in the current datapath, you must specify the Connections/bus_in_clock value.

The ICL model is designed as multiple pipeline stages, according to the requirements.

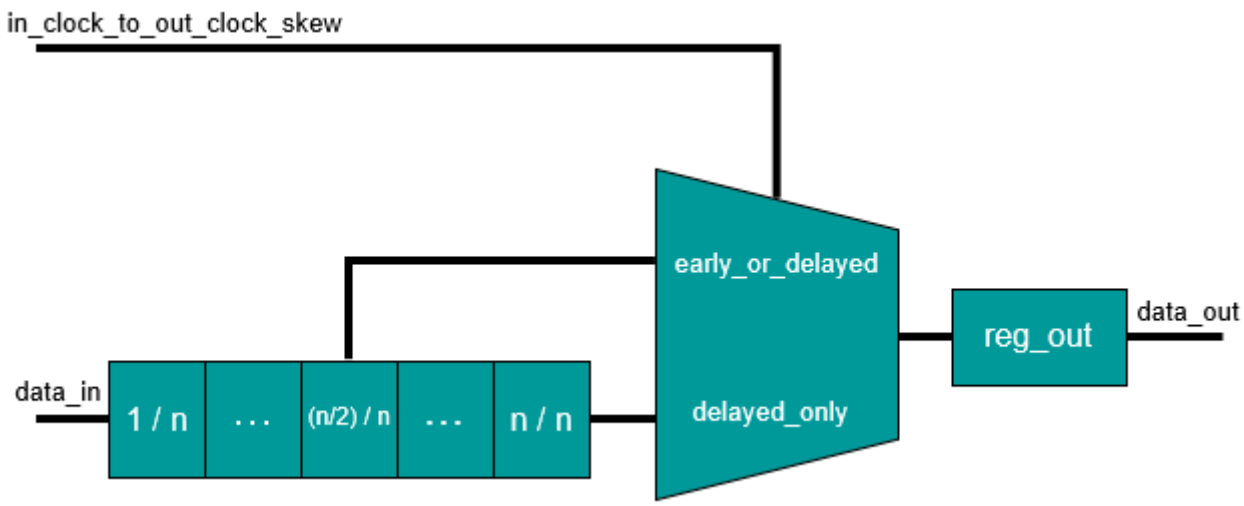
When you specify programmable skew tolerance (in_clock_to_out_clock_skew_programmable on), the tool generates a ScanRegister, DataMux, and full FIFO. Otherwise, only a FIFO of the required size is modeled. For a fixed FIFO, the pipeline stage length is generated based on the following equations:

$$fifo_reg_size_{delayed_only} = frequency_ratio$$

$$fifo_reg_size_{early_or_delayed} = frequency_ratio / 2$$

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following is a block diagram representation of a FIFO with the programmable feature:



Calculate the total FIFO pipeline stage size with the following equation:

total_fifo_node_size = fifo_reg_size + reg_out

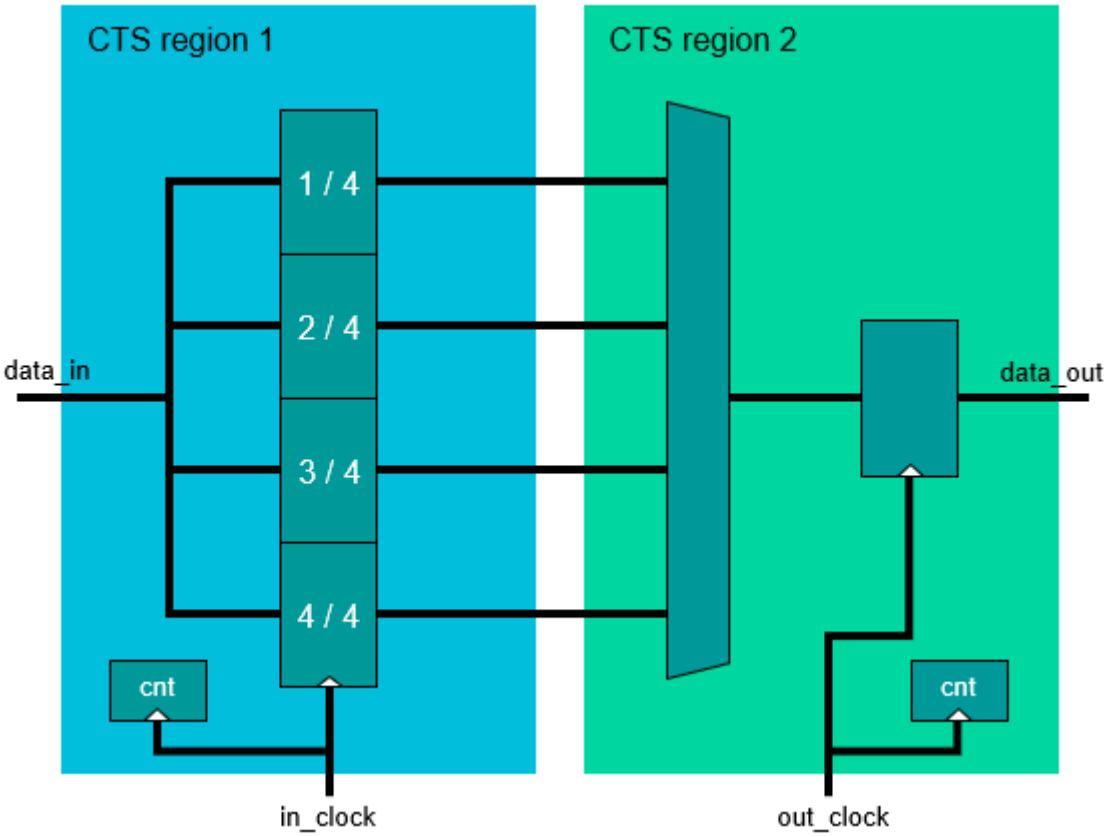
Figure 13-100 shows a FIFO based on a frequency_ratio value of four. The tool considers both early_or_delayed and delayed_only versions in terms of timing and data propagation across nodes. A Fifo node always contains as many FIFO registers as the frequency_ratio value (so, in this example, four registers). There is always a single output register. There can be one input retiming flop present before the FIFO registers if the input_retimed property is on. The following explanation does not include a description of this optional retiming flop.

In CTS region 1, the phase counter (cnt) starts counting from zero. At this value, the node stores input data in the first FIFO register. When the phase counter value is three, counting starts over from zero and the node again stores data in the first FIFO register. Both early_or_delayed and delayed_only modes have this behavior in common.

In CTS region 2, the phase counter (cnt) starts counting from zero, but the mode (early_or_delayed or delayed_only) determines which FIFO register the tool picks the data up from and passes it along to the output register for a given cycle count. Refer to Figure 13-98 on page 4451 and Figure 13-99 on page 4452 to see how the mode affects packet selection. Additionally, refer to Figure 13-101 on page 4456 and Figure 13-102 on page 4457 for waveforms that illustrate this behavior for each mode.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-100. FIFO With frequency_ratio of Four



The following waveforms illustrate early_or_delayed and delayed_only configurations with the frequency_ratio set to 4. This value means that input data is stored in registers for 4 clock cycles. There are two different clock domains present:

- **bus_in_clock** — Related to bus_data_in, and used to drive fifo_reg[0-3]
- **bus_out_clock** — Responsible for reading data from fifo_reg[0-3] and storing it in the output register

To help visualize cross-clock domains, consider the first four vectors. The write operation for both modes (early_or_delayed and delayed_only) works the same way.

The write operation works according to the following pattern:

1. During the first clock cycle, the input data is set and becomes ready for writing to the first FIFO register, fifo_reg0.
2. In the second clock cycle, V1 is stored into this register and input vector V2 is set.
3. In the third clock cycle, V2 is stored into the next FIFO register, fifo_reg1. Input vector V3 is set.

This pattern continues through cycle five, where the last FIFO register (fifo_reg3) has the fourth vector stored into it.

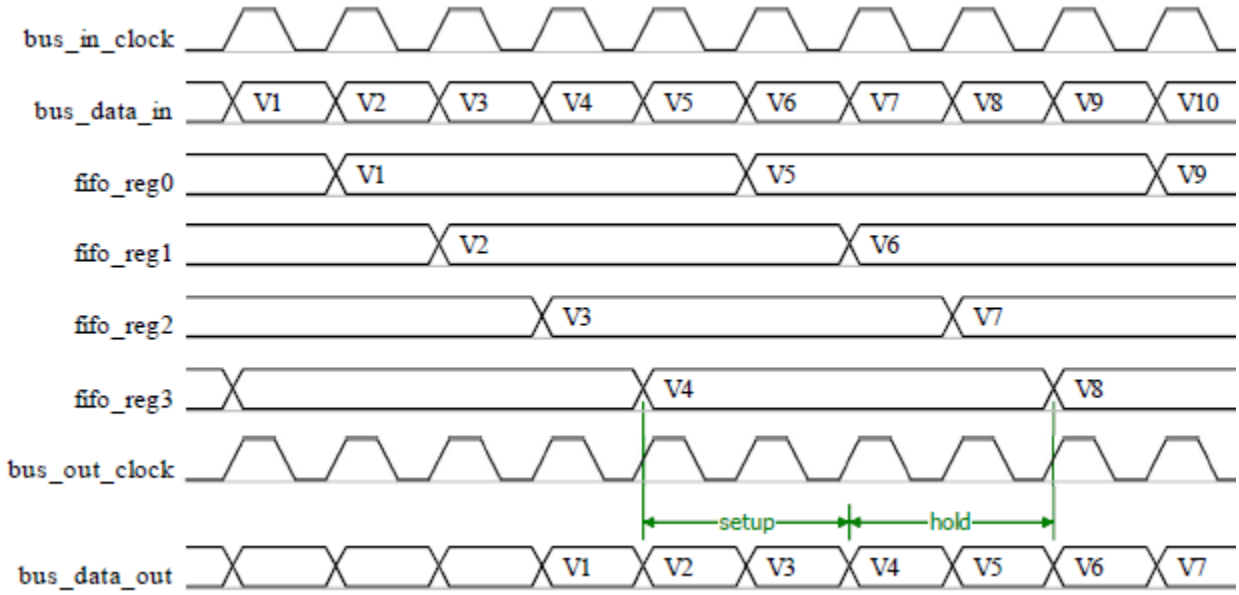
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In the sixth cycle, vector V5 is stored in fifo_reg0, and the pattern repeats.

The waveform in Figure 13-101 illustrates the read operation for the early_or_delayed configuration with the frequency_ratio set to 4. The read operation works on bus_clock out. For simplicity, consider only vector V4. When the data for V4 is stored in fifo_reg3 during the fifth clock cycle, it becomes available for the next four clock cycles. To perform deskew for each direction, the early or delayed data must be strobed after two clock cycles—that is, during the seventh clock cycle, in the middle of the vector "eye." This strobing point enables the bus_in_clock to be early or delayed by up to two clock cycles and still get a proper data read from fifo_reg3, to be stored into the output register, bus_data_out. Thus, this FIFO configuration with a frequency ratio of four behaves like a three-stage pipeline: two clock cycles for the FIFO register and one clock cycle for the output register. This is why vector V1 appears on bus_out after three input clock cycles. For vector V4, these are cycles five, six, and seven.

The pipeline stage size is different for different frequency ratio configurations.

Figure 13-101. Waveform for FIFO early_or_delayed Configuration



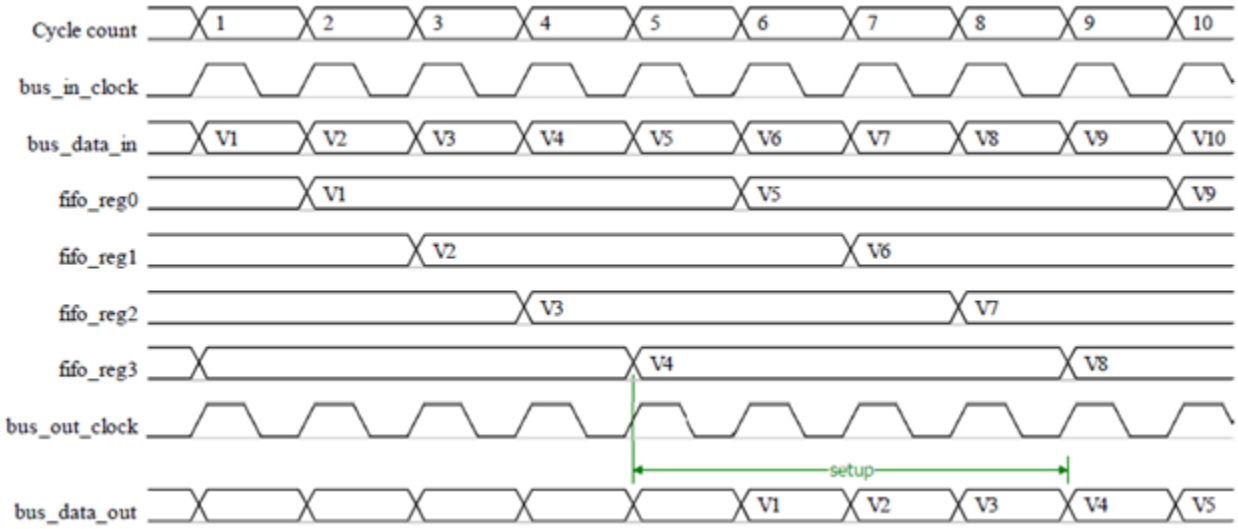
The waveform in Figure 13-102 illustrates the read operation for the delayed_only configuration with the frequency_ratio set to 4. The read operation works on bus_clock out. For simplicity, consider only vector V4. When the data for V4 is stored in fifo_reg3 during the fifth clock cycle, it becomes available for the next four clock cycles. To perform deskew for only one direction (the input clock can be delayed compared to the output clock), the data must be strobed after four clock cycles—that is, during the ninth clock cycle, at the end of the vector "eye." This strobing point enables the bus_in_clock to be delayed by up to four clock cycles and still get a proper data read from fifo_reg3, to be stored into the output register, bus_data_out. Thus, this FIFO configuration with a frequency ratio of four behaves like a five-stage pipeline: four clock cycles for the FIFO register and one clock cycle for the output register. This is why

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

vector V1 appears on bus_out after five input clock cycles. For vector V4, these are cycles five through nine.

The pipeline stage size is different for different frequency ratio configurations.

Figure 13-102. Waveform for FIFO delayed_only Configuration



SDC Extraction and FIFO Nodes

For SDC extraction, it is important to know which clocks are associated with the first sequential elements on the SSN datapath, and which clocks are associated with the last. The tool derives this information from the ICL description of the SSN. For most cases, it is possible to unambiguously identify the clocks, because there is usually only a single SSN clock input on the ICL instance that contains the ICL DataRegister representing the sequential elements in question.

However, when SSN FIFO nodes are present, there are two clocks and multiple ICL DataRegisters in the ICL instance. The ICL does not include a standardized way to associate DataRegisters with ClockPorts. Tessent Shell approaches this association question in the following way:

- If the “tessent_ssn_clocks” attribute is set on the DataRegister and its value is the name of a ClockPort in the same module, the DataRegister is assumed to be associated with this clock. The ICL for the FIFOs Tessent Shell generates during DFT insertion already contains those attributes. The attribute must exist in the ICL file. You cannot set it using the set_attribute_value command.
- If the “tessent_ssn_clock” attribute does not exist (for example, if the FIFO ICL has been created by a version of Tessent Shell that did not support this attribute or has been created by a third-party tool), and the following are both true:
    - Exactly one SSN ClockPort has a name starting with “in_”, containing “_in_”, or ending with “_in”

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

  - Exactly one SSN ClockPort has a name starting with “out_”, containing “_out_”, or ending with “_out”

  then the ClockPort containing “in” is assumed to be the relevant one for the SDC information for the SSN bus data inputs (if the FIFO is the first building block on the SSN datapath) and the ClockPort containing “out” is assumed to be the relevant one for the SDC information for the SSN bus data outputs (if the FIFO is the last building block on the SSN datapath).

- If neither of these is the case, you must manually add the tessent_ssn_clocks attribute to the DataRegister specifications of the FIFO ICL. Each DataRegister in the FIFO ICL should contain this attribute with the name of the clock it is associated with.

Arguments

- ijtag_host_interface : *host_node_leaf_name* ;

  A property that inherits its value from the ijtag_host_interface property in the Datapath wrapper. It has the exact same format. The default is Sib(ssn). Specify it here when you want to use a different IJTAG host interface for each node; for most cases, the default value is sufficient. To precisely control the ordering of the IJTAG network for the SSN nodes, specify the ijtag_connection_order property in each of the SSN hardware nodes.

- ijtag_connection_order : *int* | “”;

  An optional property that controls the connection order for the IJTAG scan interface of this node relative to the others sharing a common ijtag_host_interface value. When left unspecified, it defaults to a null string, which means that the connection order for the IJTAG scan interface matches the relative order of the node with respect to the others in the DFT specification. The IJTAG scan chain gets connected from the bottom node to the top node in the SSN wrapper. If you specify an integer value for one node, then you must specify it for all other nodes having the same ijtag_host_interface value and the integer value must be unique for each node. The node with the smallest ijtag_connection_order value will be closest to tdo.

- frequency_ratio : *int* ;

  A required property that specifies the maximum pipeline stage length in the FIFO for storing input data. This directly reflects the skew tolerance: that is, for a frequency_ratio value of two, the Fifo node tolerates up to one clock pulse in each direction for early_or_delayed mode or up to two clock pulses for delayed_only mode. For a value of four, it tolerates up to two early_or_delayed pulses or four delayed_only pulses, and so forth. This value must be an even integer from 2 through 16, inclusive.

- input_retimed : on | off ;

  An optional property that specifies the insertion of a negative edge retiming flip-flop stage on the bus data inputs. Use this property when implementing a source synchronous timing interface to the Fifo node, as described in Example 3 of the section “BusFrequencyDivider” on page 4420.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- input_retiming_cell_type : dff | latch ;

  A property that specifies the default input retiming cell type. The default value is inherited from the DefaultChildConfiguration wrapper in the ancestry of the Fifo node. This property has an effect only when the input_retimed property is set to on.

- in_clock_to_out_clock_skew : early_or_delayed | delayed_only ;

  An optional property that specifies skew tolerance. When you specify early_or_delayed, the Fifo node tolerates an input clock that is earlier in phase than the output clock or is delayed. The phase can be shifted 180 degrees (180 × *frequency_ratio*) in each direction. When you specify delayed_only, the tool accepts only an input clock delayed relative to the output clock. In this case, the tool accepts 360 degrees (360 × *frequency_ratio*) of skew.

- in_clock_to_out_clock_skew_programmable : on | off ;

  An optional property that enables you to program the skew tolerance. If you need to change the default hardware skew tolerance later, use the programmable version. When you specify this property as "on" (default value), the tool generates the Fifo node with an IJTAG interface and a one-bit TDR to control the skew tolerance behavior. Write a one to the TDR to enable delayed_only skew tolerance and a zero to enable early_or_delayed. The in_clock_to_out_clock_skew property sets the default skew tolerance after a reset. When you enable programmable skew tolerance, the tool generates two multiplexers to select data from the FIFO register and uses one additional mux to select output from one of the other two and pass it to the output register. If you specify this property as "off", the tool generates only fixed skew tolerance and you cannot change it during pattern execution.

  Note

  This property enables you to change only the skew tolerance between early_or_delayed and delayed_only. It does not enable you to change the FIFO size.

- parent_instance : *instance_name* ;

  An optional property that specifies the parent instance in which the given node should be instantiated into. When unspecified, the node is instantiated directly into the current design.

- leaf_instance_name : *leaf_instance_name* ;

  An optional property that specifies the leaf instance name of the node. When unspecified, the leaf instance name defaults to the module name with the "_inst" suffix appended to it.

- Interface

  The properties in the Interface wrapper enable you to change the names of all the ports on the node. Refer to the Interface wrapper of the DefaultChildConfiguration wrapper to see the default value of each port name. If the property within the local Interface wrapper is not specified, the value is inherited from the Interface wrapper in the first DefaultChildConfiguration wrapper found in the ancestry of the current node.

- Connections/bus_in_clock_in : *port_pin_name* ;

  An optional property that specifies the source of the bus clock from the data input side for the Fifo node. Typically, you do not specify this property so that the tool infers the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Connection automatically. In this case, if a DesignInstance wrapper appears below the FIFO wrapper and this instance was created without an output bus clock port, the bus clock input of the Fifo node connects to it unless you set the ignore_bus_clock_out property to “off” in the DesignInstance/Connections wrapper. If a node below the Fifo wrapper has the Connections/bus_clock_in specified, the Fifo node will also use this value.

- Connections/bus_out_clock_in : *port_pin_name* ;

  A required property that specifies the source of the bus clock from the data output side for the Fifo node. Nodes above the Fifo node inherit this clock source.

  When the design level is “chip”, this port must already exist and be connected to a pad buffer cell. You may also have a boundary scan cell equipped with AuxiliaryInputData logic. When you point to such a top-level port, the tool automatically maps the connection across the pad and optional boundary scan cell.

Examples

Example 1

The following example inserts three instances of the Fifo node. It demonstrates how each pipeline and scan host infers the clock from the Fifo node.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
SSN {
  Datapath(1) {
    output_bus_width : 32;
    bus_register_reset : on;
    Connections {
      bus_clock_in : fifo_out_clock, fifo2_out_clock, fifo3_out_clock, \
          fifo_in_clock;
    }
    Fifo(3) {
      frequency_ratio : 4;
      input_retimed : on;
      in_clock_to_out_clock_skew : delayed_only;
      in_clock_to_out_clock_skew_programmable : off;
      Connections {
        bus_out_clock_in : fifo3_out_clock;
      }
    }
    Pipeline(2) {
    }
    ScanHost(1) {
      input_chain_count : 8;
      output_chain_count : 8;
    }
    Fifo(2) {
      frequency_ratio : 4;
      input_retimed : on;
      in_clock_to_out_clock_skew : delayed_only;
      in_clock_to_out_clock_skew_programmable : off;
      Connections {
        bus_out_clock_in : fifo2_out_clock;
      }
    }
    Pipeline(1) {
    }
    Fifo(1) {
      frequency_ratio : 2;
      input_retimed : on;
      in_clock_to_out_clock_skew : delayed_only;
      in_clock_to_out_clock_skew_programmable : on;
      Connections {
        bus_out_clock_in : fifo_out_clock;
        bus_in_clock_in : fifo_in_clock;
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This specification generates the following programmable Fifo node with PDL:

```
//    Processing of SSN
//      Generating design files for SSN Fifo module corea_fifo_insertion_1_tessent_ssn_fifo_3
//        Verilog RTL : <path_filtered>/corea_fifo_insertion_1_tessent_ssn_fifo_3.v
//        IJTAG ICL   : <path_filtered>/corea_fifo_insertion_1_tessent_ssn_fifo_3.icl
//      Generating design files for SSN Fifo module corea_fifo_insertion_1_tessent_ssn_fifo_2
//        Verilog RTL : <path_filtered>/corea_fifo_insertion_1_tessent_ssn_fifo_2.v
//        IJTAG ICL   : <path_filtered>/corea_fifo_insertion_1_tessent_ssn_fifo_2.icl
//      Generating design files for SSN Fifo module corea_fifo_insertion_1_tessent_ssn_fifo_1
//        Verilog RTL : <path_filtered>/corea_fifo_insertion_1_tessent_ssn_fifo_1.v
//        IJTAG ICL   : <path_filtered>/corea_fifo_insertion_1_tessent_ssn_fifo_1.icl
//        PDL         : <path_filtered>/corea_fifo_insertion_1_tessent_ssn_fifo_1.pdl
```

In this example, the following inferences occur:

- The Pipeline(1) bus_clock_in and Fifo(2) bus_in_clock_in infer the clock from the Fifo(1) bus_out_clock_in connection, which is connected to the fifo_out_clock primary input.
- The ScanHost(1) bus_clock_in, Pipeline(2) bus_clock_in, and Fifo(3) bus_in_clock_in infer the clock from the Fifo(2) bus_out_clock_in, which is connected to the fifo2_out_clock primary input.

Example 2

This example shows how to reconfigure the programmable Fifo node shown in Example 1 during pattern generation.

```
set_test_setup_icall {corea_fifo_insertion_1_tessent_ssn_fifo_1_inst.setup \
  in_clock_to_out_clock_skew_delayed_only 0}
```

- Writing a 1 to the in_clock_to_out_clock_skew_delayed_only register programs the node to use the delayed_only mode.
- Writing a 0 to the in_clock_to_out_clock_skew_delayed_only register programs the node to use the early_or_delayed mode.

You can also perform this reconfiguration with an iCall.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Multiplexer

A wrapper used to specify the creation of a SSN multiplexer node.

Usage

```
DftSpecification(module_name,id) {
  SSN {
    Datapath(id) {
      Multiplexer(id) {
        ijtag_host_interface      : host_node_leaf_name;
                                                  // default: Sib(ssn)
        ijtag_connection_order    : int | "";
        bus_clock_period          : time;                // def: inherited
        secondary_input_bus_width : width | same as output bus width;
        include_pipeline_stage    : on | off;
        include_clock_mux         : on | off | auto;
        parent_instance           : instance_name;
        leaf_instance_name        : leaf_instance_name;
        mux_select                : internal | external | auto;
        DefaultChildConfiguration {
        }

        Interface {
          bus_clock             : port_name; // def: inherited
          bus_clock_out         : port_name; // def: inherited
          bus_data_in           : port_name; // def: inherited
          secondary_bus_data_in : port_name; // def: inherited
          secondary_bus_clock   : port_name; // def: inherited
          bus_data_out          : port_name; // def: inherited
          select_in             : port_name; // def: inherited
          select_out            : port_name; // def: inherited
          IjtagScanInterface {
            reset               : port_name; // def: inherited
            tck                 : port_name; // def: inherited
            select              : port_name; // def: inherited
            capture_en          : port_name; // def: inherited
            shift_en            : port_name; // def: inherited
            update_en           : port_name; // def: inherited
            scan_in             : port_name; // def: inherited
            scan_out            : port_name; // def: inherited
          }
        }

        Connections {
          bus_clock_in            : port_pin_name;   // def: ssn_bus_clock
          input_datapath_id       : id;              // def: in<#>
          secondary_bus_data_in   : (- | port_pin_name, ...}; // def: ""
          select_in               : {- | port_pin_name };     // def: ""
          secondary_bus_clock_in  : port_pin_name;
        }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
            BusFrequencyDivider(id) {
            }
            BusFrequencyMultiplier(id) {
            }
            DesignInstance(instance_name) {
            }
            ExtraOutputPath {
            }
            Multiplexer {
            }
            Pipeline(id) {
            }
            Receiver1xPipeline(id) {
            }
            ScanHost(id) {
            }
          }
        }
      }
    }
```

Description

This wrapper is used to create and optionally insert a SSN node of type Multiplexer.

You use this node when you want to control the inclusion and exclusion of a section of a Datapath. In Figure 13-83 on page 4398 found in Example 1 of the SSN wrapper description, you can see how the Multiplexer node is used to provide conditional access to the data path going into the instances above and below the current block. The main data path goes sideways, but the multiplexer nodes can be programmed to include the data path of the bottom and/or the upper instances. You could have chosen to always daisy chain the data path through the upper and lower instances. It would have had limited impact on the overall test time as the added latency through the SSN data path is only felt once at the beginning of a pattern set to fill in the data path and once at the end of a pattern set to flush out the data path. The main reason to allow bypassing sections of the data path is in case you have massive failures in one core, and you still want to be able to test the other cores because you do core harvesting and can still use a chip with some defective cores.

The DefaultChildConfiguration wrapper is allowed here because the Multiplexer wrapper supports child wrappers. The DefaultChildConfiguration wrapper is needed to maintain the inheritance through the wrapper hierarchy as explained in the DefaultChildConfiguration section.

Arguments

- ijtag_host_interface : *host_node_leaf_name*;

  A property that inherits its value from the ijtag_host_interface property in the Datapath wrapper. The default is Sib(ssn). You specify it here when you want to use a different IJTAG host interface for each node, but it is unlikely that you need to do that. To precisely control the ordering of the IJTAG network for the SSN nodes, specify the ijtag_connection_order property in each of the SSN hardware nodes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ijtag_connection_order : *int* | “”;

  An optional property that controls the connection order for the IJTAG scan interface of this node relative to the others sharing a common ijtag_host_interface value. When left unspecified, it defaults to a null string, which means that the connection order for the IJTAG scan interface matches the relative order of the node with respect to the others in the SSN wrapper. If you specify an integer value for one node, then you must specify it for all other nodes having the same ijtag_host_interface value, and the integer value must be unique for each node. The node with the smallest ijtag_connection_order value will be closest to tdo.

  If you do not specify the property, the IJTAG chain ordering follows the SSN data path ordering. For example, in “Example 2” on page 4468, the property is not specified, and the IJTAG scan chain is stitched from smux1_1 through smux1_2 and smux2_1 to finish on smux2_2.

- bus_clock_period : *time*;

  This property is used to set the bus clock period to be used by the run_synthesis command when synthesizing the SSN nodes in isolation. When unspecified, it inherits its value from the DefaultChildConfiguration wrapper in its ancestry wrappers. When synthesizing the SSN nodes with the rest of the functional logic, the SDC procs created by the extract_sdc command, which is automatically called by the extract_icl command, is used. The ssn_bus_clock_period for the entire physical block is configured using the set_load_unload_timing_options command. Refer to “ScanHost Scan Interface Operation and Timing Requirements” on page 4498 in the ScanHost page for a description of how to keep a common definition of those settings for both SDC and pattern generation.

- secondary_input_bus_width : *int* | same_as_output_bus_width;

  A property that specifies the width of the secondary input bus. The value is allowed to be smaller or equal to the output bus width of the node, which it inherits from the nodes above it, or the output_bus_width property of the data path when it is the top node. When unspecified, it defaults to “same_as_output_bus_width”.

- include_pipeline_stage : on | off;

  This property specifies whether or not the SSN Multiplexer node includes a pipeline stage after the multiplexer. When set to on, the clock rate must match the effective data rate through the multiplexer, since the pipeline stage has no update_phase. If update_phase is required, set include_pipeline_stage to off and add a Pipeline node above it with the required update_phase value.

  When the Multiplexer node with no pipeline stage and more than one bus_clock_in connection is within a Datapath wrapper, the Pipeline node specified above it must define its bus_clock_in connection, as it cannot inherit it from the Multiplexer node below, since it has no clocks.

- include_clock_mux : on | off | auto;

  An optional property that requests that the multiplexer perform multiplexing not only on the data from a different path but also on the clock. This is needed when the bus_clock_in

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

signals of the nodes sourcing the two data inputs of the multiplexer do not have the same source.

The default value of "auto" is inferred as "on" when the Connections/secondary_bus_clock_in property is specified, as demonstrated in Example 3 of this topic.

- parent_instance : *instance_name*;

  A property that specifies the parent instance in which the given node should be instantiated into. When unspecified, the node is instantiated directly into the current design.

- leaf_instance_name : *leaf_instance_name*;

  A property that specifies the leaf instance name of the node. When unspecified, the leaf instance name defaults to the module name with the "_inst" suffix appended to it.

- mux_select : internal | external | auto;

  A property that specifies whether the mux select for the multiplexer mode uses an internal or external source. The default "auto" value uses an external source when you specify the Connections/select_in property; otherwise, it uses an internal source. When you specify an external mux select, either using this property or implicitly with the Connections/select_in property, a DataInPort of the module controls the mux select, and you must connect it to a valid IJTAG data source. In this case, the multiplexer no longer has an internal scan register that controls the select.

- Interface/*

  The properties in the Interface wrapper enable you to change the names of all the ports on the node. Refer to the Interface wrapper of the DefaultChildConfiguration wrapper to see the default value of each port name. If the property within the local Interface wrapper is not specified, the value is inherited from the Interface wrapper in the first DefaultChildConfiguration wrapper found in the ancestry of the current node.

- Connections/bus_clock_in : port_pin_name;

  A property that can be used to specify the source of the bus clock for the Multiplexer node. This connection is typically inferred automatically. If unspecified and a DesignInstance wrapper is found below the Multiplexer wrapper and this instance was created with an output bus clock port, the bus clock input of the Multiplexer node will connect to it unless the ignore_bus_clock_out property is set to off in the DesignInstance/Connections wrapper. If a node below the Multiplexer wrapper has its Connection/bus_clock_in property specified, it will be used by the Multiplexer node also. Otherwise, it will connect to the object specified by the Datapath/Connections/bus_clock_in property, which defaults to a port named "ssn_bus_clock". You typically will never use this property as the default value is correct in all cases.

  When the design level is chip, the port must already exist and be connected to a pad buffer cell. You can also have a boundary scan cell equipped with AuxiliaryInputData logic. When you point to such a top-level port, the tool automatically maps the connection across the pad and optional boundary scan cell.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Connections/input_datapath_id : *id* ;

  An optional property that provides a unique label to the secondary input data bus. Use this value when the Connections/secondary_bus_data_in property refers to top-level input ports. When this property is not specified, the ID for the value is automatically assigned as "in<#>", where <#> is a counter that starts at one and increments for each Multiplexer wrapper that the tool processes.

  Any ID you specify must be unique across all DataPath(<*id*>) wrappers, the Multiplexer/Connections/input_datapath_id properties, and the ExtraOutputPath/Connections/output_datapath_id properties.

- Connections/secondary_bus_data_in : {- | *port_pin_name*, ...} ;

  A optional property used to specify the source of the data path, which terminates on the secondary input bus of the Multiplexer node. The total width of all specified objects must equal to the secondary_input_bus_width value.

  When specified as a hyphen (-), the pin is tied low. Otherwise, the specified values must refer to existing pins, or to an input or inout port that may or may not already exist in the current design. It is created if not already present when the design_level specified with the set_design_level command is physical_block or sub_block. When the design level is chip, the port must already exist and be connected to a pad buffer cell. You can also have a boundary scan cell equipped with AuxiliaryInputData logic. When you point to such a top-level port, the tool automatically maps the connection across the pad and optional boundary scan cell.

  Note

  Generally, the tool makes a connection from an SSN instance to the ends of the SSN (such as bus_data_in and bus_data_out). When you specify "-" as a value to these properties, the tool skips making this connection. Instead, you can use a custom process_dft_specification.post_insertion callback procedure to customize the datapath connection. See "register_callback" on page 1586 for more information about the callback procedure.

  This custom procedure uses instance pointers for SSN nodes. For more information about retrieving the instance pointers for SSN nodes inserted during processing of the DftSpecification, refer to the section "Instantiated Nodes Section in the SSN Instrument Dictionary" on page 4393.

  When connecting to a bus port or pin, you can use [%d] as the index. The %d symbol will automatically be converted to "*n*-1:0" where *n* is the secondary_input_bus_width value.

  When unspecified, the value defaults to the same source as the bus_data_in port as shown in Example 2, following. The property cannot be left unspecified when there are no child nodes within the Multiplexer wrapper.

- Connections/secondary_bus_clock_in : *port_pin_name*;

  An optional property used to specify the source of the secondary_bus_clock input port. The tool reports an error if this property is specified and the include_clock_mux property is

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

specified to off. It also reports an error if this property is not specified and the include_clock_mux property is specified to “on”.

- Connections/select_in : {- | *port_pin_name*};

  An optional property used to specify the source of an external mux select for the multiplexer node. By default, this value is “” (null). Use the mux_select property to specify whether to use an internal or external mux select. If you specify this property, the default “auto” value for the mux_select property uses the value of this property for the external mux select. This property is mandatory when you specify when you specify mux_select as external, and it is not permitted when you select mux_select as internal. If you specify a hyphen (-) for this property, the pin is tied low.

Note

Generally, the tool makes a connection from an SSN instance to the ends of the SSN (such as bus_data_in and bus_data_out). When you specify “-” as a value to these properties, the tool skips making this connection. Instead, you can use a custom process_dft_specification.post_insertion callback procedure to customize the datapath connection. See “register_callback” on page 1586 for more information about the callback procedure.

This custom procedure uses instance pointers for SSN nodes. For more information about retrieving the instance pointers for SSN nodes inserted during processing of the DftSpecification, refer to the section “Instantiated Nodes Section in the SSN Instrument Dictionary” on page 4393.

Examples

Example 1

Refer to “Example 1” on page 4395 and “Example 2” on page 4399 in the SSN wrapper description. It shows how to use the Multiplexer node to build a middle block that has the ability to include or exclude the data paths from blocks located above and below it. This example illustrates a use model best suited for the tiling physical flow

Example 2

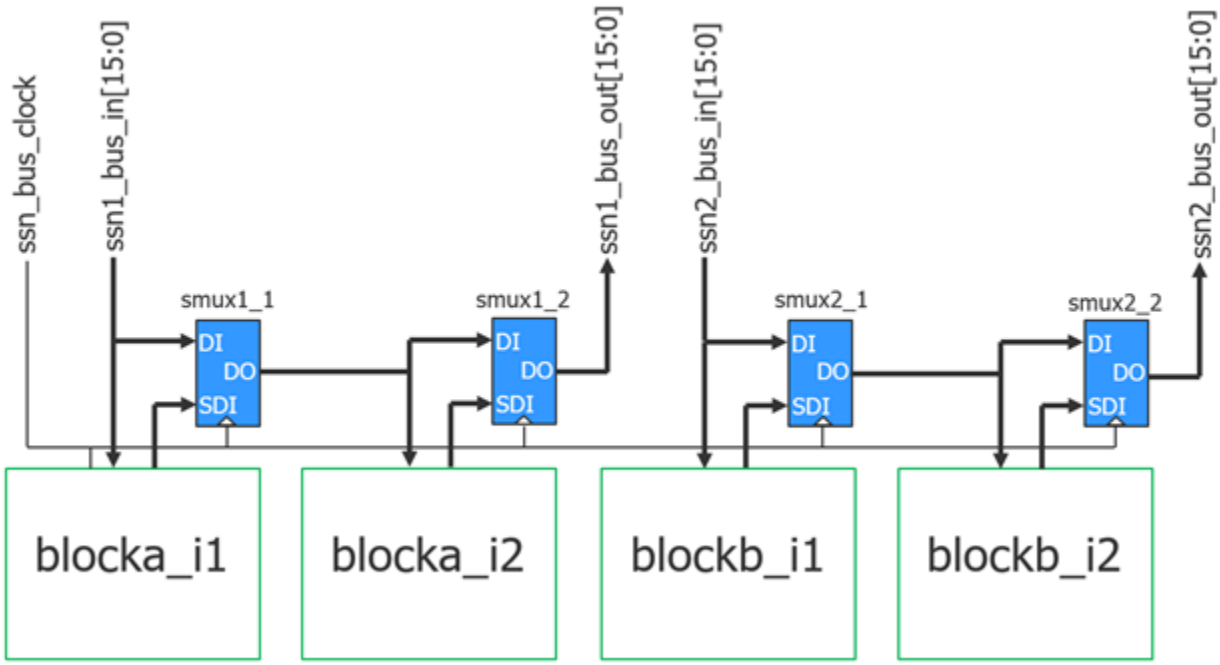
This example illustrates a use model best suited for the non-tiling flow. The multiplexer nodes are inserted at the top level to allow bypassing the child blocks. Refer to “Example 3” on page 4400 in the SSN wrapper description if you are following a tiling physical flow or even a near abutment flow. Notice that the secondary_bus_data_in property is not specified in the Connections wrappers of the Multiplexer nodes. This means the connection defaults to the same place as the bus_data_in port in such a way that the nodes found within the Multiplexer wrapper share the same source as the multiplexer when it is closed. This is illustrated in Figure 13-103.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set spec [create_dft_specification -sri_sib_list ssn]

read_config_data -in $spec -from_string {
  SSN {
    Datapath(1) {
      output_bus_width : 16;
      Connections {
        bus_data_in  : ssn1_bus_in[%d];
        bus_data_out : ssn1_bus_out[%d];
      }
      Multiplexer(1_2) {
        DesignInstance(blocka_i2) {
        }
      }
      Multiplexer(1_1) {
        DesignInstance(blocka_i1) {
        }
      }
    }


    Datapath(2) {
      output_bus_width : 16;
      Connections {
        bus_data_in  : ssn2_bus_in[%d];
        bus_data_out : ssn2_bus_out[%d];
      }
      Multiplexer(2_2) {
        DesignInstance(blockb_i2) {
        }
      }
      Multiplexer(2_1) {
        DesignInstance(blockb_i1) {
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-103. Multiplexer Usage Example



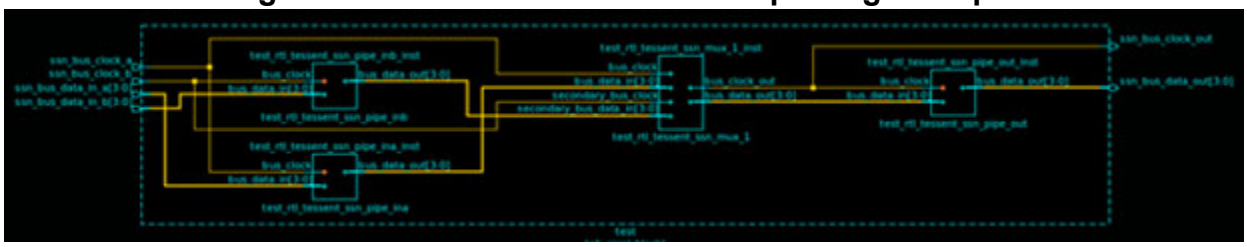
Example 3

The following example shows how you can use the Multiplexer node both to switch the data of a datapath and the clock source. The resulting logic from the following DftSpecification appears in the schematic in Figure 13-104.

```
set spec [create_dft_specification -sri_sib_list ssn]
read_config_data -in $spec -from_string {
  SSN {
    Datapath(1) {
      output_bus_width : 4;
      Connections {
        bus_clock_in  : ssn_bus_clock_a;
        bus_clock_out : ssn_bus_clock_out;
        bus_data_in   : ssn_bus_data_in_a[3:0];
        bus_data_out  : ssn_bus_data_out[3:0];
      }
      Pipeline(out) {
      }
      Multiplexer(1) {
        Connections {
          secondary_bus_data_in : ssn_bus_data_in_b[3:0];
          secondary_bus_clock_in : ssn_bus_clock_b;
        }
        Pipeline(inb) {
        }
      }
      Pipeline(ina) {
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-104. Data and Clock multiplexing example



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

OutputPipeline

A wrapper used to specify the creation of an output pipeline node.

Usage

```
DftSpecification(module_name,id) {
  SSN {
    Datapath(id) {
      OutputPipeline(id) {
        ijtag_host_interface    : host_node_leaf_name;
                                                        // default: Sib(ssn)
        ijtag_connection_order  : int | "";
        bus_clock_period        : time ;                // default: 2.5ns
        frequency_ratio         : int | auto;
        parent_instance         : instance_name;
        leaf_instance_name      : leaf_instance_name;
        Interface {
          bus_clock    : port_name; // default: bus_clock
          bus_data_in  : port_name; // default: bus_data_in
          bus_data_out : port_name; // default: bus_data_out
          IjtagScanInterface {
            reset      : port_name; // default: ijtag_reset
            tck        : port_name; // default: ijtag_tck
            select     : port_name; // default: ijtag_sel
            capture_en : port_name; // default: ijtag_ce
            shift_en   : port_name; // default: ijtag_se
            update_en  : port_name; // default: ijtag_ue
            scan_in    : port_name; // default: ijtag_si
            scan_out   : port_name; // default: ijtag_so
          }
        }
        Connections {
          bus_clock_in : port_pin_name;
        }
      }
    }
  }
}
```

Description

This wrapper is used to create and optionally insert an SSN node of type output pipeline. This node is used to inject a pipeline stage between the BusFrequencyDivider and the output ports of the device to minimize the loop timing path described in "Using the BusFrequencyDivider to Enable Faster Internal Bus Clock Frequency" on page 4423.

Use the OutputPipeline node to instruct the CTS flow to use an early branch of the clock tree for the flip-flops within the node. The OutputPipeline node has a persistent clock buffer on the bus_clock_in port to enable you to attach the CTS constraints to it. This is why you would use an OutputPipeline node instead of a normal Pipeline node. When you use an early clock branch to source the clock of the OutputPipeline node, you typically cannot drive any nodes after it, because you no longer meet the hold time requirement to those nodes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The phase counter registers within the OutputPipeline node are reset asynchronously by the ijtag_reset signal. The update_en pulse also resets the phase counters, enabling you to run several SSN pattern sets in series without resetting the IJTAG network. You can choose to make the ijtag_clock and ijtag_ue ports inactive, as long as all the SSN clocks in the active network are kept in lockstep from the time the ijtag_reset is released.

Arguments

- ijtag_host_interface : *host_node_leaf_name*;

  A property that inherits its value from the ijtag_host_interface property in the Datapath wrapper. The default is Sib(ssn). You specify it here when you want to use a different IJTAG host interface for each node, but it is unlikely that you need to do that. To precisely control the ordering of the IJTAG network for the SSN nodes, specify the ijtag_connection_order property in each of the SSN hardware nodes.

- ijtag_connection_order : *int* | “”;

  An optional property that controls the connection order for the IJTAG scan interface of this node relative to the others sharing a common ijtag_host_interface value. When left unspecified, it defaults to a null string, which means that the connection order for the IJTAG scan interface matches the relative order of the node with respect to the others in the SSN wrapper. If you specify an integer value for one node, then you must specify it for all other nodes having the same ijtag_host_interface value, and the integer value must be unique for each node. The node with the smallest ijtag_connection_order value will be closest to TDO.

  If you do not specify the property, the IJTAG chain ordering follows the SSN data path ordering. For example, in Example 2 of the Multiplexer section, the property is not specified, and the IJTAG scan chain is stitched from smux1_1 through smux1_2 and smux2_1 to finish on smux2_2.

- frequency_ratio : *int* | auto;

  A property that is used the specify the frequency ratio between the data and the clock. When an OutputPipeline node follows a BusFrequencyDivider node, it must hold the data *n*-1 out of *n* cycles to match the frequency_ratio value of the BusFrequencyDivider node. When the OutputPipeline node is in a Datapath wrapper that also contains the BusFrequencyDivider node, the auto value automatically infers the proper value. The only time you need to specify this property explicitly is if you are inserting the OutputPipeline node in a block by itself, and it will follow a BusFrequencyDivider node when assembled into a higher block. If you forget to specify it correctly, the problem will be detected by the <ICL-X> semantic rule when running the extract_icl command at the level where the OutputPipeline node interacts with the BusFrequencyDivider it follows.

- parent_instance : *instance_name*;

  A property that specifies the parent instance in which the given node should be instantiated into. When unspecified, the node is instantiated directly into the current design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- leaf_instance_name : *leaf_instance_name*;

  A property that specifies the leaf instance name of the node. When unspecified, the leaf instance name defaults to the module name with the "_inst" suffix appended to it.

- bus_clock_period : *time*;

  This property is used to set the bus clock period to be used by the run_synthesis command when synthesizing the SSN nodes in isolation. When unspecified, it inherits its value from the DefaultChildConfiguration wrapper in its ancestry wrappers. When synthesizing the SSN nodes with the rest of the functional logic, the SDC procs created by the extract_sdc command, which is automatically called by the extract_icl command, is used. The ssn_bus_clock_period for the entire physical block is configured using the set_load_unload_timing_options command. Refer to "ScanHost Scan Interface Operation and Timing Requirements" on page 4498 in the ScanHost page for a description of how to keep a common definition of those settings for both SDC and pattern generation.

- Interface/*

  The properties in the Interface wrapper enable you to change the name of all the ports on the node. Refer to the Interface wrapper of the DefaultChildConfiguration wrapper to see the default value of each port name. If the property within the local Interface wrapper is not specified, the value is inherited from the Interface wrapper in the first DefaultChildConfiguration wrapper found in the ancestry of the current node.

- Connections/bus_clock_in : *port_pin_name*;

  A property that can be used to specify the source of the bus clock for the OutputPipeline node. This connection is typically inferred automatically. If unspecified and a DesignInstance wrapper is found below the OutputPipeline wrapper and this instance was created with an output bus clock port, the bus clock input of the OutputPipeline node will connect to it unless the ignore_bus_clock_out property is set to off in the DesignInstance/Connections wrapper. If a node below the OutputPipeline wrapper has its Connection/bus_clock_in property specified, it will also be used by the OutputPipeline node. Otherwise, it will connect to the object specified by the Datapath/Connections/bus_clock_in property, which defaults to a port name "ssn_bus_clock". You typically will never use this property as the default value is correct in all cases.

Examples

Example 1

The following example inserts an OutputPipeline node after a BusFrequencyDivider node to enable pipelining the data using an early branch of the clock to minimize the loop timing delay explained in the "Using the BusFrequencyDivider to Enable Faster Internal Bus Clock Frequency" on page 4423 sub-section of the BusFrequencyDivider wrapper description. The DftSpecification shown in this example builds the network illustrated in Figure 13-90 on page 4425. A normal Pipeline node could also be used as they are equivalent. You use the presence of an OutputPipeline node to instruct the CTS flow to use an early branch of the clock tree for the flip-flops within the node. To make this easier, the OutputPipeline node is equipped with a persistent clock buffer on the bus_clock_in port to help put the CTS constraints on it.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Once you have used an early clock branch to source the clock of the OutputPipeline node, there is little hold margin for any nodes following the OutputPipeline node. This is the reason to use an OutputPipeline node instead of a normal Pipeline. The fact that you instructed the CTS flow to use an early clock branch to it would not be captured in the ICL and the extract_icl command would not report an error if you added another node after it.

```
set spec [create_dft_specification -sri_sib_list ssn]
read_config_data -in $spec -from_string {
  SSN {
    Datapath(1) {
      output_bus_width : 64;
      OutputPipeline(1) { //operates at 1/2 rate
      }
      BusFrequencyDivider(1) {
        frequency_ratio : 2;
      }
      ScanHost(1) { //operates at full rate
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Receiver1xPipeline

A node used at the start of the Datapath, or when crossing SSN clock CTS regions and time-multiplexing is not used.

Usage

```
DftSpecification(module_name,id) {
  SSN {
    Datapath(id) {
      Receiver1xPipeline(id) {
        ijtag_host_interface     :  host_node_leaf_name ;
                                                  // default: Sib(ssn)
        ijtag_connection_order   :  order ;
        bus_clock_period         :  time ;        // default: 2.5ns
        parent_instance          :  instance_name ;
        leaf_instance_name       :  leaf_instance_name ;
        input_retiming_cell_type :  dff | latch ;  // default: Inherited
        Interface {
        }
        Connections {
          bus_clock_in : port_pin_name;
        }
        ExtraOutputPath {
        }
      }
    }
  }
}
```

Description

At the start of the network where the SSN data path receives the data from the tester, the SSN data in values change at time 0 of the tester period. When using a BusFrequencyMultiplier node at the start of the network, the update_phase of the node is “receiver,” which means that the input data bits are sampled in the middle of the tester cycle, allowing the propagation delay of the clock and data_in to differ by as much as 50% of the tester period in either direction. Use the time multiplexing node enable to have a tester period and an I/O data rate of 200 MHz at the boundary of the chip and a fast internal data rate within the chip as explained in “Using the BusFrequencyDivider to Enable Faster Internal Bus Clock Frequency” on page 4423. If you prefer to keep the internal bus rate in-line with the I/O data rate, do not use BusFrequencyMultiplier nodes at the start of the data path and BusFrequencyDivider nodes at the end of the data path. You will likely need them when crossing SSN clock CTS regions as explained in “Using the BusFrequencyDivider to Cross CTS Regions” on page 4421.

You use the Receiver1xPipeline node at the start of the data path when your internal SSN bus frequency is the same as the I/O rate. This node samples the data in bits on the negative edge of the SSN bus clock so as to strobe it in the middle of the tester period to again, as it was the case with the BusFrequencyMultiplier node described in the above paragraph, to tolerate skew in the propagation delay of the clock with respect to the input data bits. You will also use this node as the receiver node of a SSN bus clock CTS region crossing when your SSN bus data rate is slow enough that the skew between the two CTS regions is less than half a SSN bus clock period.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When it is larger, you use a BusFrequencyMultiplier node as explained in the "Using the BusFrequencyDivider to Cross CTS Regions" on page 4421 section.

Arguments

- ijtag_host_interface : *host_node_leaf_name*;

  A property that inherits its value from the ijtag_host_interface property in the Datapath wrapper. It has the exact same format. The default is Sib(ssn). You specify the property when you want to use a different IJTAG host interface for each node, but it is unlikely that you need to do that. To precisely control the ordering of the IJTAG network for the SSN nodes, specify the ijtag_connection_order property in each of the SSN hardware nodes.

- ijtag_connection_order : *int* | "";

  An optional property that controls the connection order for the IJTAG scan interface of this node relative to the others sharing a common ijtag_host_interface value. When left unspecified, it defaults to a null string, which means that the connection order for the IJTAG scan interface matches the relative order of the node with respect to the others in the SSN wrapper. If you specify an integer value for one node, then you must specify the integer value for all other nodes having the same ijtag_host_interface value, and the integer value must be unique for each node. The node with the smallest ijtag_connection_order value will be closest to tdo.

- bus_clock_period : *time*;

  This property is used to set the bus clock period to be used by the run_synthesis command when synthesizing the SSN nodes in isolation. When unspecified, it inherits its value from the DefaultChildConfiguration wrapper in its ancestry wrappers. When synthesizing the SSN nodes with the rest of the functional logic, the SDC procs created by the extract_sdc command, which is automatically called by the extract_icl command, is used. The ssn_bus_clock_period for the entire physical block is configured using the set_load_unload_timing_options command. Refer to "ScanHost Scan Interface Operation and Timing Requirements" on page 4498 in the ScanHost page for a description of how to keep a common definition of those settings for both SDC and pattern generation.

- parent_instance : *instance_name*;

  A property that specifies the parent instance in which the given node should be instantiated into. When unspecified, the node is instantiated directly into the current design.

- leaf_instance_name : *leaf_instance_name*;

  A property that specifies the leaf instance name of the node. When unspecified, the leaf instance name defaults to the module name with the "_inst" suffix appended to it.

- input_retiming_cell_type : dff | latch ;

  A property that specifies the default input retiming cell type. The default value is inherited from the DefaultChildConfiguration wrapper in the ancestry of the Receiver1XPipeline node.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Interface/*

  The properties in the Interface wrapper enable you to change the names of all the ports on the node. Refer to the Interface wrapper of the DefaultChildConfiguration wrapper to see the default value of each port name. If the property within the local Interface wrapper is not specified, the value is inherited from the wrapper in the first DefaultChildConfiguration wrapper found in the ancestry of the current node.

- Connections/bus_clock_in : *port_pin_name*;

  A property used to specify the source of the SSN bus clock in. Given this node is typically the first node of a Datapath, its source defaults to the bus_clock_in connection of the data path. You will need to specify this connection explicitly when the data path has more than one bus_clock_in connection.

Examples

This example uses a Receiver1xNode at the bottom of the Datapath wrapper because it is the start of the SSN data path on the chip, and the designer chose to operate the SSN bus inside the chip as the same rate as the I/Os of the chip.

```
set spec [create_dft_specification -sri_sib_list {ssn}]
read_config_data -in $spec -from_string {
  SSN {
    Datapath(1) {
      output_bus_width: 4;
      ScanHost(1) {
      }
      Receiver1xPipeline(in) {
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Pipeline

A wrapper used to specify the creation of a pipeline node.

Usage

```
DftSpecification(module_name,id) {
  SSN {
    Datapath(id) {
      Pipeline(id) {
        ijtag_host_interface    : host_node_leaf_name; //default: Sib(ssn)
        ijtag_connection_order  : int | "";
        bus_clock_period        : time;    // default: inherited
        frequency_ratio         : int  | auto;
        update_phase            : receiver | transmitter | auto;
        parent_instance         : instance_name;
        leaf_instance_name      : leaf_instance_name;
        Interface {
          bus_clock    : port_name; // default: inherited
          bus_data_in  : port_name; // default: inherited
          bus_data_out : port_name; // default: inherited
          IjtagScanInterface {
            reset      : port_name; // default: inherited
            tck        : port_name; // default: inherited
            select     : port_name; // default: inherited
            capture_en : port_name; // default: inherited
            shift_en   : port_name; // default: inherited
            update_en  : port_name; // default: inherited
            scan_in    : port_name; // default: inherited
            scan_out   : port_name; // default: inherited
          }
        }
        Connections {
          bus_clock_in : port_pin_name;
        }
        ExtraOutputPath {
        }
      }
    }
  }
}
```

Description

This wrapper is used to create and optionally insert an SSN node of type pipeline. This node is used to inject a pipeline stage along the data path to help with timing closure. For best results, pipeline the data path before it exits a physical block, assuming the SSN CTS region spans the two physical regions.

Note

If the SSN CTS region within the physical block is different than the SSN CTS region in the next block, you must use a BusFrequencyDivider node instead, as explained in "Using the BusFrequencyDivider to Cross CTS Regions" on page 4421.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The Pipelining node is typically used on the data path section operating at the full rate of the clock. If, however, you insert a Pipeline node before a BusFrequencyMultiplier node or after a BusFrequencyDivider node, the Pipeline node must hold *n*-1 out of *n* cycles to match the data rate relative to the clock. This is controlled by the frequency_ratio and update_phase properties. Their values both default to "auto," which means they are inferred correctly when the Datapath wrapper contains both a BusFrequencyDivider or a BusFrequencyMultiplier node with a Pipeline node inside the Datapath wrapper. The only time you need to specify those properties is if you are inserting the Pipeline node in a block by itself, and it will precede a BusFrequencyMultiplier or follow a BusFrequencyDivider node when assembled into a higher block. If you incorrectly specify the node, the problem will be detected by the ICL139 semantic rule when running the extract_icl command at the level where the Pipeline node interacts with the BusFrequencyDivider it follows, or the BusFrequencyMultiplier it precedes.

The phase counter registers within the Pipeline node are reset asynchronously by the ijtag_reset signal. The update_en pulse also resets the phase counters, enabling you to run several SSN pattern sets in series without resetting the IJTAG network. You can choose to make the ijtag_clock and ijtag_ue ports inactive, as long as all the SSN clocks in the active network are kept in lockstep from the time the ijtag_reset is released.

Arguments

- ijtag_host_interface : *host_node_leaf_name*;

  A property that inherits its value from the ijtag_host_interface property in the Datapath wrapper. The default is Sib(ssn). You specify it when you want to use a different IJTAG host interface for each node, but it is unlikely that you need to do that. To precisely control the ordering of the IJTAG network for the SSN nodes, specify the ijtag_connection_order property in each of the SSN hardware nodes.

- ijtag_connection_order : *int* | "";

  An optional property that controls the connection order for the IJTAG scan interface of this node relative to the others sharing a common ijtag_host_interface value. When left unspecified, it defaults to a null string, which means that the connection order for the IJTAG scan interface matches the relative order of the node with respect to the others in the DftSpecification. If you specify an integer value for one node, then you must specify this value for all other nodes having the same ijtag_host_interface value, and the integer value must be unique for each node. The node with the smallest ijtag_connection_order value will be closest to tdo.

  If you do not specify the property, the IJTAG chain ordering follows the SSN data path ordering. For example, in "Example 2" on page 4468 of the Multiplexer wrapper description, the property is not specified, and the IJTAG scan chain is stitched from smux1_1 through smux1_2, and smux2_1 to finish on smux2_2.

- frequency_ratio : ***int*** | auto;

  A property that is used to specify the frequency ratio between the data and the clock. When a Pipeline node precedes a BusFrequencyMultiplier node or when it follows a BusFrequencyDivider node, the Pipeline node must hold the data n-1 out of *n* cycles to

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

match the frequency_ratio value of the BusFrequencyMultiplier or BusFrequencyDivider nodes. When the Pipeline node is in a Datapath wrapper that also contains the BusFrequencyMultiplier or BusFrequencyDivider nodes, the auto value automatically infers the correct value. The only time you need to specify this property is if you are inserting the Pipeline node in a block by itself, and the block will precede a BusFrequencyMultiplier or follow a BusFrequencyDivider node when assembled into a higher block. If you incorrectly specify this, the problem is detected when running the extract_icl command at the level where the Pipeline node interacts with the BusFrequencyDivider it follows or the BusFrequencyMultiplier it precedes.

- update_phase : receiver| transmitter| auto;

  A property that is used to specify which cycle out of *n* the update is to happen in. When a Pipeline node precedes a BusFrequencyMultiplier node or when it follows a BusFrequencyDivider node, it must hold the data *n*-1 out of *n* cycles to match the frequency_ratio property value of the BusFrequencyMultiplier or BusFrequencyDivider nodes. The phase of the update must be "receiver" when it precedes a BusFrequencyMultiplier node, and it must be "transmitter" when it follows a BusFrequencyDivider node.

  When the Pipeline node is in a Datapath wrapper that also contains the BusFrequencyMultiplier or BusFrequencyDivider node, the auto value automatically infers the correct value. The only time you need to specify this property is if you are inserting the Pipeline node in a block by itself and it will precede a BusFrequencyMultiplier or follow a BusFrequencyDivider when assembled into a higher block. If you specify it incorrectly, the problem is detected when running the extract_icl command at the level.

- parent_instance : *instance_name*;

  A property that specifies the parent instance in which the given node should be instantiated into. When unspecified, the node is instantiated directly into the current design.

- leaf_instance_name : *leaf_instance_name*;

  A property that specifies the leaf instance name of the node. When unspecified, the leaf instance name defaults to the module name with the "_inst" suffix appended to it.

- bus_clock_period : *time*;

  This property is used to set the bus clock period to be used by the run_synthesis command when synthesizing the SSN nodes in isolation. When unspecified, it inherits its value from the DefaultChildConfiguration wrapper in its ancestry wrappers. When synthesizing the SSN nodes with the rest of the functional logic, the SDC procs created by the extract_sdc command, which is automatically called by the extract_icl command, is used. The ssn_bus_clock_period for the entire physical block is configured using the set_load_unload_timing_options command. Refer to "ScanHost Scan Interface Operation and Timing Requirements" on page 4498 in the ScanHost page for a description of how to keep a common definition of those settings for both SDC and pattern generation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Interface/*

  The properties in the Interface wrapper enable you to change the name of all the ports on the node. Refer to the Interface wrapper of the DefaultChildConfiguration wrapper description to see the default value of each port name. If the property within the local Interface wrapper is not specified, the value is inherited from the Interface wrapper in the first DefaultChildConfiguration wrapper found in the ancestry of the current node.

- Connections/bus_clock_in : *port_pin_name*;

  A property that can be used to specify the source of the bus clock for the Pipeline node. This connection is typically inferred automatically. If unspecified and a DesignInstance wrapper is found below the Pipeline wrapper and this instance was created with an output bus clock port, the bus clock input of the Pipeline node will connect to it unless the ignore_bus_clock_out_port property is set to off in the DesignInstance/Connections wrapper. If a node below the Pipeline wrapper has its Connection/bus_clock_in property specified, it will also be used by the Pipeline node. Otherwise, it will connect to the object specified by the Datapath/Connections/bus_clock_in property, which defaults to a port name “ssn_bus_clock”. You typically will never use this property as the default value is correct in all cases.

  When the design level is chip, the port must already exist and be connected to a pad buffer cell. You may also have a boundary scan cell equipped with AuxiliaryInputData logic. When you point to such a top-level port, the tool automatically maps the connection across the pad and optional boundary scan cell.

Examples

Example 1

Refer to “Example 1” on page 4395 in the SSN section for a typical use of the Pipeline node where it is used to cut the long path to and from the upper block.

Example 2

This example shows a Pipeline(2) node inserted below (meaning in front of) a BusFrequencyMultiplier node. The frequency_ratio and update_phase properties are not specified in the Pipeline wrapper so they default to “auto”. Because the Pipeline node precedes a BusFrequencyMultiplier node with a frequency_ratio of 2, the frequency_ratio value is inferred as 2 and the update_phase value is inferred as “receiver”. The Pipeline(1) is above a BusFrequencyMultiplier node, and it operates at full rate.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set spec [create_dft_specification -sri_sib_list ssn]

read_config_data -in $spec -from_string {
  SSN {
    Datapath(1) {
      output_bus_width : 16;
      Pipeline(1) { //operates at full rate
      }
      BusFrequencyMultiplier(1) {
        frequency_ratio : 2;
      }
      Pipeline(2) { //operates at 1/2 rate
      }
    }
  }
}
```

Example 3

This example shows a Pipeline(1) node inserted above (meaning after) a BusFrequencyDivider node. The frequency_ratio and update_phase properties are not specified in the Pipeline wrapper so they default to “auto”. Because the Pipeline node follows a BusFrequencyDivider node with a frequency_ratio of 4, the frequency_ratio value is inferred as 4 and the update_phase value is inferred as “transmitter”. The Pipeline(2) is below the BusFrequencyDivider node, and it operates at full rate.

```
set spec [create_dft_specification -sri_sib_list ssn]
read_config_data -in $spec -from_string {
  SSN {
    Datapath(1) {
      output_bus_width : 16;
      Pipeline(1) { //operates at 1/4 rate
      }
      BusFrequencyDivider(1) {
        frequency_ratio : 4;
      }
      Pipeline(2) { //operates at full rate
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ScanHost

A wrapper used to specify the creation of a ScanHost node and its optional insertion into a SSN datapath.

Usage

```
DftSpecification(module_name,id) {
  SSN {
    Datapath(id) {
      ScanHost(id) {
        ijtag_host_interface          : host_node_leaf_name;
                                                  // default: Sib(ssn)
        ijtag_connection_order        : int | "";

        bus_clock_period              : time; // default: Inherited
        max_capture_clock_pulses      : int;  // default: Inherited
        max_capture_to_shift_clock_period_ratio : int;
                                                // default: Inherited
        parent_instance               : instance_name;
        leaf_instance_name            : leaf_instance_name;
        max_scan_chain_length         : int | auto; // default: Inherited
        max_scan_en_mcp               : 2 | 4 | 8;  // default: Inherited
        max_edt_update_mcp            : 2 | 4 | 8;  // default: Inherited

        input_chain_count             : int | from_edt_controller;
        output_chain_count            : int | from_edt_controller |
                                        same_as_input_chain_count;
        output_chain_count_in_on_chip_compare_mode : int |
                     from_edt_controller |
                     from_edt_controller_high_compression_configuration;
        high_compression_input_channel_count   : int | auto;
        high_compression_output_channel_count  : int | auto;
        use_high_compression_channel_count_in_bypass_mode : on | off;
        support_from_scan_out_le_strobing      : on | off;
        scan_signals_bypass                    : on | off |
                                                 controls_only | auto;
        use_clock_dff_cell                     : on | off;
        use_clock_or_cell                      : on | off;
        use_clock_shaper_cell                  : on | off | auto;
        support_output_clock_activation_when_ssh_is_off : on | off;
        use_ssn_bus_clock_as_test_clock_bypass : on | off;
        dft_signals_not_mapped_on_ssh_outputs  :
                                               dynamic_dft_signal_name, …;

        OnChipCompareMode {
          present                  : on | off | auto;
          sticky_status_resolution : ssh | output_chain;
          status_groups            : int; // default: 128
        }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
ChainGroup(id) {
  input_chain_count                    : int | from_edt_controller;
  output_chain_count                   : int | from_edt_controller |
                                          same_as_input_chain_count;
  output_chain_count_in_on_chip_compare_mode : int |
            from_edt_controller  |
            from_edt_controller_high_compression_configuration;
  high_compression_input_channel_count       : int | auto;
  high_compression_output_channel_count      : int | auto;
  use_high_compression_channel_count_in_bypass_mode : on | off;
  support_from_scan_out_le_strobing          : on | off;
  scan_signals_bypass                        : on | off;
}

Interface {
  bus_clock             : port_name; // default: Inherited
  bus_data_in           : port_name; // default: Inherited
  bus_data_out          : port_name; // default: Inherited
  ssh_is_active         : port_name; // default: Inherited
  ssh_is_active_present : on | off | auto ;
  ChainGroup {
    scan_en                    : port_name;  // default: Inherited
    scan_en_bypass_in          : port_name;  // default: Inherited
    edt_update                 : port_name;  // default: Inherited
    edt_update_bypass_in       : port_name;  // default: Inherited
    test_clock_present         : on | off | auto;
    test_clock                 : port_name;  // default: Inherited
    test_clock_bypass_in       : port_name;  // default: Inherited
    edt_clock_present          : on | off | auto;
    edt_clock                  : port_name;  // default: Inherited
    edt_clock_bypass_in        : port_name;   / default: Inherited
    shift_capture_clock_present : on | off | auto;
    shift_capture_clock        : port_name;  // default: Inherited
    shift_capture_clock_bypass_in : port_name;
                                             // default: Inherited
    shift_clock_present        : on | off | auto;
    shift_clock                : port_name;
                                          // default: shift_clock
    shift_clock_bypass_in      : port_name;
                                 // default: shift_clock_bypass_in
    capture_clock_present   : on | off | auto;
    capture_clock              : port_name;
                                        // default: capture_clock
    capture_clock_bypass_in    : port_name;
                               // default: capture_clock_bypass_in
    to_scan_in                 : port_name;  // default: Inherited
    to_scan_in_bypass_in       : port_name;  // default: Inherited
    from_scan_out              : port_name;  // default: Inherited
    from_scan_out_bypass_out   : port_name;  // default: Inherited
  }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
            ScanHostClockSignalModule {
              define_clock_on_boundary      : on | off | auto;
              module_name                   : module_name;
              write_out_module_definition   : on | in_separate_file |
                                              off | auto;
              // The default value for all following properties is the
              // property name itself. (For example, the default value of
              // the "clock" property is "clock".) The values are inherited
              // from the DefaultChildConfiguration wrapper.
              clock                         : port_name;
              enable                        : port_name;
              enable_sync                   : port_name;
              ijtag_clock_cg_en             : port_name;
              ijtag_clock                   : port_name;
              next_edt_clock_div            : port_name;
              edt_clock_cg_en               : port_name;
              edt_clock_bypass_in           : port_name;
              edt_clock                     : port_name;
              next_shift_capture_clock_div  : port_name;
              shift_capture_clock_cg_en     : port_name;
              shift_capture_clock_bypass_in : port_name;
              shift_capture_clock           : port_name;
              next_shift_clock_div          : port_name;
              shift_clock_cg_en             : port_name;
              shift_clock_bypass_in         : port_name;
              shift_clock                   : port_name;
              next_capture_clock_div        : port_name;
              capture_clock_cg_en           : port_name;
              capture_clock_bypass_in       : port_name;
              capture_clock                 : port_name;
              next_test_clock_div           : port_name;
              test_clock_cg_en              : port_name;
              test_clock_bypass_in          : port_name;
              test_clock                    : port_name;
            }

            IjtagScanInterface {
              reset               : port_name; // default: Inherited
              tck                 : port_name; // default: Inherited
              select              : port_name; // default: Inherited
              capture_en          : port_name; // default: Inherited
              shift_en            : port_name; // default: Inherited
              update_en           : port_name; // default: Inherited
              scan_in             : port_name; // default: Inherited
              scan_out            : port_name; // default: Inherited
            }
          }
          Connections {
          }
          ExtraOutputPath {
          }
        }
      }
    }
  }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

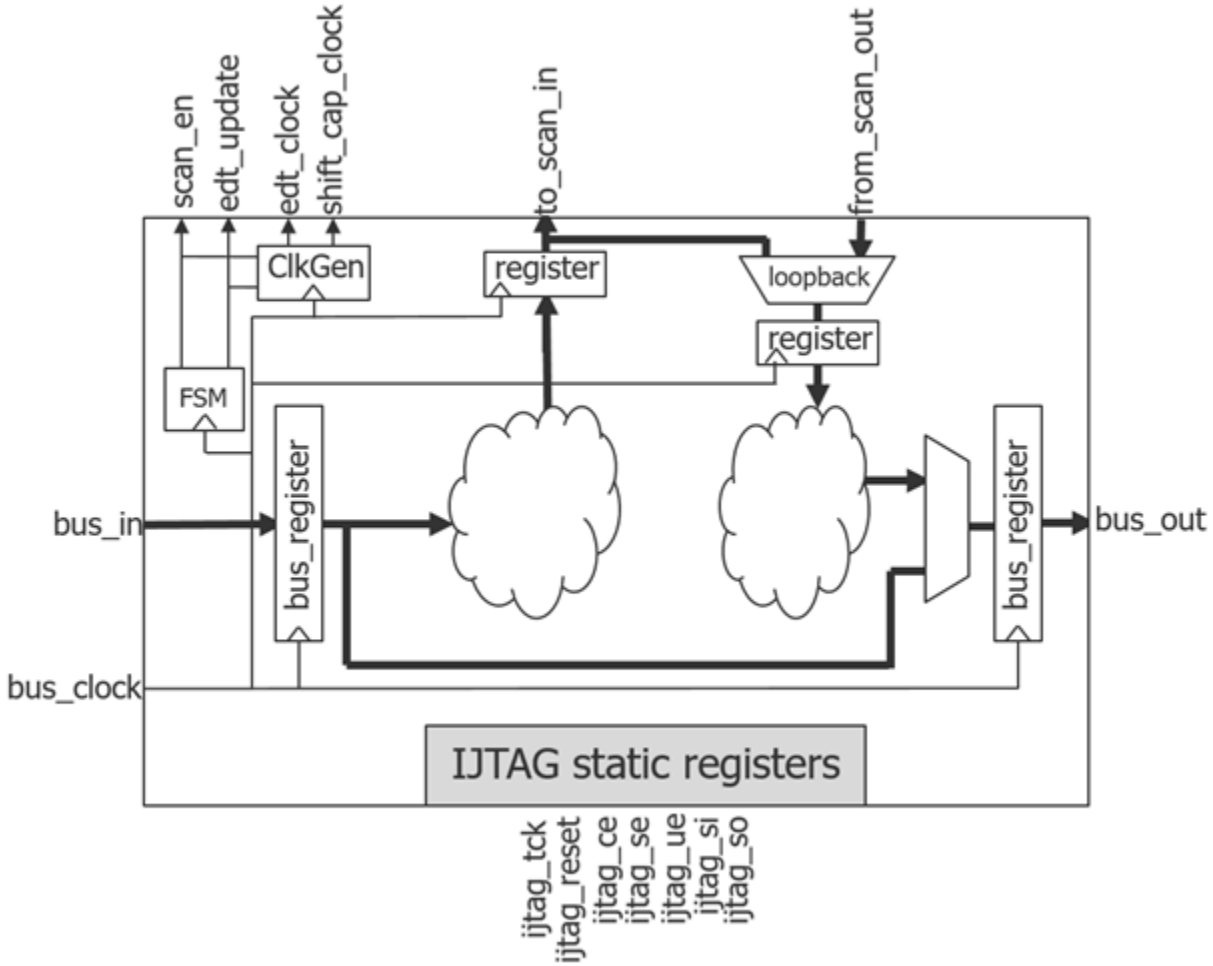
Description

This wrapper creates a ScanHost SSN node. This node type transfers data between the SSN datapath and the logic test circuitry, specifically EDT controllers and/or uncompressed scan chains. Several scan modes can be multiplexed onto a single ScanHost node. See "Example 1" on page 4523 for an example of a ScanHost node created with two chain groups where one group drives an uncompressed scan chain collecting all OCCs, and the other chain group drives the EDT controllers used for internal and external scan modes as well as for the bypass multi-chain mode.

The block diagram of the ScanHost node (often labeled "SSH") is shown in Figure 13-105. The data flows from bus_in to bus_out with two internal pipelining stages. When the data destined for the current node is passing between the two pipelining stages, the data is collected and presented on the to_scan_in output bus. The data received from the from_scan_out input bus is re-timed and used to replace the same time slots in the data stream when not using the on-chip compare mode. See Figure 13-108 on page 4491 for an illustration of the packets when not using on-chip compare. See Figure 13-111 on page 4493 for an illustration of the packets when using the on-chip compare mode, a mode especially useful for identical cores as it allows reusing the same data for an infinite number of repeated core instances.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-105. ScanHost Block Diagram



The ScanHost node is programed using the IJTAG network to enable it so that it extracts and replaces the data destined to and from it. Table 13-14 on page 4494 lists the IJTAG registers present within the ScanHost node and explains their usage. These registers are automatically programmed by the tool to match the requirements of the intended usage whether it is for ATPG, scan pattern retargeting, or with the StreamingScanNetworkVerify patterns. The SSN nodes are described in ICL, and their interconnectivity is design rule checked and extracted using the extract_icl command. Using the information contained in the ICL file, the tool knows exactly how to program each node such that the correct data is sent to and received from the intended places.

The ScanHost node generates the four controllers and clocks needed to operate an EDT controller and any generic logic test scan chain. When a complete scan word is present on the to_scan_in bus, the shift_capture_clock and the edt_clock are pulsed. The from_scan_out bus is sampled on the falling edge of the shift_capture_clock, but it can be programed to be sampled on the rising edge for when the scan chains do not have a trailing edge element at the end (see the support_from_scan_out_le_strobing property description). The to_scan_in data is updated relative to the falling edge of the shift_capture_clock. The effective shift frequency of the logic test scan chains can be much slower than the SSN bus frequency when the packet size is large.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Refer to "ScanHost Scan Interface Operation and Timing Requirements" on page 4498 to know how the timing works, and how it is described to synthesis and layout tools.

Note

See also "Bus Width Scaling Limitations for Several Active SSHs With Different Frequencies in One Datapath" on page 4413 in the Datapath description.

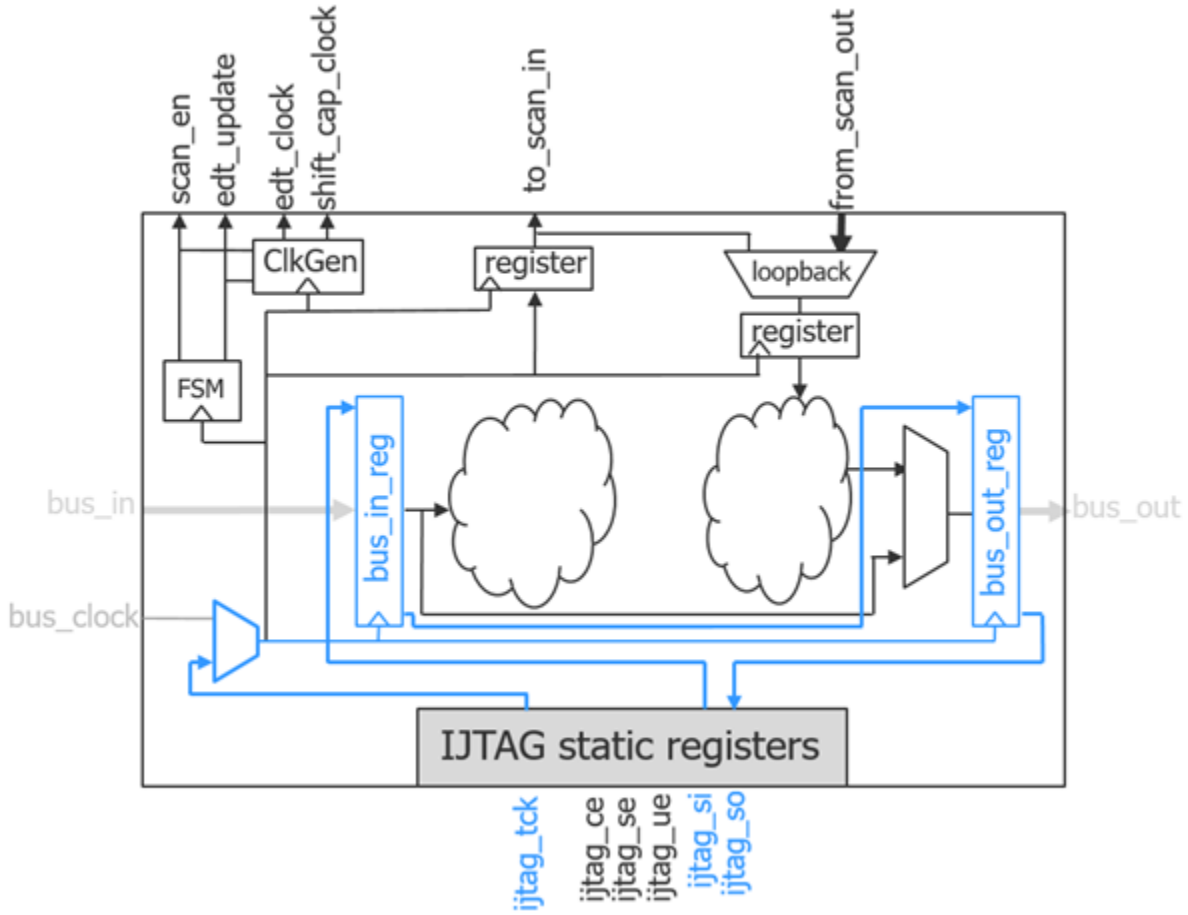
Streaming-Through-IJTAG

Refer to the section "Streaming-Through-IJTAG Scan Data" in the *Tessent Shell User's Manual* for information on using the Streaming-Through-IJTAG feature.

Debug Using IJTAG

You can configure the ScanHost to read the internal bus registers through IJTAG. Figure 13-106 shows the access path to read the bus_in_reg and bus_out_reg registers via IJTAG. The create_ssn_continuity_patterns command uses this path through the ScanHost node.

Figure 13-106. ScanHost Bus Register Access Via IJTAG



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SSN Packet Formats

Different ScanHost nodes can be programmed differently while sharing a common SSN datapath. Some may be programmed in the normal mode where the time slots used to carry in the scan-in values are used to carry out the scan-out values. Other nodes can be programmed in the on-chip capture mode when you have several identical cores or when you want a pass/fail bit to automatically identify failing cores.

Figure 13-107 shows two ScanHost nodes in series. In reality, the nodes could be separated by several inactive nodes, and this separation would not affect the packet format. The node in Block A is programed with a scan word size of five bits to service a 5-channel EDT controller. The node in Block B is programmed with a scan word of four bits to service a 4-channel EDT controller. The resulting packet shown in Figure 13-108 on page 4491 is with no throttling. You use throttling to minimize test time when running multiple ScanHost nodes in parallel that have scan clients with mismatched chain length or pattern count.

Continuing with Figure 13-108 on page 4491, the bus size is 8-bits wide, and the packet size is 4+5=9 bits. The packets are packed into the bus with no gaps. The state machines in the ScanHost nodes keeps track of the position of the packet over time, including which bits belong to it and which bits are to be passed though unaltered. The labels in Figure 13-108 use the following format:

```
<node_id>I<chain_time_slot><chain_id>
```

where:

- *node_id* — Refers to the node. In this example, this is either “A” or “B” to identify the time slots associated with Block A and Block B, respectively.
- I — Identifies the nodes as ScanIn time slots.
- *chain_time_slot* — Identifies the bits of the scan chain with time slot 0 being closest to the output.
- *chain_id* — In this example, this is the 5 and 4 chains for each EDT controllers.

The scan out data returned by the EDT controllers replaces the I time slots with an offset of 2. For example, the first scan words out of BlockA AO0x replaces the AI2x values, the AO1x values replaces the AI3x values and so on.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-107. Two SSH Nodes Used in Standard Mode

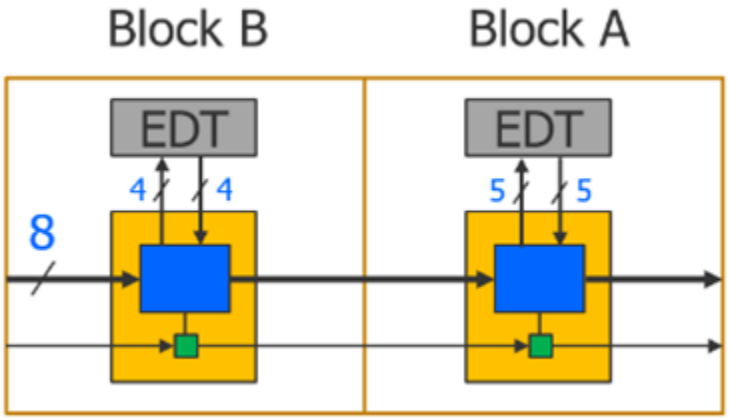


Figure 13-108. SSN Packet Formats When Not Using On-Chip Compare

BI91	BI82	BI73	AI70	AI61	AI52	AI43	AI34	BI20	BI11	BI02
BI90	BI81	BI72	BI63	AI60	AI51	AI42	AI33	AI24	BI10	BI01
AI94	BI80	BI71	BI62	BI53	AI50	AI41	AI32	AI23	AI14	BI00
AI93	AI84	BI70	BI61	BI52	BI43	AI40	AI31	AI22	AI13	AI04
AI92	AI83	AI74	BI60	BI51	BI42	BI33	AI30	AI21	AI12	AI03
AI91	AI82	AI73	AI64	BI50	BI41	BI32	BI23	AI20	AI11	AI02
AI90	AI81	AI72	AI63	AI54	BI40	BI31	BI22	BI13	AI10	AI01
BI83	AI80	AI71	AI62	AI53	AI44	BI30	BI21	BI12	BI03	AI00

Figure 13-109 on page 4492 shows six identical cores daisy-chained into a SSN datapath. They are programmed in the on-chip compare mode where the expected and mask values are transmitted after the scan-in values.

The circuit shown in Figure 13-110 on page 4492 is used to perform the on-chip compare and cumulate the result of the comparison with the results from the previous nodes. A sticky bit observable at the end of the pattern set with IJTAG also holds the pass/fail result of the specific node. For more information about the on-chip compare mode, and how you can trade off test time versus diagnosis ability, see "On-Chip Compare With SSN" in the *Tessent Shell User's Manual*.

Figure 13-111 on page 4493 shows the packet format when using several ScanHost nodes in on-chip compare mode and two diagnostic status groups are used. A packet can contain bits for ScanHost nodes programed in both normal and on-chip compare modes simultaneously but Figure 13-108 and Figure 13-111 on page 4493 show them separately to simplify the picture. The label of the time slot follows the same convention as those described Figure 13-108 except

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

there are more labels than “I”. The label “I” still represents the scan-in values, the “E” labels represent the expected values, and the “M” labels represent the mask values. The “a” and “b” labels represent the status time slots. In this example, there are two status groups labeled “a” and “b”. As shown in Figure 13-109, the A1, A2, and A3 instances are programmed to cumulate their comparison status in the “a” status time slots. The A4, A5, and A6 instances are programmed to cumulate their comparison status in the “b” status time slots. Just like with the standard packet, there is an offset of two packets between the input time slots (I, E, and M) and the output time slots (a and b).

Figure 13-109. Six SSH Nodes Used in On-Chip Compare Mode

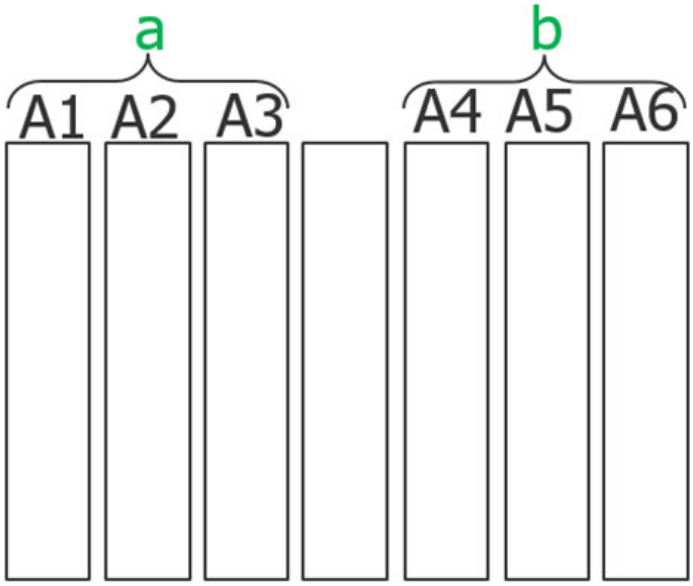
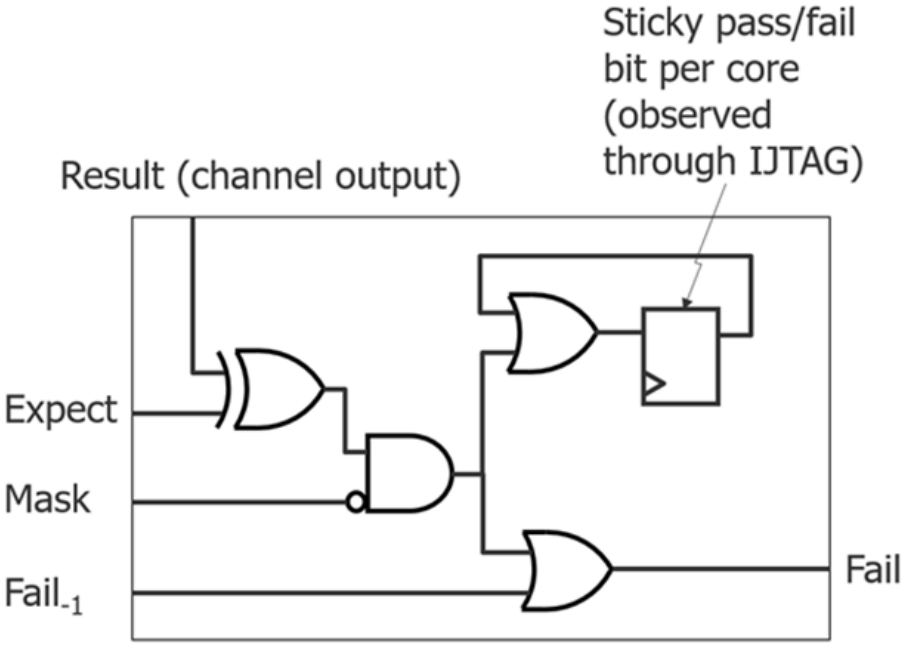


Figure 13-110. SSH On-Chip Compare Logic



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-111. SSN Packet Formats When Using On-Chip Compare

AI65	Aa31	AI54	Aa20	AI43	AM31	AI32	AM20	AI21	AE11	AI10	AE00
AI64	Aa30	AI53	AM41	AI42	AM30	AI31	AE21	AI20	AE10	Ab-1	AI06
AI63	AM51	AI52	AM40	AI41	AE31	AI30	AE20	Ab-1	AI16	Ab-0	AI05
AI62	AM50	AI51	AE41	AI40	AE30	Ab01	AI26	Ab-0	AI15	Aa-1	AI04
AI61	AE51	AI50	AE40	Ab11	AI36	Ab00	AI25	Aa-1	AI14	Aa-0	AI03
AI60	AE50	Ab21	AI46	Ab10	AI35	Aa01	AI24	Aa-0	AI13	AM01	AI02
Ab31	AI56	Ab20	AI45	Aa11	AI34	Aa00	AI23	AM11	AI12	AM00	AI01
Ab30	AI55	Aa21	AI44	Aa10	AI33	AM21	AI22	AM10	AI11	AE01	AI00

Programmable IJTAG Registers Inside the ScanHost Node

Table 13-14 lists the IJTAG registers present in the ScanHost node.

The registers within the ScanHost are reset synchronously on the first SSN bus_clock pulse of a session. The internal ScanHost enable_sync signal is asynchronously reset to 0 when the ijtag_reset port is asserted and during every ijtag_ue pulse. Resetting on the update_en pulse enables you to run several SSN pattern sets in series without having to reset the IJTAG network.

The three register names starting with “on_chip_compare...” are only present if the node was built with on-chip compare using the OnChipCompareMode/present property.

The to_scan_in_bits and from_scan_out_bits registers may have several copies, uniquified by a suffix if the ChainGroup wrapper was used in the DftSpecification. The tool automatically sets those registers up based on your setup.

Note

The following table of IJTAG registers in the SSH is provided as information only. Refer to the list of IJTAG registers in the SSH that you can reprogram during create_patterns and scan pattern retargeting in the reference for the command “set_core_instance_parameters” on page 2402. All other SSH IJTAG registers are automatically programmed by the tool.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-14. IJTAG Registers in ScanHost Nodes

IJTAG register name	Description
bits_per_packet	This register holds the number of bits per packet the node must expect. This value is between 1 and the sum of all to_scan_in_bits_$chain_group_id values. When the value is smaller than the maximum, the ScanHost node is said to be in throttling mode. For more information about throttling, refer to the "Yield Statistics on ATE With SSN" section in the "Streaming Scan Network" chapter of the *Tessent Shell User's Manual*.
bus_width	This register holds the currently used bus width. The bus_width cannot be larger than the packet_size value so it specifies a value smaller than the actual bus size when running only a few nodes in parallel, which causes the packet to be small.
capture_packets	This register holds the number of capture packets that triggers a pulse on the shift_capture_clock. It gets multiplied by (packets_per_capture_clock_minus_one+1) to determine the total number of capture packets to expect.
config_sib	This register provides access to the bulk of the configuration bits listed in this table. It resets to 0. When it is 0, the IJTAG chain within the ScanHost node is two bits long, including the config_sib itself and the streaming_through_ijtag bit. When it is 1, all bits are included in a long shift register. The config_sib shift register is used to capture the sticky_status result when the ScanHost supports on-chip compare mode.
disable_first_scan_load_masking	This register controls whether the ScanHost node masks the first one or two (when init_scan_load_used is also set) scan loads, which makes retention testing possible. When this register is set to 1, the ScanHost node does not perform this masking.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-14. IJTAG Registers in ScanHost Nodes (cont.)

IJTAG register name	Description

disable_on_chip_compare_contribution	This register is used to suppress the contribution of all but one node contributing to a common status group (node with on_chip_compare_group register having a common value) to obtain the precise diagnostic data for the one node during on-chip compare diagnostic runs. If this register is set, the status sticky bits for the SSH are not set.
	
	To create a testbench with this bit set to 1 in a ScanHost instance, set the Tcl variable ssh_disable_contribution_icl_instance_name_list in the global name space to define a list of ICL instance names that point to the ScanHost ICL instance you want to disable, as follows:
	
	`set ssh_disable_contribution_icl_instance_name_list {corea_i1.corea_rtl2_tessent_ssn_scan_host_1_inst}`
	
	A special annotation in the STIL file identifies this IJTAG register. The annotation directly precedes the vector where the register is set, enabling simple patching on the ATE.
	
	Subsequent write_patterns commands display this message for each ScanHost that is disabled:
	
	`Setting disable_on_chip_compare_contribution to 1 for ICL Instance 'corea_i1.corea_rtl2_tessent_ssn_scan_host_1_inst'.`
	
	The following example demonstrates the annotation, where the hierarchical prefix path is the ICL instance name of the associated SSH.
	
	`Ann {* TESSENT_PRAGMA bit_variable core_g1_tessent_ssn_scan_host_piccpu_inst.disable_on_chip_compare_contribution -type write -var_bits {0} -pin ijtag_si -inversion 0b0 *}`
	`V {`
	`_pi_=00010101000000;`
	`_po_=XXXXXX;`
	`}`
	
	For more information on the presence of iWriteVars in the STIL file and how they can be patched, refer to the "PDL Retargeting With Symbolic Variables" section of the *Tessent IJTAG User's Manual*.

edt_update_falling_launch_word	This register holds the count at which the edt_update falls back after the edt_update packet to control its timing. It is a value between edt_update_falling_transition_words and 0, inclusively.
edt_update_falling_transition_words	This register holds the total number of scan word packets needed to perform the edt_update cycle and transition the edt_update signal to zero before the first shift_clock pulse is issued.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-14. IJTAG Registers in ScanHost Nodes (cont.)

IJTAG register name	Description
enable	This register is loaded with a 1 when the ScanHost node is enabled. ScanHost host nodes that are not enabled forward the data unaltered with two pipeline delay stages.
extra_shift_packets	This register holds the number of extra shift packets the given nodes need to skip before going into the capture sequence. It is typically left at 0 unless the capture needs to be aligned between several ScanHost nodes belonging to a common ATPG group.
from_scan_out[_$chain_group_id]_le_strobe_en	This register only exists if the support_from_scan_out_le_strobing property is set to on for the given chain group. The tool detects that the chains or channel outputs associated to the chain groups are all LE scan elements and sets this register to 1 to activate the LE strobing into the controller.
from_scan_out_bits[_$chain_group_id]	This register holds the number of channel outputs or chain outputs used in each chain group. The number of output EDT channels cannot be scaled down for a given EDT controller, but because the same ScanHost node can be reused by several scan modes, you may have different chain outputs in each mode. For example, 16 in multi_bypass_mode, 8 in internal EDT mode and 2 in external EDT mode.
initial_bit0_position	This register holds the position of the first bit of the first packet belonging to the current ScanHost node.
initial_offset	This register holds the number of bus clock cycles it takes for the packets to travel from the input of the datapath to the current ScanHost node. The register is sometimes set to a larger value than the physical distance of the node within the datapath when capture aligned patterns are created and padding packets are injected at the beginning to align the capture of a group of ScanHost nodes.
init_scan_load_used	This register is set to 1 when the ScanHost node drives an EDT controller that requires an initialization scan load, such as when there are pipelining flops on the input channels, or the EDT has the shift power control circuitry. The register tells the hardware to mask the response of the first one or two scan loads to avoid X reconvergence issues during simulation.
loop_back_en	This register activates the loopback mode of the ScanHost node. The shift_capture_clock output port is gated off when loop_back_en is on so as not to burn excessive power while the random data is begin used. See the loopback multiplexer in Figure 13-105 on page 4488.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-14. IJTAG Registers in ScanHost Nodes (cont.)

IJTAG register name	Description
min_capture_clock_low_width	This register holds the minimum number of bus clock cycles available to fit half a capture clock cycle. This enables maintaining a 50 percent duty cycle for the generated capture clock.
min_shift_clock_low_width	This register holds the minimum number of bus clock cycles available to fit half a shift clock cycle. This enables maintaining a 50 percent duty cycle for the generated shift clock.
on_chip_compare_enable	This register is 1 when the on-chip compare mode is active.
on_chip_compare_group	This register holds the compare group the current ScanHost node is to cumulate its status into. When programmed with a value of 0, the node does not use any status time slots and only cumulates its comparison status into the sticky register. In the examples shown in Figure 13-109 on page 4492 and Figure 13-111 on page 4493, this value is programmed to 1 in instance I1, I2, I3 and this value is programmed to 2 in instance I4, I5, I6.
on_chip_compare_group_count	This register holds the number of status groups present in the packet for the on-chip compare group associated with the current ScanHost. In the examples shown in Figure 13-109 on page 4492 and Figure 13-111 on page 4493, this value is programmed to 2.
packet_size	This register holds the total number of bits for the packets going through the current ScanHost node.
packet_width_is_less_than_two_bus_clock	This register is set to 1 when the packet_size/bus_width generates a number that is less than 2. The value of this register is used to decide if the shift_capture_clock and edt_clock are created with a clock divider, or with a clock gater, as shown in the ScanHost Scan Interface Operation and Timing Requirements section.
packets_per_capture_clock_minus_one	This register holds the number of packets between each capture clock pulse so as not to exceed the minimum capture clock period needed to perform staggered low-speed captures.
scan_en_launch_packet	This register holds the count at which point the scan_en and edt_update signals transition. The value is between scan_en_transition_packets and 1, inclusively.
scan_en_transition_packets	This register holds the number of packets needed to transition scan_en high and low. The value also applies to the rising transition of edt_update.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-14. IJTAG Registers in ScanHost Nodes (cont.)

IJTAG register name	Description
status_sticky_per_output_chain_sib	This register reduces the number of sticky bits to be shifted out on the TDO pin during the ssn_end procedure, when the ScanHost supports on-chip compare and the sticky_status_resolution value is set to “output_chain” This Sib register is controlled by a register under the config_sib. When this register is 0, it is equivalent to the case where sticky_status_resolution=ssh (the default). When this register is 0 and config_sib is 1, the IJTAG chain within the ScanHost node is trimmed to have no more than four configuration bits, with the remainder being on-chip compare sticky status. When nonzero, this shift register captures the sticky status per output chain results. This register and its corresponding SIB are generated only when the following requirements are met: • On-chip compare is used with the setting “sticky_status_resolution: output_chain”. • The number of on-chip compare sticky bit status bits to shift out is less than the total number of bits in the SSH IJTAG shift register.
to_scan_in_bits[_$chain_group_id]	This register holds the number of channel inputs or chain inputs used in each chain group. The number of input EDT channels can be scaled down at runtime if the EDT controller was built with the separate_control_data_channels property set to on. An uncompressed scan mode cannot scale its input chain count. This register is needed even if you do not have compressed scan chains because the same ScanHost node can be reused by several scan modes, and you may have different chain inputs in each mode. For example, 16 in multi_bypass_mode, 14 in internal EDT mode and 2 in external EDT mode.
total_shift_cnt_minus_one	This register holds the total number of shift cycles minus one. This include the sum of the shift cycles needed by the scan client of the ScanHost node plus any extra shift packets programmed with the extra_shift_packet register.

ScanHost Scan Interface Operation and Timing Requirements

The following introduction describes how the ScanHost node generates the scan clocks and how the SDC describes them. The ScanHost node generates the scan control and data signals when it drives the logictest modes of a block. The signals are all generated from the SSN bus_clock as shown in Figure 13-112 on page 4500 and Figure 13-113 on page 4501.

The test patterns apply the SSN bus clock using up to three different frequencies, depending on the mode of operation. The first one is called ssn_bus_clock_network and indicates the SSN

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

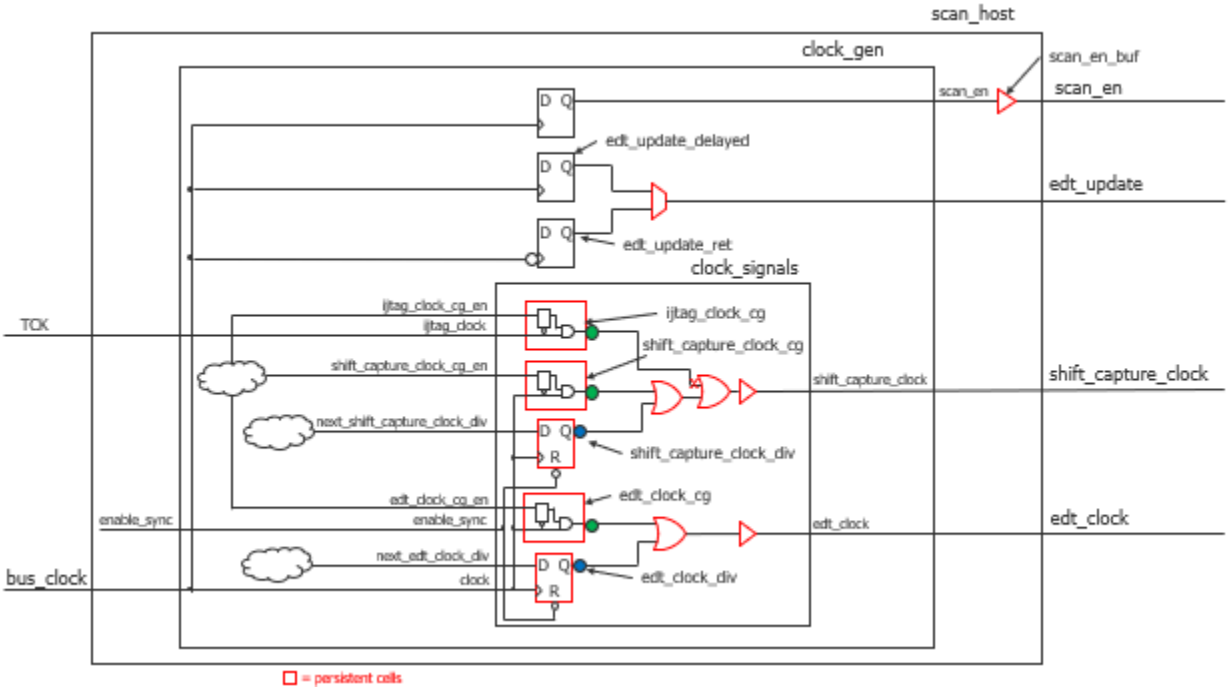
maximum target bus frequency, regardless of whether the ScanHost node is active. An active SSH might sometimes require a slower SSN bus clock than this maximum. If so, the maximum speed could still be required when transferring test data through the current block for other parts of the chip. This clock period default is 2.5 ns and is configured using the "set_load_unload_timing_options -ssn_bus_clock_period" command. The set_load_unload_timing_options command is used to configure the SDC parameters and also to configure the pattern generation such that the parameters of the generated patterns are consistent with those used during timing closure and analysis. For detailed information about Tessent SSN-related SDC constraints for the ScanHost scan interface, refer to the section "SSN SDC Constraints in the Design Flow" in the *Tessent Shell User's Manual*.

When the ScanHost node is active, the SSN bus clock is used to generate the shift_capture_clock and the edt_clock used to operate the scan circuitry.

- When the SSN packet is small enough to occupy fewer than two bus clock cycles on the network, the scan clocks must be generated from the SSN bus_clock using a clock gater, labeled **shift_capture_clock_cg** and **edt_clock_cg** in Figure 13-112. The three blue dots show persistent cell output pins that serve as anchor points on which to declare the gated clocks in the Tessent SDC file. The test patterns also adjust the SSN bus clock frequency to comply with the shift_clock period specified by the set_load_unload_timing_options command.
- When the SSN packet is large enough to occupy a minimum of *N* bus clock cycles on the network, the scan clocks are generated using a clock divider to maintain, as much as possible, a fifty percent duty cycle. The two divided clocks come from the two flop outputs. These flops are labeled shift_capture_clock_div and edt_clock_div in the following figure. The two green dots show persistent cell output pins that serve as anchor points on which to declare the divided clocks in the Tessent SDC file.

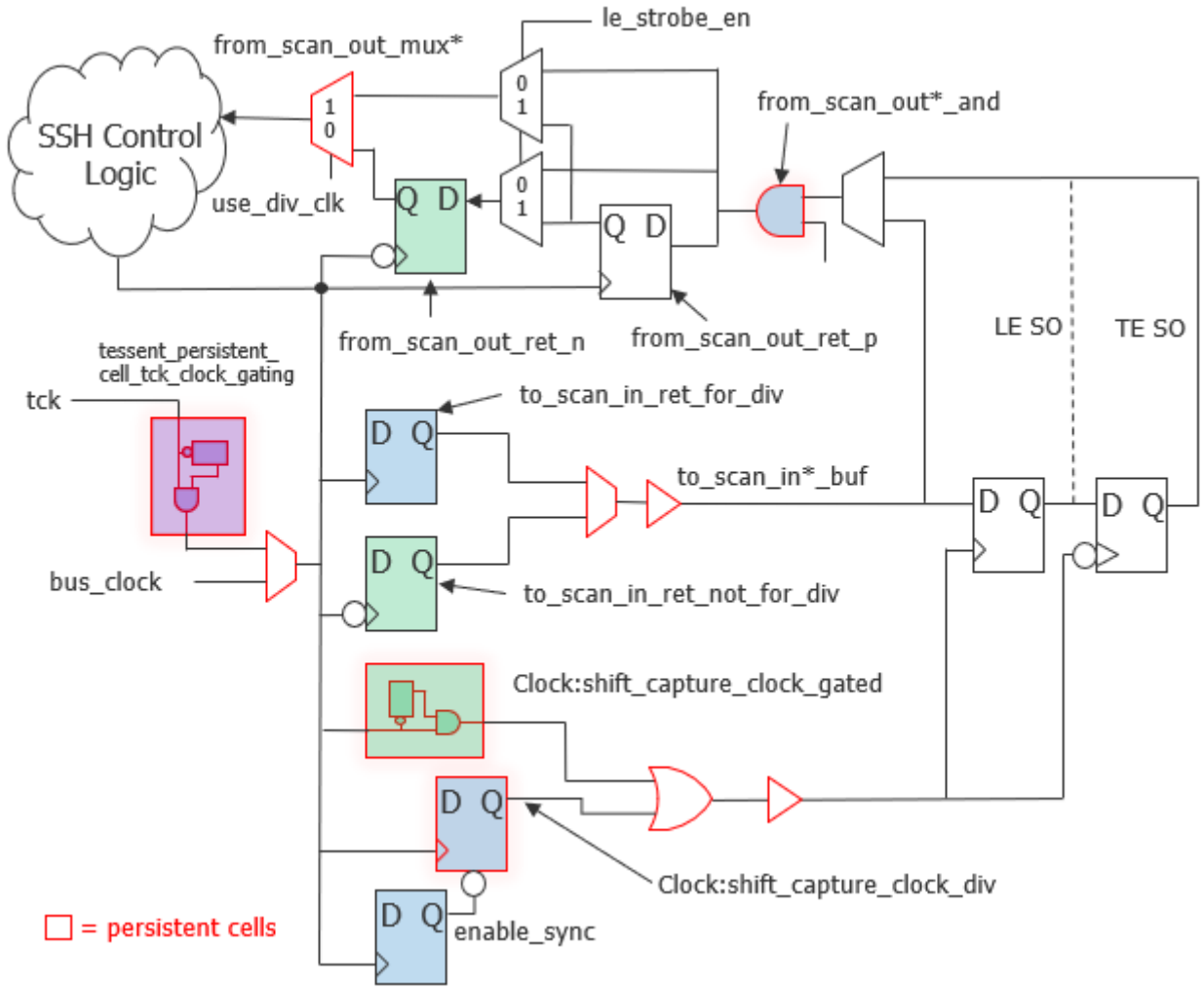
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-112. SSH Circuitry Creating Scan Control and Clock Signals



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-113. SSH Scan Datapaths Interacting With Scan Chains



The patterns then apply an ssn_bus_clock whose period is the target shift_clock period divided by *N*. If the resulting clock is faster than the maximum bus clock speed, its frequency is reduced to that maximum frequency. For example, suppose you specified an ssn_bus_clock_period of 2.5 ns and a shift_clock period of 12 ns with the set_unload_timing_options command, and that the packet size (*N*) is four bus clock cycles. The applied bus clock period becomes 3 ns, because 3 ns * *N* = 12 ns. When the packet is wider than six bus clock cycles, the SSN bus period can be 2.5 ns because 2.5 ns * 6 = 15 ns, which is larger than 12 ns.

The scan_en and edt_update signals are generated from a posedge flop when running scan with a divided SSN bus clock, and from a negedge flop when running scan with a gated SSN bus clock; this guarantees a setup and hold margin both of 0.5 shift_clock cycles around the active posedge of the chain flop. The scan_en and edt_update scan controls can be programmed with extra cycles around their transition edges. This is again controlled by the set_load_unload_timing_options command and reflected with set_multi_cycle_path commands in the SDC file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The SSH scan chain SI/SO interface also features negedge and posedge scan flops, which support both gated and divided scan clocks configurations, as shown in Figure 13-113 on page 4501. The objective of this interface is to always provide negedge-retimed shift_clock data going to the chain SI flops and a full cycle of shift clock for data coming back from the chain SO, for both gated and divided shift_clocks, and for both LE (leading Edge) and TE (Trailing Edge) chain SO flops. The chain SO path is a full-cycle timing loop starting from the shift clock source pin, going through the chain SO flop, and going back to the from_scan_out sampling flop.

Generating the SSN Scan Clocks Using the Clock Shaper Cell

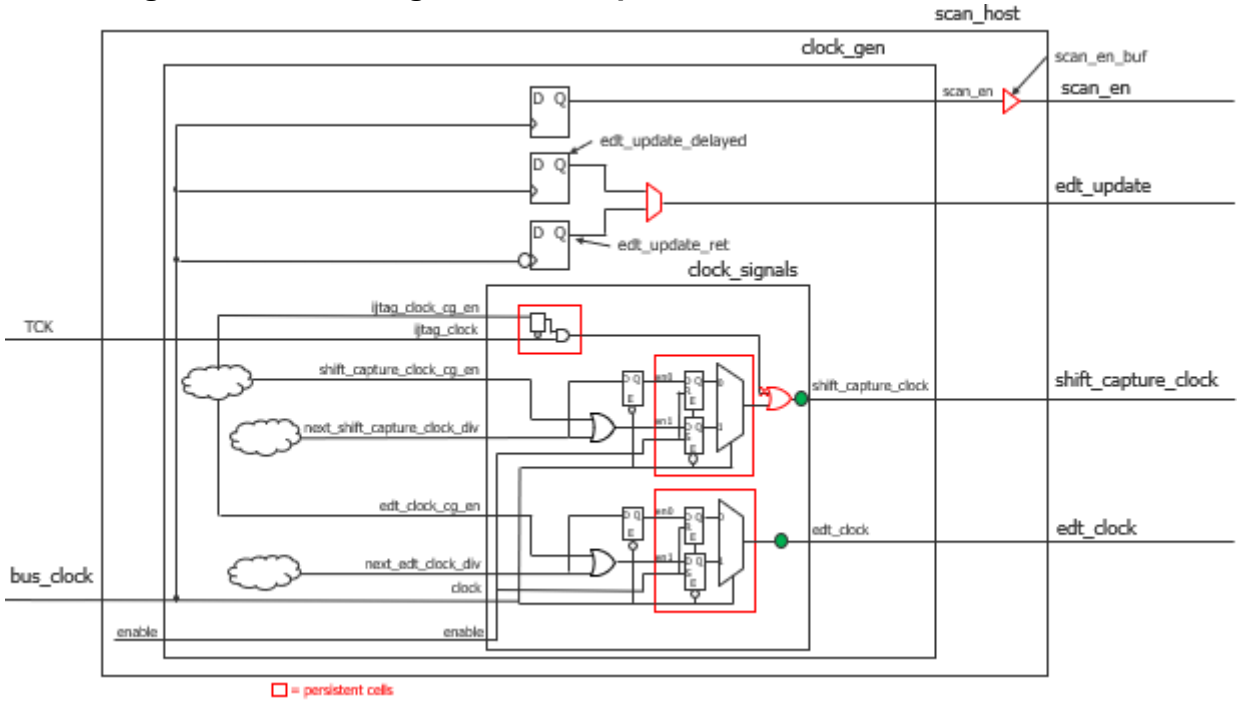
Figure 13-112 shows how the shift_capture_clock and the edt_clock can be gated versions of bus_clock when the packet is less than two clock cycles wide. It also shows how they can be divided versions of the bus_clock when the packet is two or more clock cycles wide. This behavior is achieved by using a flop and a clock gater ORed together, ensuring only one of them is high at any given time. The advantage of this circuitry is that the implementation can use only common standard cells. The disadvantage is that you must define two physically exclusive generated clocks in the SDC, which increases the burden of timing closure.

You can specify the use_clock_shaper_cell property as “on” to reduce the timing closure burden. When you do this, the tool implements the clock signals module of the ScanHost using one or more clock shaper cells as shown in Figure 13-114. Refer to the section “Clock Shaper Cells” on page 4542 in “RtlCells” to understand the operation and design of the this cell. As illustrated in Figure 13-127 on page 4544, the cell can generate a gated and divided version of the bus clock using a single timing path. Therefore, you can constrain both modes of operation with a single SDC generated clock, which prevents creation of reconvergent clock networks and reduces the timing closure burden.

The trigger for the capture pulse from the TCK clock, when you use external capture, is the output of clock gater ORed with the output of the clock shaper. This is an asynchronous timing mode, and the TCK clock is prevented from propagating to the scan cell in SDC with a set_disable_timing command. In the following figure, this appears as a red X on the second input of the OR gate. Refer to the section “External Capture With SSN” on page 4508 for more information about external capture operation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

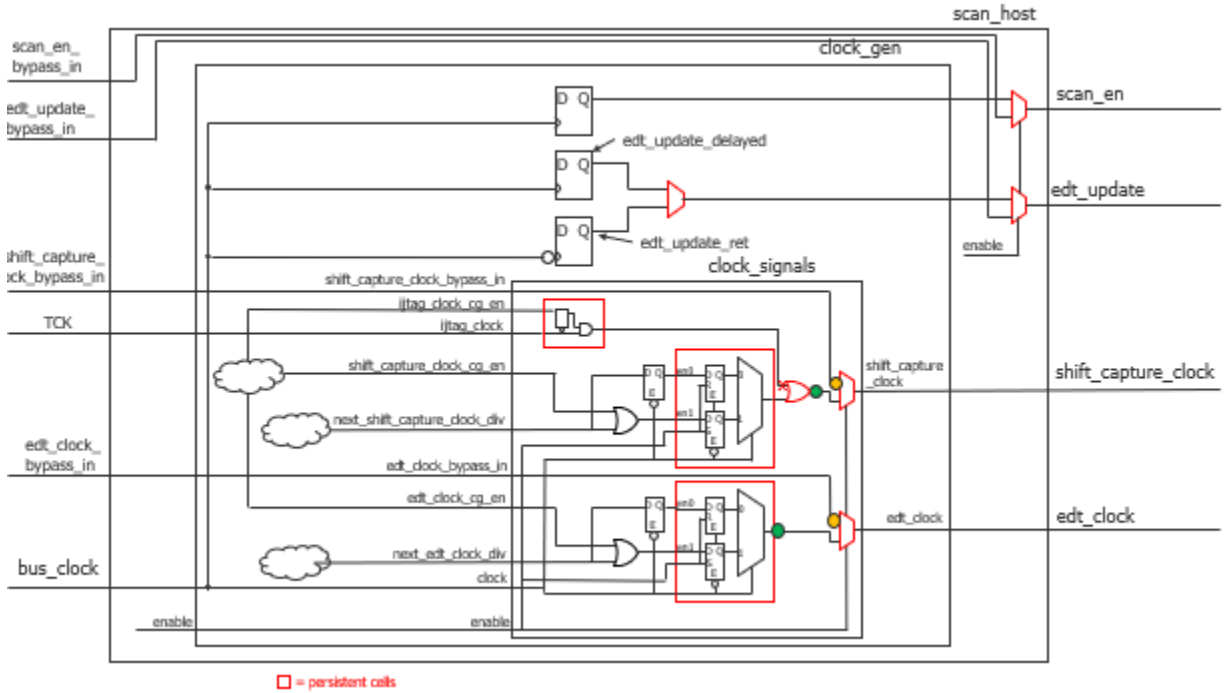
Figure 13-114. Using a Clock Shaper Cell To Generate Scan Clocks



If you implement SSN over a legacy scan access approach where you set the scan_signals_bypass property to “on”, as shown in Example 2 of this topic, there is an extra clock multiplexer on the two clocks to inject your legacy clock in that mode. The legacy clock is selected as soon as the ScanHost node is not enabled, as shown in Figure 13-115. To properly constrain this legacy mode, the tool creates an extra SDC generated clock, illustrated with yellow circles in the figure. It is possible to generate only the test_clock within the ScanHost node and use DFT signal clock gating cells after the ScanHost to recreate the shift_capture_clock and the edt_clock, as described in the section “Generating shift_capture_clock and edt_clock With DFT Signals After the ScanHost Node” on page 4504. In that case, if you are reusing the SSN bus_block as your test_clock during the legacy bypass mode, you can avoid the presence of the extra multiplexer by setting the use_ssn_bus_clock_as_test_clock_bypass property to “on” as shown in Figure 13-117.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-115. Using a Clock Shaper Cell To Generate Scan Clocks With Bypass Mode



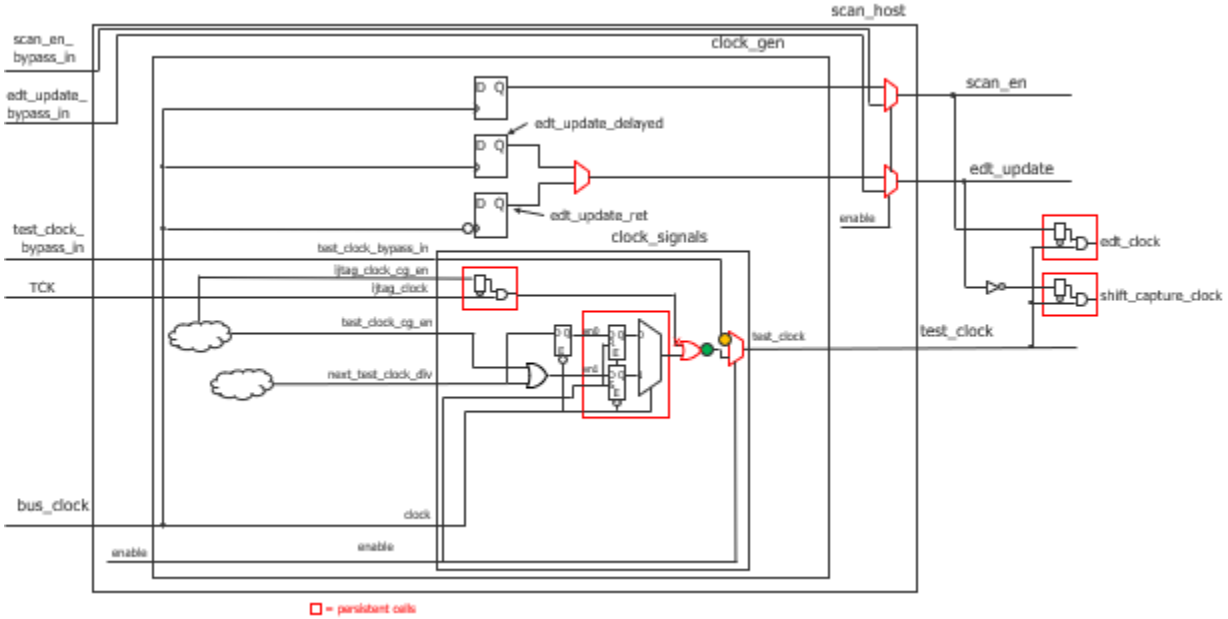
Generating shift_capture_clock and edt_clock With DFT Signals After the ScanHost Node

When the ScanHost node generates the test_clock, it no longer generates the shift_capture_clock and edt_clock by default. Instead, the tool inserts DFT signal clock gaters after the ScanHost to re-create the shift_capture_clock and the edt_clock from the generated test_clock. This makes it easier to balance the edt_clock and shift_capture_clock when you want to maximize the shift clock rate. Activate this new mode by either specifying the "set_dft_specification_requirements -add_test_clock_on_ssh on" or by setting the test_clock_present property in the ScanHost/Interface/ChainGroup wrapper to "on". Figure 13-116 shows the generation of the test_clock on the ScanHost and the DFT signal clock gaters after it to re-create the shift_capture_clock and edt_clock. The tool automatically creates DFT signal clock gaters in the same parent instance as the ScanHost node as soon as they are needed by an add_dft_control_points command or by a DftSignal(<*XXX*>) connection value within the EDT or OCC Connection wrappers.

You can also use local DFT signal clock gaters within the parent instance of the OCC and EDT controllers by using the "add_dft_signals -parent_instance" command in setup mode. The DftSignal(<*XXX*>) connection value in the OCC and EDT wrappers selects the DFT signal source that is in or above its parent instance. Example 4 illustrates this use model.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-116. ScanHost Node Sourcing test_clock With Following DFT Signal Clock Gater

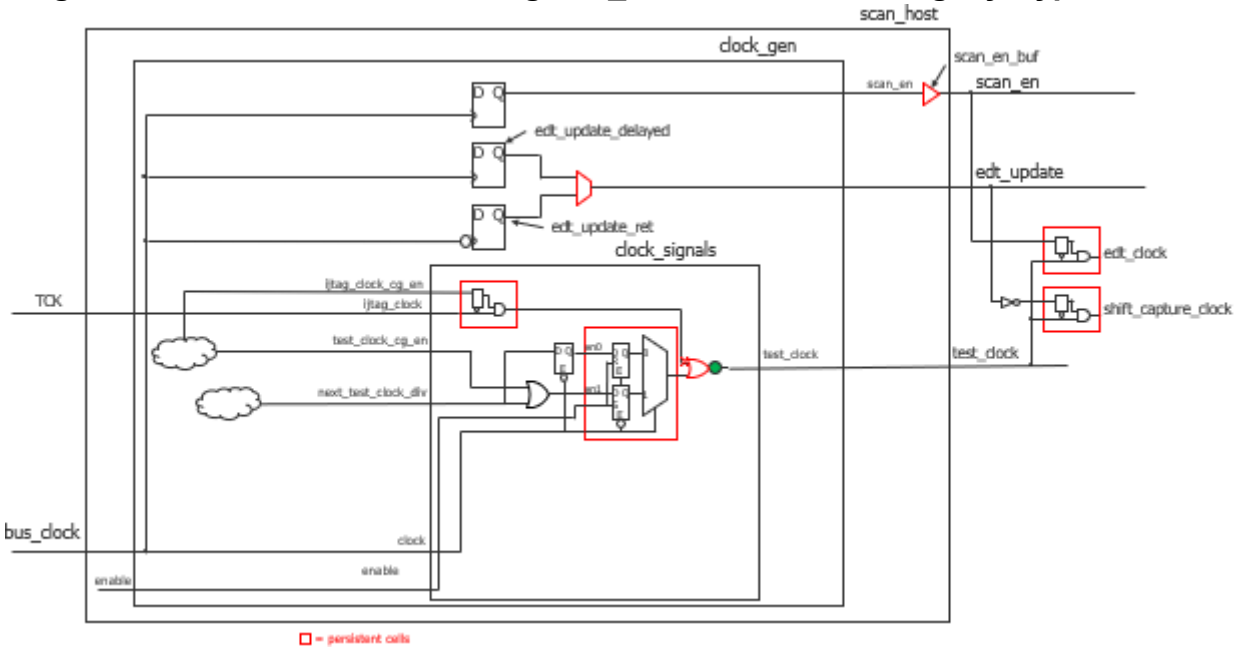


In the example in the preceding figure, the scan_signals_bypass property is “on” to enable the legacy bypass mode. To properly time this mode, an SDC generated clock is defined on the second input of the multiplexer, illustrated with the yellow circle.

When implementing the legacy bypass mode, it is also common to reuse the bus_clock signal as the test_clock for the legacy mode. The bypass mux on test_clock becomes redundant when the ScanHost generates only the test_clock, because both inputs have the same source. In this case, simply set the use_ssn_bus_clock_as_test_clock_bypass property to “on”, and both the bypass multiplexer and the extra SDC generated clock are removed, as shown in the following figure.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-117. ScanHost Sourcing test_clock Reused for Legacy Bypass Mode



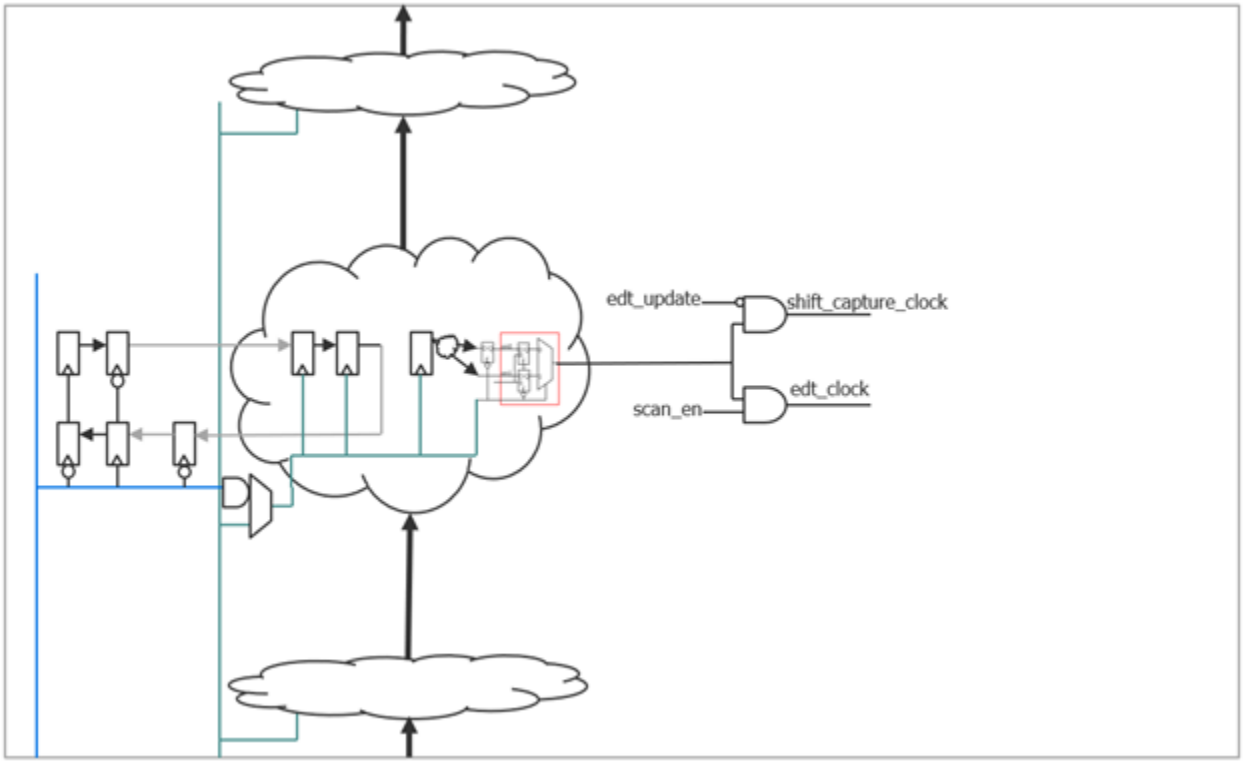
SSN Bus Clock Tree Synthesis

The SSN bus clock network is a simple and standard network. It is typically at least a balanced clock tree within a physical region. You can handle skew when crossing clock domain regions by using a pair of BusFrequencyDivider/BusFrequencyMultiplier nodes or the Fifo node. Refer to these reference pages and the topic "Types of Clock Networks To Use With SSN" in the "Advanced Topics" section of the *Tessent Shell User's Manual* for more information about how to do this.

Consider a single clock tree region where the ScanHost node resides, and a physical block where the SSN bus enters and feeds a pipeline stage before going to the ScanHost and exiting the physical block through another pipeline stage. Figure 13-118 illustrates the SSN bus clock tree with the green clock network. The cloud of logic at the bottom represents the input pipeline stage, and the cloud of logic at the top represents the output pipeline stage, while the middle one represents the ScanHost node. A multiplexer appears in the ScanHost node that the SSN bus clock network goes through. When you operate the ScanHost through the SSN bus, you must balance the SSN bus clock tree through that multiplexer for the input and output pipeline stages to be synchronous with the bus registers within the ScanHost.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-118. SSN Bus Clock Tree When Streaming Through the Bus

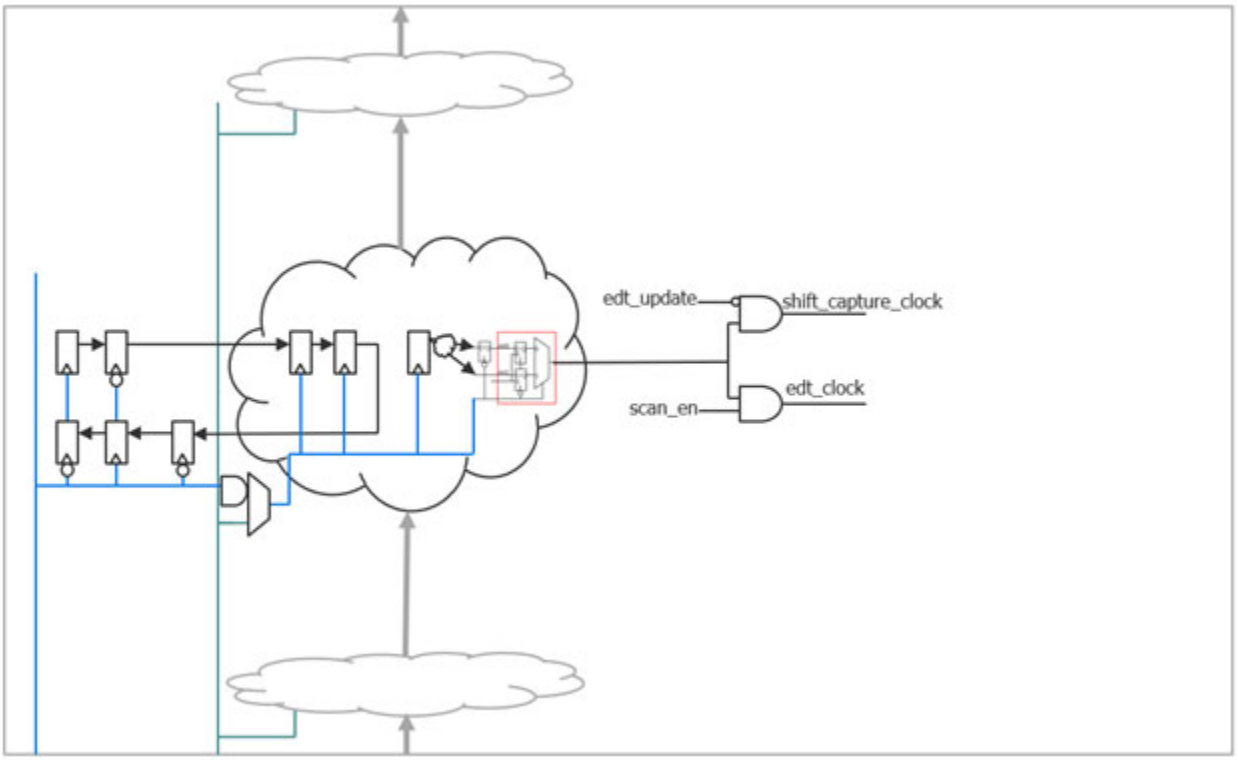


The ScanHost can also operate in Streaming-Through-IJTAG mode, where the streaming data is provided to the ScanHost through the IJTAG network. In this mode, the multiplexer is switched to select the gated version of TCK to drive the logic within the ScanHost node. Figure 13-119 illustrates this clocking mode.

In this mode, the timing paths that Figure 13-118 shows as dimmed are now activated and must work at the TCK clock rate. Those are the paths between the TCK and the ScanHost. Clock tree synthesis balances TCK within the ScanHost as it enters through that multiplexer by providing an early TCK branch to compensate for the clock latency after the multiplexer. The TCK and SSN bus clocks are physically exclusive, which is why the timing path between the blue and green clocks are automatically excluded from timing. The multiplexer switches when the SSN bus clock is held at zero at the source and when TCK is still gated using the clock gater in front of the multiplexer. The clock gater turns on when the ijtag_se signal goes up on the IJTAG scan load following the one that activated Streaming-Through-IJTAG mode.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-119. ScanHost Bus Clock Tree When Streaming Through IJTAG



External Capture With SSN

When you perform scan retargeting, the cores are fully isolated and the ScanHost node internally generates the capture sequence. Data streaming continues while each core stops shifting, toggles its local scan_en, and issues a set of capture clock pulses. For the top-level ATPG mode, if you want to include the top-level I/O as part of the ATPG mode, you use SSN in external capture mode. In this mode, the ScanHost nodes are used only to shift the scan data in and out of the EDT controller and to bring the scan_en signal down to be ready for capture. Once this time is reached between each scan load, the SSN bus clock is frozen and the capture clocking and the control and observation of the PIs and POs is returned to ATPG through top-level pins. If you have clocks that are not directly controlled by PIs (for example, because they come from an internal PLL), then you must provide a bypass path for the clock so that ATPG can pulse it from the outside. The ScanHost sensitizes the TCK clock onto the test_clock output port when it enters the external capture state. Figure 13-120 shows this with the TCK branch that becomes ORed with the test_clock. A clock gater enable by E2 ensures the test_clock is zero before TCK pulses are permitted to be ORed into the test_clock fanout. The timing of this inject path is not important, which is why timing propagation is disabled through that path (as illustrated by the red X). You can enable this path with TCK defined to check for intra-domain hold time violations when the TCK clock is used as the clock source for stuck-at capture cycles.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-120. TCK as an External Capture Clock

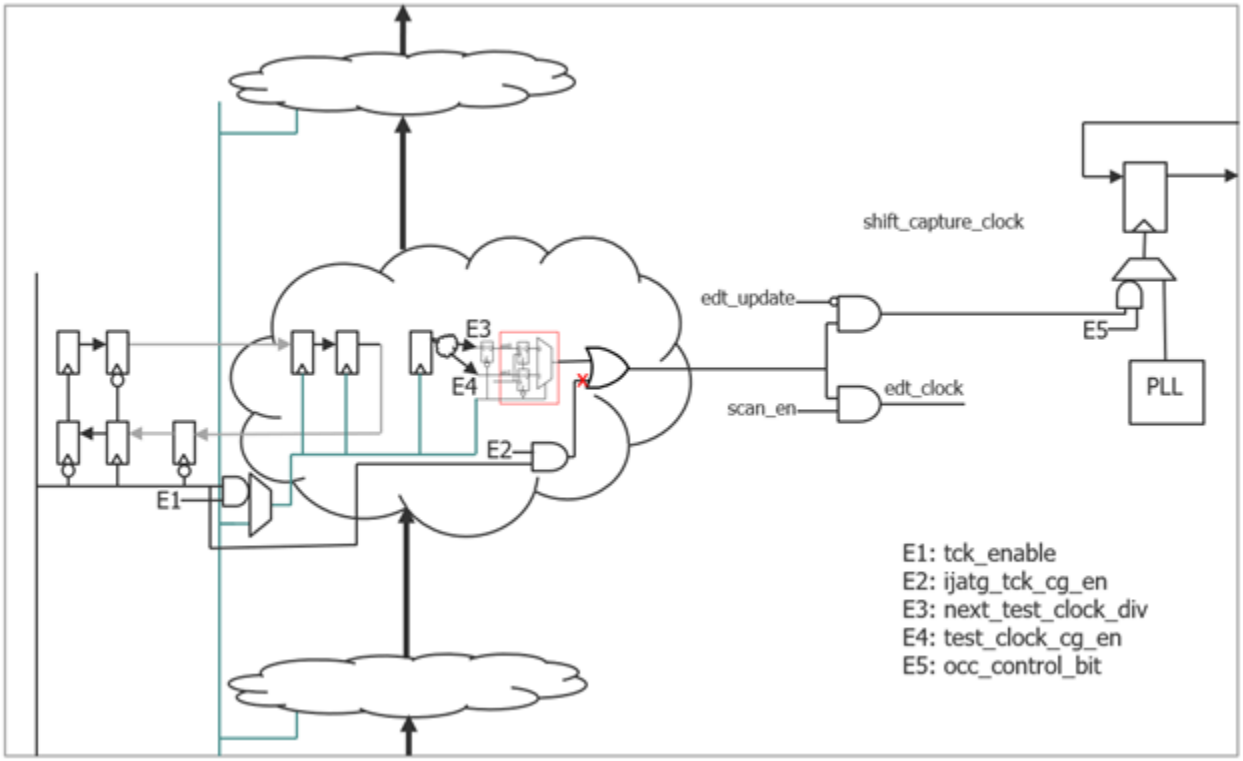
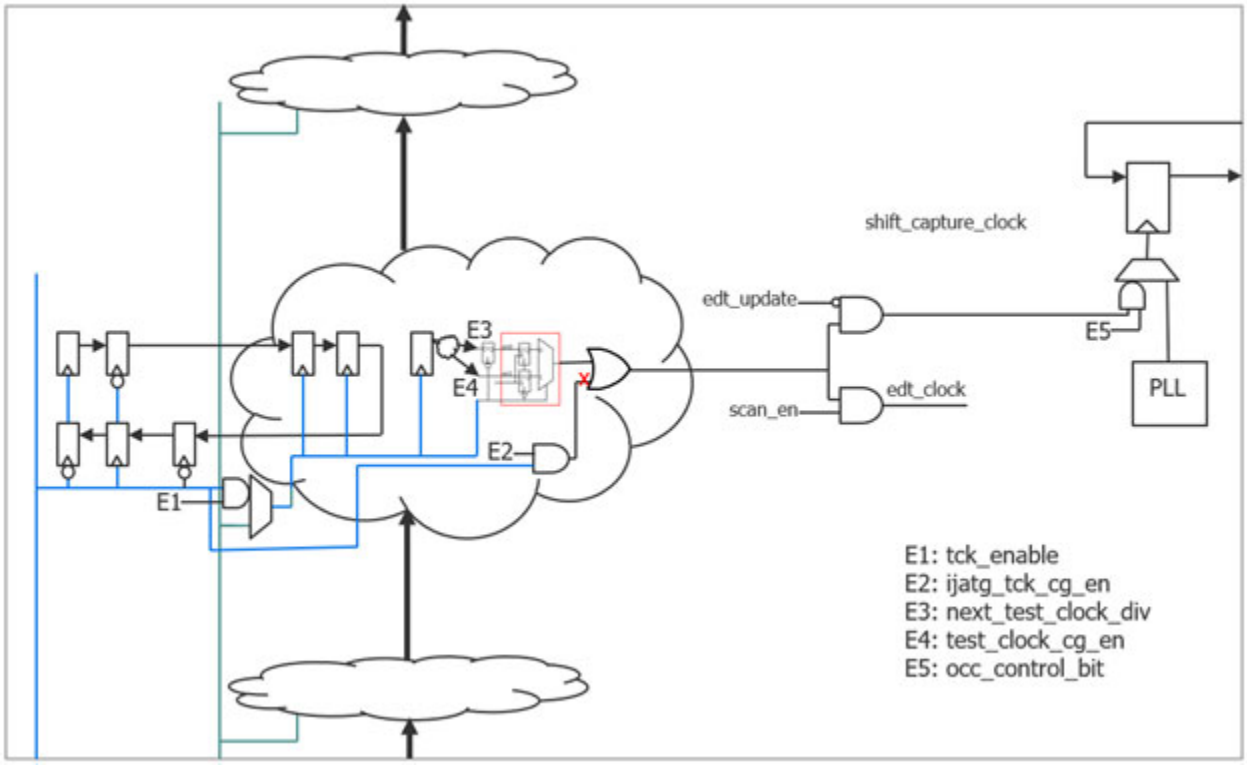


Figure 13-121 shows the same external capture operation when using Streaming-Through-IJTAG mode. In this case, TCK is used both for shifting and for injecting external capture clock pulses. The control of E1 versus E2 ensures that the TCK pulse intended for shifting does not reach the OR gate with test_clock and vice versa. To achieve this requirement, the IJTAG network is parked into its pause_dr state between each scan_load so that TCK can be pulsed without affecting the content of the IJTAG registers during the capture phase.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-121. TCK as an External Capture Clock When Streaming Through IJTAG



SSN Patterns With Timing Matching the SDC Settings

One objective of the write_patterns command is to set the pattern's clock and data timing parameters so they are consistent with the parameters that synthesis and timing closure use. It must meet the parameters specified with the set_load_unload_timing_options and set_external_capture_options commands when generating the patterns. You also use these commands in your SDC flow to set the timing parameters. It is recommended to store these two commands in a separate file and source the files for your SDC in the synthesis and layout tools. That way, you can source the same file in Tessent Shell to configure the patterns consistently. When you import these set_* commands for a child physical block into Tessent Shell, call the set_current_physical_block command to associate these setting commands in the sourced file with a given child physical block, as in the following example dofile:

```
set_current_physical_block -icl_module blockA
source <setting_files_used_for_SDC_of_blockA>
set_current_physical_block -icl_module blockB
source <setting_files_used_for_SDC_of_blockB>
set_current_physical_block -current_design
source <setting_files_used_for_SDC_of_current_design>

write_patterns …
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- ijtag_host_interface

  ijtag_host_interface : *host_node_leaf_name*;

  A property that inherits its value from the ijtag_host_interface property in the Datapath wrapper. It has the exact same format. The default is Sib(ssn). You specify this property here when you want to use a different ijtag host interface for each node, but it is unlikely that you need to do that. To precisely control the ordering of the IJTAG network for the SSN nodes, specify the ijtag_connection_order property for each node sharing a common *host_node_leaf_name* value.

- ijtag_connection_order : *int* | “”;

  An optional property that controls the connection order for the IJTAG scan interface of this node relative to the others sharing a common *host_node_leaf_name* value. When left unspecified, it defaults to a null string, which means that the connection order for the IJTAG scan interface matches the relative order of the node with respect to the others in the DftSpecification. The IJTAG scan chain gets connected from the bottom node to the top node in the SSN wrapper. If you specify an integer value for one node, then you must specify it for all other nodes having the same *host_node_leaf_name* value. This integer value must be unique for each node. The node with the smallest ijtag_connection_order value is closest to tdo.

- bus_clock_period : *time*;

  This property is used to set the bus clock period to be used by the run_synthesis command when synthesizing the SSN nodes in isolation. When unspecified, it inherits its value from the DefaultChildConfiguration wrapper in its ancestry wrappers. When synthesizing the SSN nodes with the rest of the functional logic, the SDC procs created by the extract_sdc command which is automatically called by the extract_icl command is used. The ssn_bus_clock_period for the entire physical block is configured using the set_load_unload_timing_options command. Refer to “ScanHost Scan Interface Operation and Timing Requirements” on page 4498 for a description of how to keep a common definition of those settings for both SDC and pattern generation.

- max_capture_clock_pulses : *int*

  A property that specifies the maximum number of ScanHost-generated clock pulses on shift_capture_clock/test_clock during capture. When unspecified, it inherits its value from the DefaultChildConfiguration wrapper in the ancestry of the ScanHost node.

  int — The integer value may be 1, 3, 7, 15, or 31. If the property was not specified in any DefaultChildConfiguration wrapper in the ScanHost node's ancestry, it defaults to 7.

- max_capture_to_shift_clock_period_ratio : *int*

  A property that specifies the maximum ratio between the slow capture clock and the shift capture clock. Typically, the slow capture clock is programmed to have the same high pulse width as the shift clock. The default periods of the slow capture clock (40 ns) and the shift capture clock (10 ns) ensures the pulses are separated by the equivalent of 4 shift clock periods to in turn ensure safe capture staggering. These default clock periods for the slow

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

capture clock and shift capture clock result in a max_capture_to_shift_clock_period_ratio of 4.

When unspecified, it inherits its value from the DefaultChildConfiguration wrapper in the ancestry of the ScanHost node.

int — The integer value may be 2, 4, 8, 16, 32, 64, or 128. If the property was not specified in any DefaultChildConfiguration wrapper in the ScanHost node's ancestry, it defaults to 8.

You can change the ratio of the slow capture clock to the shift capture clock using the set_capture_timing_options and set_load_unload_timing_options commands. Change the value of the max_capture_to_shift_clock_period_ratio property if the ratio of those two clock periods exceeds the default ratio of 8.

In the default SSH implementation, the slow capture clock and shift capture clock are the same output pin, shift_capture_clock. You can create the SSH with separate slow capture clock and shift capture clock output pins. For more information about changing this implementation, see the Interface wrapper section in this topic.

- parent_instance : *instance_name*;

  A property that specifies the parent instance in which the given node should be instantiated into. When unspecified, the node is instantiated directly into the current design.

- leaf_instance_name : *leaf_instance_name*;

  A property that specifies the leaf instance name of the node. When unspecified, the leaf instance name defaults to the module name with the "_inst" suffix appended to it.

- max_scan_chain_length : *int* | auto;

  A property that specifies the maximum length of the ScanHost-driven scan chains. When unspecified, it inherits its value from the DefaultChildConfiguration wrapper in the ancestry of the ScanHost node. If the property was not specified in any DefaultChildConfiguration wrapper in the ancestry of the ScanHost node, it defaults to auto. In this case, max_scan_chain_length is set to 100,000 times the total number of chain inputs specified by the chain_input_count properties outside or inside the ChainGroup wrapper. The 100,000 value is a worst-case condition for the number of scan cells that can exist behind each SSN to_scan_in output, based on a chain length of 500 and a compression ratio of 200. You do not need a single chain scan mode when using SSN because the ScanHost nodes can drive any number of scan chains, even when the SSN bus is programmed to be a single bit.

  Note

  When changing the value of this property, you should consider the lengths of the scan chains in the parent or child block as well. If the lengths of the scan chains in those blocks exceed this value, the tool reports an SSN_R10 error.

- max_scan_en_mcp : 2 | 4 | 8 ;

  A property that specifies the maximum multicycle path delay you may need for the transition of scan_en. This number is the sum of the number of setup and hold cycles

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

needed. When unspecified, it inherits its value from the DefaultChildConfiguration wrapper in the ancestry of the ScanHost node. If the property was not specified in any DefaultChildConfiguration wrapper in the ancestry of the ScanHost node, it defaults to 4, which is more than enough to make sure the scan enable transition delay is never the shift clock period limiting factor.

- max_edt_update_mcp : 2 | 4 | 8 ;

  A property that specifies the maximum multicycle path delay you may need for the transition of edt_update. This number is the sum of the number of setup and hold cycles needed. When unspecified, it inherits its value from the DefaultChildConfiguration wrapper in the ancestry of the ScanHost node. If the property was not specified in any DefaultChildConfiguration wrapper in the ancestry of the ScanHost node, it defaults to 4, which is more than enough to make sure the edt_update transition delay is never the shift clock period limiting factor.

- input_chain_count : *int* | from_edt_controller;

  A property that specifies the number of chain inputs you want the ScanHost node to drive. When specified within a ChainGroup(*id*) wrapper, the count applies to the chain count of that chain group. The property existing outside the ChainGroup wrapper is ignored as soon as one ChainGroup wrapper is specified. When you build and insert the ScanHost node at the same time as the EDT controllers, you typically do not need to specify this property as its default value is the count from the EDT controllers. If you want to have a ChainGroup for the EDT controllers and one for an uncompressed scan chain to collect the OCCs, insert the two chain group wrappers as shown in "Example 1" on page 4523. The number of input channels can be programmed at runtime if you build the EDT controller with the separate_control_data_channels property set to on. You do not have to predict how many channels you should optimally use during ATPG as the number of channels can be scaled down and optimized when it is time to create the final patterns.

  When you have several ChainGroup wrappers, the input_chain_count property specified outside the ChainGroup wrapper has no meaning, and the input_chain_count property specified within the ChainGroup wrapper must be greater than 0. When you have no ChainGroup wrapper, the input_chain_count property specified outside the ChainGroup wrapper must be greater than 0.

- output_chain_count : *int* | from_edt_controller | same_as_input_chain_count;

  A property that specifies the number of chain outputs you want to observe from the ScanHost node. When specified within a ChainGroup wrapper, the count applies to the chain count of that chain group. The property existing outside the ChainGroup wrapper is ignored as soon as one ChainGroup wrapper is specified. When you build and insert the ScanHost node at the same time as the EDT controllers, you typically do not need to specify this property as its default value is determined by the EDT controller. If you want to have a ChainGroup for the EDT controllers and one for an uncompressed scan chain to collect the OCCs, insert the two chain group wrappers as shown in "Example 1" on page 4523.

  When you have several ChainGroup wrappers, the output_chain_count property specified outside the ChainGroup wrapper has no meaning, and the output_chain_count property

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

specified within the ChainGroup wrapper must be greater than 0. When you have no ChainGroup wrapper, the output_chain_count property specified outside the ChainGroup wrapper must be greater than 0.

- output_chain_count_in_on_chip_compare_mode : *int* | from_edt_controller | from_edt_controller_high_compression_configuration;

  A property that specifies the number of chain outputs you want to observe from the ScanHost node in on-chip compare mode. The on-chip compare logic can be costly, so limit this value to a working minimum by limiting the number of output channels on the EDT controller and using the default setting of this parameter. Because the number of output channels cannot be scaled down and optimized at run time, do not specify a count in the EDT controller that is too large, especially when using on-chip compare, as the expect and mask values for each output channel must be sent.

  You can run some cores with and without on-chip compare enabled, depending on configuration of the chip. In such cases, create the EDT for that core with a dual mode with the minimum number of output channels in high compression mode. Use high compression mode for on-chip compare mode, and the lower compression mode of the EDT when not using on-chip compare mode. To enable high compression, run the command "set_core_instance_parameters {inst} -param_values {edt_configuration high_compression}" in setup mode.

  When set to the default value of "from_edt_controller" the output_chain_count_in_on_chip_compare_mode property takes the output_channel_count value specified in the wrapper for the EDT "Controller" on page 3871.

  When set to "from_edt_controller_high_compression_configuration" the output_chain_count_in_on_chip_compare_mode property takes the output_channel_count value specified in the HighCompressionConfiguration wrapper, regardless of the value of the main controller wrapper. This means that if the HighCompressionConfiguration wrapper specifies a lower output_channel_count than the main controller wrapper, the output_chain_count_in_on_chip_compare_mode property takes the lower value.

  Note

  When you specify a number for the *int* parameter of the input_chain_count, output_chain_count, and output_chain_count_in_on_chip_compare_mode properties, the number must not exceed 10,000.

  The sum of all the output_chain_count_in_on_chip_compare_mode properties must be equal to or smaller than the sum of all ScanHost input_chain_count properties. This enables testing of the output comparators in loopback mode. In the rare case where you have more output chains than input chains in on-chip compare mode, increase the input_chain_count property value to satisfy this rule. The additional input chains are only used in loopback test when testing the comparators.

  This property is ignored unless the OnChipCompareMode/present property is on. When you have several ChainGroup wrappers, the output_chain_count_in_on_chip_compare_mode property specified outside the ChainGroup wrapper has no meaning. The

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

output_chain_count_in_on_chip_compare_mode must be equal or smaller than the output_chain_count property.

- high_compression_input_channel_count : *int* | auto;

  A property that specifies the number of channel inputs related to the dual-compression EDT high-compression configuration to be connected to the ScanHost to_scan_in_bypass_in port when the scan_signals_bypass property is set to on. This property does not affect the SSN operation mode.

  If you specify this property inside a ChainGroup(<*id*>) wrapper, the count applies to the channel count of that particular chain group. If you specify this property inside any ChainGroup wrapper, the tool ignores any value set outside of the ChainGroup wrappers.

  If you build and insert the ScanHost node at the same time as the EDT controllers, this property is usually not necessary, because the default value is the count from the EDT high compression configuration.

  The tool ignores the value of this property when use_high_compression_channel_count_in_bypass_mode is set to off.

- high_compression_output_channel_count : *int* | auto;

  A property that specifies the number of channel outputs related to the dual-compression EDT high-compression configuration to be connected to the ScanHost to_scan_in_bypass_out port when the scan_signals_bypass property is set to on. This property does not affect the SSN operation mode.

  If you specify this property inside a ChainGroup(<*id*>) wrapper, the count applies to the channel count of that particular chain group. If you specify this property inside any ChainGroup wrapper, the tool ignores any value set outside of the ChainGroup wrappers.

  If you build and insert the ScanHost node at the same time as the EDT controllers, this property is usually not necessary, because the default value is the count from the EDT high compression configuration.

  The tool ignores the value of this property when use_high_compression_channel_count_in_bypass_mode is set to off.

- use_high_compression_channel_count_in_bypass_mode : on : off;

  A property that specifies whether to us the high-compression configuration EDT channels from dual-compression EDT to connect to_scan_in_bypass_in and from_scan_out_bypass_out ports. The tool generates the interface with capability to connect the high-compression configuration only in this case.

  If the high-compression configuration is not available, the tool uses channels from the single configuration. If no EDT controller is present, the tool uses values from the input and output chain count.

  If this property is set to off (default), bypass mode reflects the input and output chain counts.

  If you specify this property inside a ChainGroup(<*id*>) wrapper, the behavior applies to that particular chain group. Otherwise, the value is inherited from the root wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- support_from_scan_out_le_strobing : on | off;

  A property that specifies whether or not the ScanHost is equipped with hardware to enable chain groups to optionally strobe on the leading edge (LE) of the scan clock, instead of the default trailing edge (TE). This property, when found within the ChainGroup wrapper, gets its default value from the same property specified outside the ChainGroup wrapper.

  - If all scan sources driving the ScanHost are clocked on the TE, the ScanHost should strobe on the TE. This allows for a full cycle of propagation, as described in the ScanHost Scan Interface Operation and Timing Requirements section. In this case, the property may be turned off but may also be left on.
  - If those scan sources are clocked on the LE, the ScanHost must strobe on the LE.
  - If there is a mix of LE and TE scan sources, the ScanHost should strobe on the LE.

    Note

    In this case, you only have a half cycle of propagation time from the TE scan elements, which may impact your scan shift speed.

  With this property enabled by default, the ScanHost can be automatically programmed during pattern generation to strobe on either the LE or TE, as required. This option provides the most flexibility. Only consider turning it off if you are certain the scan sources driving the ScanHost are TE.

- scan_signals_bypass : on | off;

  A property that specifies the inclusion of a bypass multiplexer in the ScanHost node to allow implementing SSN while keeping the traditional star connections of the EDT channels. Use this property to integrate LogicBIST with SSN. Specify the controls_only option if star connections of the EDT channels are not needed.

  Possible values are:

  on — Specifies the inclusion of bypass multiplexers in the ScanHost node for EDT channels and for scan control signals. This setting is supported in the presence of LogicBIST.

  off — Specifies that bypass multiplexers are not included in the ScanHost. This value is not allowed when the LogicBIST wrapper is present in a DftSpecification or when the EDT/LogicBistOptions/present property is enabled.

  controls_only —Specifies the inclusion of bypass multiplexers for scan control signals needed for LogicBIST operation, but skips the creation of scan channels bypass multiplexers.

  auto — Evaluates to controls_only when EDT controllers specified in a current processing run has LogicBistOptions/present property enabled or the LogicBist/Controller wrapper is present, otherwise it evaluates to off. This is the default behavior.

  See “Example 2” on page 4525 for a complete example of using this property.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- use_clock_dff_cell : on | off;

  An optional Boolean property that specifies to use clock DFF cells with the ScanHost node. This property is inherited from the DefaultChildConfiguration wrapper, where it defaults to on. Refer to the description in the DefaultChildConfiguration wrapper for full details on this property.

- use_clock_or_cell : on | off;

  An option Boolean property that specifies to use clock OR cells with the ScanHost node. This property defaults to on. If your library does not define a clock OR cell, set this property to off. In this case, the ScanHost node uses normal OR cells instead; however, you may need to replace it with a true clock OR cell later in the physical flow.

  Note

  When you use the clock shaper, this property has no impact. For more information about the clock shaper, refer to "Generating the SSN Scan Clocks Using the Clock Shaper Cell" on page 4502.

- use_clock_shaper_cell : on | off | auto ;

  An optional Boolean property that specifies to generate the selected clocks (test_clock, shift_capture_clock, shift_clock, capture_clock, and edt_clock) using a clock shaper cell, rather than a flop and a clock gater ORed together. Refer to Figure 13-112 on page 4500 to see how the clocks are generated without the clock shaper cell. Refer to Figure 13-114 on page 4503 to see how the clocks are generated with the clock shaper cell. Refer to the section "Clock Shaper Cells" on page 4542 in the "RtlCells" reference page for further explanation of clock shaper cells.

  The advantage of using a clock shaper cell is that you do not require two generated clocks in the SDC for each clock (one gated and one divided). The clock shaper creates both clocks using the same timing path. This speeds up timing closure in large designs.

- support_output_clock_activation_when_ssh_is_off :on | off ;

  An optional Boolean property that specifies to generate a TDR that can force clock gaters open in an inactive ScanHost node. This register is called activate_clock_outputs_shift_reg, and it drives a signal called activate_clock_outputs.

  If you are not using a clock shaper cell or you are using a reset-inactive clock shaper cell, the output clocks (such as edt_clock and shift_capture_clock) are forced off when the ScanHost node is inactive. This prevents activity in inactive cores and regions of the design when using SSN. If you need the output clocks to propagate during test_setup, you can use an iWrite call to write a 1 to this TDR at the beginning of the test_setup procedure, which enables the ssn_bus_clock to reach ScanHost outputs during the rest of the test_setup procedure. At the end of the test_setup procedure, if you do not want the ssn_bus_clock to continue driving the clock output for the inactive ScanHost node when active ScanHost nodes are running, you must then write a 0 to the TDR. ScanHost nodes automatically clear this bit when they are activated.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- use_ssn_bus_clock_as_test_clock_bypass : on | off;

  An optional Boolean property that specifies that the SSN bus clock is to be reused as the test_clock for the legacy bypass mode. This property is used only when the scan_signal_bypass property is "on" and when the ScanHost sources the test_clock only where the shift_capture_clock and edt_clock are re-created using DFT signal clock gaters, as shown in Figure 13-117 on page 4506.

- dft_signals_not_mapped_on_ssh_outputs : *dynamic_dft_signal_name*, …;

  An optional property that instructs the tool not to map the specified DFT signals to the output of the ScanHost node. This can enable the specified pins to remain as the sources of their DFT signals. Use the Connections/<*dft_signal_name*> property to connect the output of the ScanHost node to a pin that sources the DFT signal where you specified it when SSN is active.

  If you do not specify this property, the fanout of the specified DFT signal is moved to the corresponding output of the ScanHost node and the DFT signal is then declared on the output of the ScanHost node. Any further connection to the DFT signal is made directly to the ScanHost output. When it is unmapped, a further connection is made to the DFT signal where you added it.

  If the DFT signal is not already added on an existing pin, the tool reports an error. If the Connection property for the DFT signal is not specified, the tool reports a warning.

- OnChipCompareMode

  An optional wrapper that specifies the on-chip compare mode functionality. See the "On-Chip Compare With SSN" section in the *Tessent Shell User's Manual* for more information.

- OnChipCompareMode/present : on | off | auto;

  A property that specifies whether the ScanHost node is to be built with the on-chip compare mode. This mode is particularly useful for wrapped cores repeated multiple times in a design. The on-chip compare mode enables the reuse of the same data for an unlimited number of cores without increasing test time.

  The on-chip compare mode is also useful if you are doing what is often referred to as core harvesting. Use this mode to correctly bin parts based on performance levels. The on-chip compare mode has a pass/fail sticky bit, observed using IJTAG at the end of the pattern set, indicating whether or not the core had any miscompares.

  A special TESSENT_PRAGMA variable identifies the pass/fail sticky bit. This enables you to identify a miscompare on the associated SSH quickly. For more information on the presence of iReadVars in the STIL file and how they are used to identify the TDO timeslots, refer to the "PDL Retargeting With Symbolic Variables" section of the *Tessent IJTAG User's Manual*.

  The following example demonstrates the annotation, where the hierarchical prefix path is the ICL instance name of the associated SSH. The annotation is directly before the vector where the compare status is read.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
 Ann {* TESSENT_PRAGMA bit_variable
core_g1_tessent_ssn_scan_host_piccpu_inst.sticky_status -type read
-var_bits {0} -pin ijtag_so -inversion 0b0 *}
 V {
    _pi_=00010101000000;
    _po_=XXXXXX;
 }
```

Note

When you set the OnChipCompareMode/present property to on while generating the ScanHost hardware, it is capable of supporting on-chip compare. However, on-chip compare is only used when turned on during ATPG or pattern retargeting. The following command enables on-chip compare on all ScanHost instances in the design:

```
set_core_instance_parameters -instrument_type ssh -parameter_values
{on_chip_compare_enable on}
```

See the SSH instrument type section of "Core Instance Parameters and Values by Instrument" on page 2403 for details on how to enable on-chip compare during ATPG or pattern retargeting.

The default "auto" value is inferred as on if you specify the OnChipCompareMode wrapper.

- OnChipCompareMode/sticky_status_resolution : ssh | output_chain ;

  A property that specifies whether the on-chip compare mode pass/fail status sticky bit is one per ScanHost or one per output scan chain. For the latter case, you can also specify whether you check the global status or the status of individual output chains using two independent core instance parameters:

  - sticky_status_global 0 | 1
  - sticky_status_per_chain_output 0 | 1

  Note

  When you set OnChipCompareMode/sticky_status_resolution to output_chain, the pass/fail status sticky bit for the ScanHost is also present, in addition to the per output scan chain bits.

  Use the "set_core_instance_parameters -parameter_values *parameter_list*" switch to specify the above parameters on all ScanHost instances or a set of ScanHost instances. When you also specify "-instrument_type ssh", those parameters are only set on ScanHost instances supporting on-chip compare. The sticky_status_per_chain_output parameter is only set on the ScanHost instances that have the sticky_status_resolution property set to output_chain and is on by default.

  The default is ssh.

- OnChipCompareMode/status_groups : *int* ;

  A property that controls how many status groups the hardware supports. Figure 13-109 and the preceding text discuss status groups.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This feature is only required for diagnosis efficiency. If you use one status group per instance, you can collect independent diagnosis data for each instance while running the test. If you combine several instances per status group, you get the diagnosis data only if one core is seen to have miscompares in the group, which you observe at the end of the test using the sticky pass/fail flag.

You can patch the disable_on_chip_compare_contribution bit in all failing ScanHosts but one, and repeat the test to collect per core diagnosis data when there is more than one core failing within one status group.

The default is 128 so you can choose to use several status groups at the early stage of the yield curve and use only a few status groups as the yield matures.

Individual ScanHost instances can be set to contribute to a status group or not using the "set_core_instance_parameters -parameter_values on_chip_compare_group" command. See the SSH instrument type section of "Core Instance Parameters and Values by Instrument" on page 2403 for details.

- ChainGroup(*id*)

  An optional, repeatable wrapper that defines a scan chain group. When you define a chain group using this wrapper, the properties do not inherit their values from the parent ScanHost wrapper. If you do not define the properties explicitly, they use the default values. This wrapper includes the following properties:

  - input_chain_count : *int* | from_edt_controller;
  - output_chain_count : *int* | from_edt_controller | same_as_input_chain_count;
  - output_chain_count_in_on_chip_compare_mode : *int* | from_edt_controller | from_edt_controller_high_compression_configuration;
  - support_from_scan_out_le_strobing : on | off;
  - scan_signals_bypass : on | off;

- Interface

  The properties in the Interface wrapper enables you to change the name of all the ports on the node. Refer to the Interface wrapper of the DefaultChildConfiguration wrapper to see the default value of each port name. If the property within the local Interface wrapper is not specified, the value is inherited from the Interface wrapper in the first DefaultChildConfiguration wrapper found in the ancestry of the current node.

- Interface/ssh_is_active_present : on | off | auto;

  This property determines whether the ScanHost provides an ssh_is_active output port. The default value is "auto". The "auto" value resolves as "on" when the Connections/ssh_is_active property is specified and otherwise resolves as "off".

  When this property is set to "on", the ScanHost node is generated with an additional signal that indicates whether the ScanHost node is enabled and operating. When the ScanHost node is disabled, or when shifting has completed (that is, the SSH is in capture state), the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

signal is low. The signal is also low when the ssn_end proc is being applied, even though the ssn_end proc can still be unloading sticky bits from the ScanHost node.

- Interface/ChainGroup/test_clock_present : on | off | auto;

  This property determines if the ScanHost provides a combined test_clock port or separate shift_capture_clock or edt_clock ports. The default "auto" value is inferred as on if one of the following conditions is true:

  - The Connections/test_clock or Connections/ChainGroup/test_clock property is specified.
  - The "get_dft_specification_requirement -add_test_clock_on_ssh" command returns true.
  - There is at least one inferred or specified "add_dft_control_points -dft_signal test_clock" call.

  Use the test_clock connection property when inserting a ScanHost outside a physical block that already contains the clock gaters to separate a common test_clock into separate shift_capture_clock and edt_clock signals.

- Interface/ChainGroup/edt_clock_present : on | off | auto;

  This property determines if the ScanHost provides an edt_clock port. The default "auto" value is inferred as on if one of the following conditions is true:

  - The Connections/edt_clock or Connections/ChainGroup/edt_clock property is specified.
  - The ScanHost sources the dynamic DFT signals, as specified using the SSN/scan_host_sourcing_dynamic_dft_signals property, and the test_clock_present property is inferred or specified as "off".

  Use this property to turn off the edt_clock presence on the ScanHost that is sourcing the dynamic DFT signals if you do not need it.

- Interface/ChainGroup/shift_capture_clock_present : on | off | auto;

  This property determines if the ScanHost provides a shift_capture_clock port. The default "auto" value is inferred as on if one of the following conditions is true:

  - The Connections/shift_capture_clock or Connections/ChainGroup/shift_capture_clock property is specified.
  - The ScanHost sources the dynamic DFT signals, as specified using the SSN/scan_host_sourcing_dynamic_dft_signals property, and the test_clock_present property is inferred or specified as "off".

  Use this property to turn off the shift_capture_clock presence on the ScanHost that is sourcing the dynamic DFT signals if you do not need it.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Interface/ChainGroup/shift_clock_present : on | off | auto;

  This property determines if the ScanHost provides a shift_clock port. The shift_clock is a clock, not generated by default, that pulses only during shift but not during capture. This is unlike shift_capture_clock and test_clock, which pulse during both shift and capture. The shift_clock is used during load_unload. The default "auto" value is inferred as on if the Connections/shift_clock or Connections/ChainGroup/shift_clock property is specified. If this property is set to on or inferred as on, you must generate the shift_capture_clock (generated by default) or capture_clock (not generated by default). If neither of these clocks is generated when this property is on, the tool reports an error.

  You must specify shift_clock_present and capture_clock_present if both test_clock_present and shift_capture_clock_present are inferred or specified off.

- Interface/ChainGroup/capture_clock_present : on | off | auto;

  This property determines if the ScanHost provides a capture_clock port. The capture_clock is a clock, not generated by default, that pulses only during capture but not during shift. This is unlike shift_capture_clock and test_clock, which pulse during both shift and capture. The capture_clock is used during slow-speed capture. The default "auto" value is inferred as on if the Connections/capture_clock or Connections/ChainGroup/capture_clock property is specified.

  You must specify shift_clock_present and capture_clock_present if both test_clock_present and shift_capture_clock_present are inferred or specified off.

- Interface/ScanHostClockSignalModule/define_clock_on_boundary : on | off | auto;

  This property affects where the clocks are defined in the SDC and where the clocks are added during DRC. When the property is off, the clocks are defined on pins of persistent cells located within the clock signals module, as shown in Figure 13-112 on page 4500. Use this when the clock signals module is a synthesized module. When the clock signals module is a hard macro and you use a Synopsys Liberty™ model for it during synthesis, the clock must be defined on the output pin of this module. This happens when the value is specified or inferred as "on".

  The default value of "auto" means "on" when the module_name property is specified or when the write_out_module_definition property is specified as "in_separate_file" or "off".

- Interface/ScanHostClockSignalModule/module_name : *module_name*;

  This property is used to specify the module name of the clock signals module when you use a hard macro version for it.

- Interface/ScanHostClockSignalModule/write_out_module_definition : on | in_separate_file | off | auto;

  This property controls whether an RTL module definition for the clock signals module is written out. When you have a hard macro version of the clock signals module, simply specify its module name with the module_name property, and this module will be instantiated within the ScanHost module. The "auto" value of this property implies off, and

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

no RTL module definition be created for the clock signals module. In this use model, you must have read in the hard macro version in setup mode using the read_verilog command.

When you specify “in_separate_file”, the clock signals module RTL definition is written out in a separate file, as opposed to within the same file as the other modules of the ScanHost. The “auto” value means off when you do not specify the module_name property.

Examples

Example 1

The following example builds a wrapped core with a ScanHost node having two chain groups.

One chain group has only one chain and it is used to drive a single uncompressed scan chain collecting all the On-Chip clock controllers to avoid diverging too much EDT encoding ability on the OCC bits.

The second chain group is built to accommodate the EDT controller used for the internal test mode, the small EDT controller used for the external test mode, and for the multi-chain bypass mode created during scan insertion. A multiplexer is built during the ScanHost/EDT insertion pass to multiplex between the two EDT controllers. The second level of multiplexing used for the multi-chain bypass mode is inserted during scan insertion.

Note

The creation of the multi-chain bypass mode is shown here for completeness but it is no longer recommended to build this bypass mode with scan insertion when using SSN. The multiple ScanHost nodes are able to drive scan chain of different length and still achieve optimal test time using the throttling feature.

Notice that the ssh_chain_group property is specified in the EDT Controller/Connections wrappers because the ScanHost has more than one chain group. Also notice the mode_enables property in the same wrapper. The mode_enables property is needed when more than one EDT controller connects to a single ScanHost node to control the selects of the multiplexers. The referenced DFT signal must be one or more DFT signals of type scan_mode. Finally, the second ChainGroup in the ScanHost is to drive the uncompressed scan chains going through the OCCs. For that reason, its chain group id was specified as “occ” and the input_chain_count/output_chain_count property was specified as 1.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
add_dft_signals int_edt_mode ext_edt_mode
set_dft_specification_requirements -logic_test on

check_design_rules

set spec [create_dft_specification -sri_sib_list {edt ssn}]

read_config_data -in $spec -from_string {
  SSN {
    Datapath(1) {
      ijtag_host_interface  : Sib(ssn);
      output_bus_width : 48;
      ScanHost(1) {
        ChainGroup(edt) {
        }
        ChainGroup(occ) {
          input_chain_count : 1;
          output_chain_count : 1;
        }
      }
    }
  }
  EDT {
    ijtag_host_interface : Sib(edt);
    Controller(int) {
      longest_chain_range  : 100,200;
      scan_chain_count     : 800;
      input_channel_count  : 16;
      output_channel_count : 16;
      Connections {
        ssh_chain_group    : edt;
        mode_enables       : DftSignal(int_edt_mode);
      }
    }
    Controller(ext) {
      longest_chain_range  : 100,200;
      scan_chain_count     : 40;
      input_channel_count  : 4;
      output_channel_count : 2;
      Connections {
        ssh_chain_group    : edt;
        mode_enables       : DftSignal(ext_edt_mode);
      }
    }
  }
}
process_dft_specification
```

When you reach scan insertion, you define the scan modes as follow:

add_scan_mode int_edt_mode -edt_instances my_rtl_tessent_int_inst

add_scan_mode ext_edt_mode -edt_instances my_rtl_tessent_ext_inst

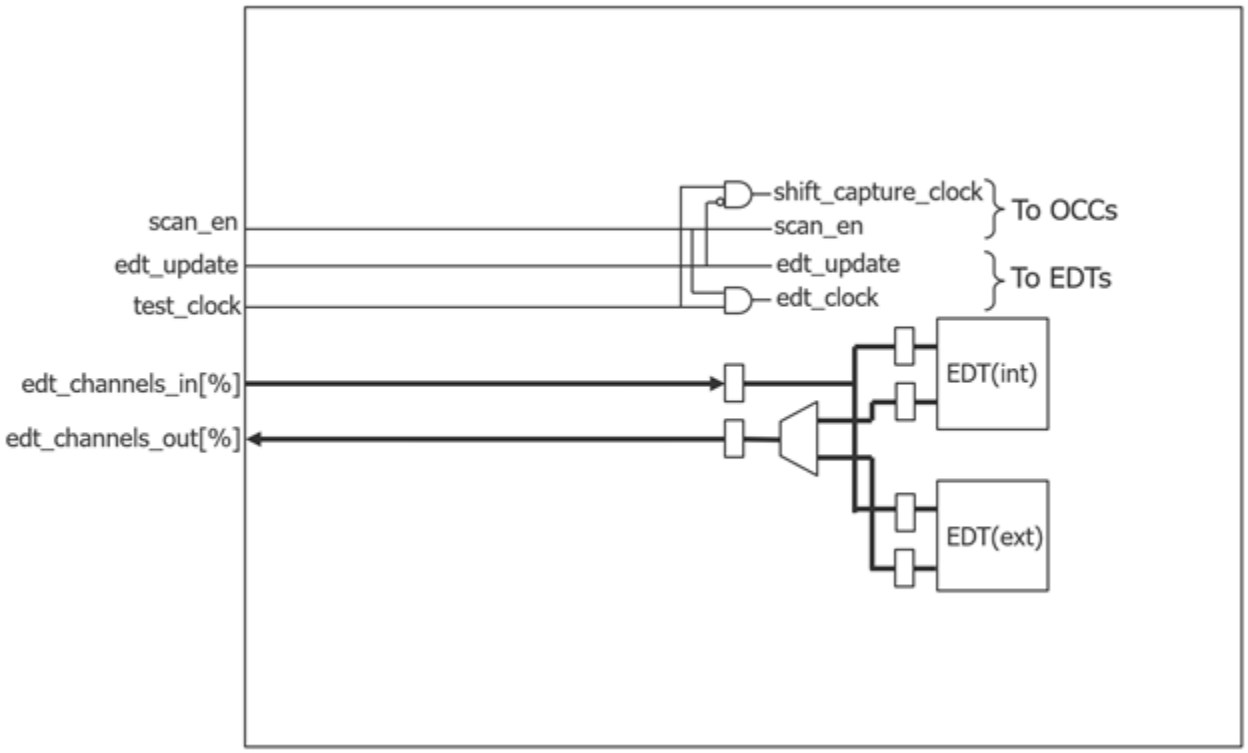
**add_scan_mode int_multi -ssh_instance my_rtl_tessent_ssn_scan_host_1_inst \
-scan_groups edt**

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

This example shows how to insert a ScanHost node while maintaining the legacy direct access to the EDT channels so that the core can be used in a chip using either the legacy scan channel muxing scheme or the SSN scheme. Figure 13-122 shows a core with two EDTs. One EDT is used for the internal scan modes and a smaller one is used for the external scan modes. The channels are connected to ports of the core and are multiplexed on to I/Os at the top level using the add_dft_modal_connections command. The edt_channel_out multiplexer is automatically inserted when the EDT/Controller/Connections/EdtChannelsOut/pin_port_name property points to the same output port name in both EDT controller wrapper.

Figure 13-122. EDT Controllers Without SSN



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

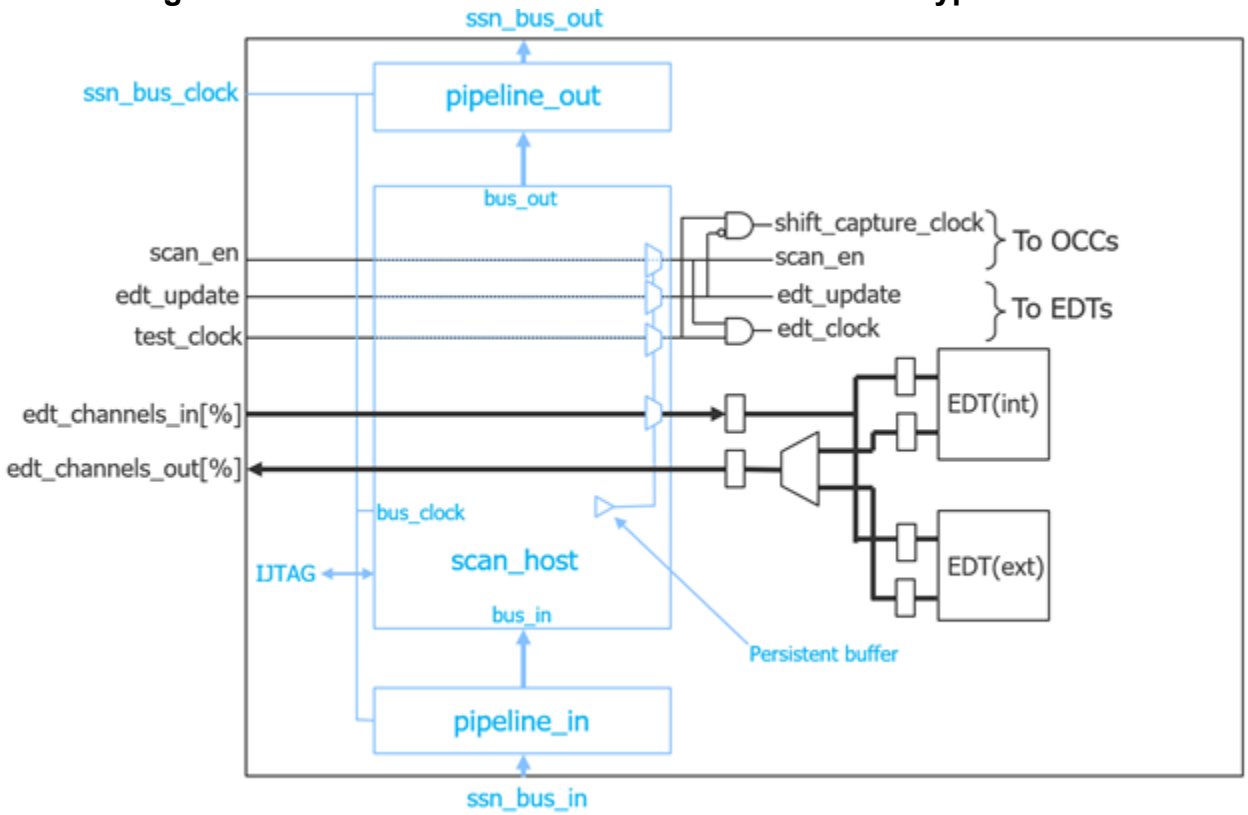
To overlay the ScanHost node on to the legacy direct access mode, simply specify the scan_signals_bypass property to on inside the ScanHost wrapper as shown below:

```
set spec [create_dft_specification -sri_sib_list {edt occ ssn}
read_config_data -in $spec -from_string {
  EDT {
    //Same as in example 1
  }
  OCC {
    ...
  }
  SSN {
    ijtag_host_interface : Sib(ssn);
    Datapath(1) {
      output_bus_width : 16;
      Pipeline(out) {
      }
      ScanHost(1) {
        scan_signals_bypass : on;
        ChainGroup(edt) {
        }
        ChainGroup(occ) {
          input_chain_count : 1;
          output_chain_count : 1;
        }
      }
Pipeline(in) {
      }
    }
  }
}
```

The preceding specification creates a ScanHost node with the bypass logic in it, as shown in Figure 13-123. In patterns -scan context, if you want to create patterns using the legacy access, simply issue the "set_ssn_options off" command as the SSN mode defaults to on when the EDT controllers connect to a ScanHost node.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-123. EDT Controllers With ScanHost and Bypass Mode



At the top level, if you want to create a chip that implements both the legacy and the SSN modes, you share the same I/Os for the legacy channels and the SSN bus bits. To achieve this, equip your I/Os with auxiliary input and output logic using the AuxiliaryInputOutputPorts wrapper.

When you insert the ScanHost SSN node, specify the same top-level input or inout port for the Datapath/Connections/bus_data_in and ScanHost/Connections/to_scan_in_bypass_in properties (shown in green in the following example dofile). The tool detects that these contain auxiliary input logic and automatically inserts the blue OR gate on the AuxiliaryInputEn pin shown in Figure 13-124.

Specify the same top-level output or inout port for the Datapath/Connections/bus_data_out and ScanHost/Connections/from_scan_out_bypass_out properties (shown in magenta in the following example dofile). The tool detects that these contain auxiliary output logic and automatically inserts the blue OR gate on the AuxiliaryOutputEn pin and the multiplexer on the AuxiliaryOutputData pin shown in Figure 13-124.

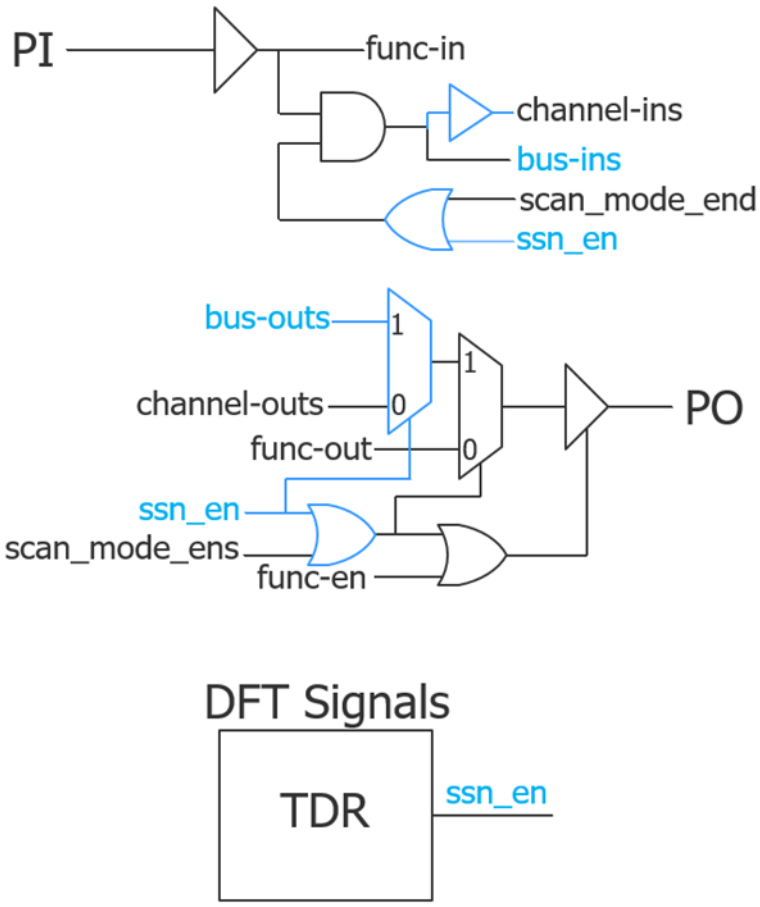
Specify the same top-level input or inout port for the Datapath/Connections/bus_clock_in that you used in the "add_dft_signals test_clock -source_node" command (shown in orange in the following example dofile).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

When you use modal connections, as specified by the add_dft_modal_connections command, the tool makes proper connections that correspond to the modal connection configuration. If you use auxiliary logic without modal connections, the tool detects auxiliary pins and connects them to the AuxEn ssn_en signal for bus_clock_in, bus_data_in, and bus_data_out pins (all related to SSN), and to ltest_en for to_scan_in_bypass_in and to_scan_out_bypass_out. The ltest_en signal is used for the bypass pins in cases where SSN is bypassed instead of active; turn off SSN with the command "set_ssn_options off". If you use the DFT control point, then the tool uses the assigned DFT control point enable signal instead of ltest_en.

Figure 13-124. Extra Logic Inserted to Share I/Os Between SSH and Legacy Mode



The following example contains sections of dofiles that implement the SSN and legacy mode at the top level together.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
add_dft_signal test_clock -source_node GPIO[4]
…
read_config_data -in $spec -from_string {
  SSN {
    ijtag_host_interface : Sib(ssn);
    DataPath(1) {
      output_bus_width :4;
      Connections {
        bus_clock_in : gpio[4];
        bus_data_in  : gpio[3:0];
        bus_data_out : gpio[8:5];
      }
      ScanHost(1) {
        scan_signals_bypass : on;
        ChainGroup(edt) {
          output_chain_count : same_as_input_chain_count;
        }
        Connections {
          ChainGroup(edt) {
            to_scan_in_bypass_in     : gpio[1:0];
            from_scan_out_bypass_out : gpio[6:5];
          }
        }
      }
    }
  }
}
```

Example 3

This example shows how to insert several ScanHost nodes at the top level to drive legacy cores not equipped with SSN. The example is built on Example 2 where the ScanHost is inserted while maintaining the legacy direct access to the EDT channels. On the first insertion pass where the auxiliary muxing logic is inserted, buffers must also be inserted to create a meeting point between the add_dft_modal_connections commands and the bypass connections for the ScanHost inserted for the child blocks not already equipped with SSN. To insert those buffers, use a similar loop at the bottom of the process_dft_specification.post_insertion proc to insert a set of buffers for the EDT channel input and output ports of each child legacy block instance.

```
set buf_cell [get_dft_cell buffer]
foreach {core_inst in_count out_count} {corea_i1 2 2 corea_i2 2 2} {
  foreach direction {in out} {
    for {set i 0} {$i < [set ${direction}_count]} {incr i} {
      create_instance ${core_inst}_edt_channel_${direction}_buf${i} \
                      -of_module $buf_cell
    }
  }
}
```

In the second insertion pass where the add_dft_modal_connections commands are used, specify the path to the input and output pin of the buffers inserted in the step above instead of to the ports of the child block.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
add_dft_modal_connections -ports gpio[2]   \
          -input_data_destination_nodes corea_i1_edt_channel_in_buf0/A \
          -enable_dft_signal $mode
add_dft_modal_connections -ports gpio[6]   \
          -output_data_source_nodes corea_i1_edt_channel_out_buf0/Y \
          -enable_dft_signal $mod
```

The ScanHost nodes make the connections between the inserted buffers and the ports of the legacy blocks using their Connections wrappers, as shown below.

```
SSN {
  ijtag_host_interface : Sib(ssn);
  scan_host_sourcing_dynamic_dft_signals : top;
  Datapath(1) {
    output_bus_width : 4;
    Pipeline(1) {
    }
    ScanHost(top) {
      scan_signals_bypass : on;
    }
    ScanHost(corea_i1) {
      scan_signals_bypass : on;
      Connections {
        test_clock                  : corea_i1/test_clock;
        edt_update                  : corea_i1/edt_update;
        scan_en                     : corea_i1/scan_en;
        to_scan_in                  : corea_i1/edt_channel_in[1:0];
        to_scan_in_bypass_in        : corea_i1_edt_channel_in_buf1/Y,
                                      corea_i1_edt_channel_in_buf0/Y;
        from_scan_out               : corea_i1/edt_channel_out[1:0];
        from_scan_out_bypass_out    : corea_i1_edt_channel_out_buf1/A,
                                      corea_i1_edt_channel_out_buf0/A;
      }
    }
    ScanHost(corea_i2) {
      scan_signals_bypass : on;
      Connections {
        test_clock                  : corea_i2/test_clock;
        edt_update                  : corea_i2/edt_update;
        scan_en                     : corea_i2/scan_en;
        to_scan_in                  : corea_i2/edt_channel_in[1:0];
        to_scan_in_bypass_in        : corea_i2_edt_channel_in_buf1/Y,
                                      corea_i2_edt_channel_in_buf0/Y;
        from_scan_out               : corea_i2/edt_channel_out[1:0];
        from_scan_out_bypass_out    : corea_i2_edt_channel_out_buf1/A,
                                      corea_i2_edt_channel_out_buf0/A;
      }
    }
    Pipeline(2) {
    }
  }
}
```

Example 4

The following example shows how to use a ScanHost that generates only the test_clock and uses DFT signal clock gaters localized in each parent instance of the OCC and EDT controllers.

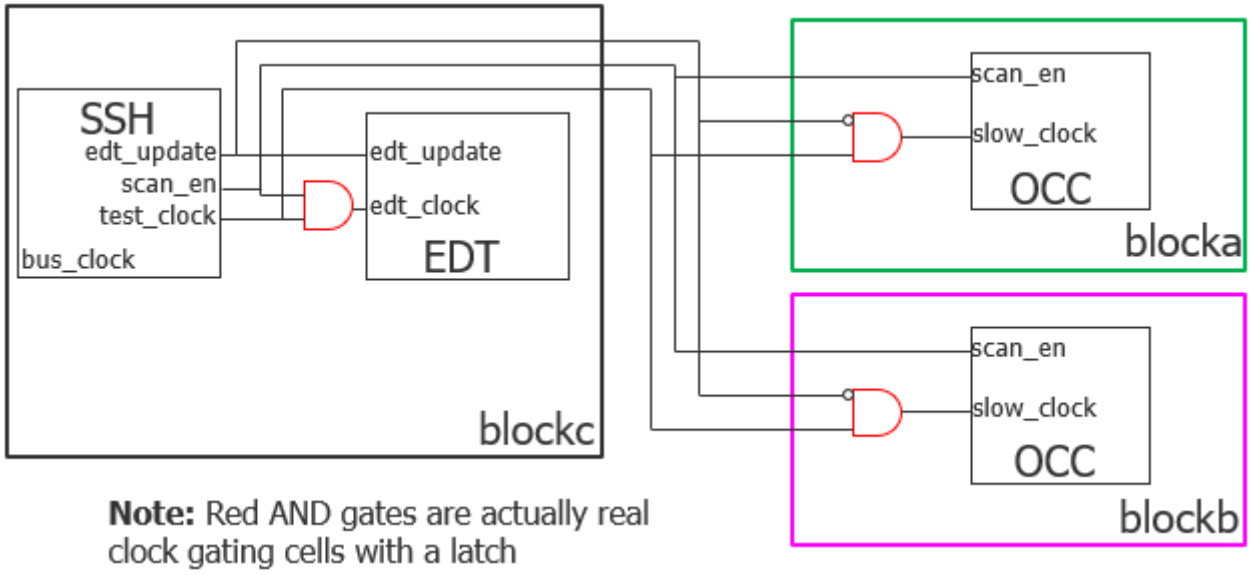
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This example uses "set_dft_specification_requirements -add_test_clock_on_ssh on" to request the presence of the test_clock in the ScanHost node. When the test_clock port is present, the shift_capture_clock and edt_clock port are no longer generated unless specified, and DFT signal clock gaters are inferred or used instead when explicitly added. In this example, the EDT and SSH are inserted into the same parent instance. An edt_clock DFT signal clock gater is automatically inferred in that parent instance to service the EDT controller. The OCCs are each inserted into distinct parent instances, and the example includes local shift_capture_clock DFT signals in each of them by using the "add_dft_signals -parent_instance" switch. If the EDT had been inserted into a different parent instance, you could have also added a local edt_clock DFT signal, and the DftSignal(edt_clock) connection in the EDT Connection wrapper would have mapped to the local DFT signal clock gater. Figure 13-125 illustrates the resulting circuitry.

```
# Request test_clock on ScanHost
set_dft_specification_requirements -add_test_clock_on_ssh on

# Request localized DFT clock gating cell for each OCCs
add_dft_signal shift_capture_clock -parent_instance blocka
add_dft_signal shift_capture_clock -parent_instance blockb

check_design_rules

set spec [create_dft_spec -sri_sib_list {ssn edt occ}]
read_config_data -in $spec -from_string {
  SSN {
   …
  }
  EDT {
   …
  }
  OCC {
    Controller(clka) {
      parent_instance : blocka;
      …
    }
    Controller(clkb) {
      parent_instance : blockb;
      …
    }
  }
}

process_dft_specification
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-125. ScanHost Sourcing test_clock With Distributed DFT Signal Clock Gaters



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Connections

A wrapper used to specify custom connections for the ScanHost node.

Usage

```
DftSpecification(module_name,id) {
  SSN {
    Datapath(id) {
      ScanHost(id) {
        Connections {
          bus_clock_in                  : port_pin_name ;
          test_clock                    : port_pin_name ;
          test_clock_bypass_in          : port_pin_constant_name ;
                                       // default: DftSignal(test_clock)
          edt_clock                     : port_pin_name, ... ;
          edt_clock_bypass_in           : port_pin_constant_name ;
                                         // default: DftSignal(edt_clock)
          edt_update                    : port_pin_name, ... ;
          edt_update_bypass_in          : port_pin_constant_name ;
                                       // default: DftSignal(edt_update)
          scan_en                       : port_pin_name, ... ;
          scan_en_bypass_in             : port_pin_constant_name ;
                                         // default: DftSignal(scan_en)
          shift_capture_clock           : port_pin_name, ... ;
          shift_capture_clock_bypass_in : port_pin_constant_name ;
                              // default: DftSignal(shift_capture_clock)
          shift_clock                   : port_pin_name, ...     ;
          shift_clock_bypass_in         : port_pin_constant_name ;
          capture_clock                 : port_pin_name, ...     ;
          capture_clock_bypass_in       : port_pin_constant_name ;
          ssh_is_active                 : port_pin_name, ... ;
          to_scan_in                    : port_pin_name, ... ;
          to_scan_in_bypass_in          : port_pin_name, ... ;
                                         // default: edt_channels_in[%d]
          from_scan_out                 : port_pin_name, ... ;
          from_scan_out_bypass_out      : port_pin_name, ... ;
                                         // default: edt_channels_out[%d]
          delete_pre_existing_functional_source : on | off ;
          ChainGroup(id) {
            to_scan_in               : port_pin_name, ... ;
            to_scan_in_bypass_in     : port_pin_name, ... ;
                                // default: edt_channels_in_%s[%d]
            from_scan_out            : port_pin_name, ... ;
            from_scan_out_bypass_out : port_pin_name, ... ;
                                // default: edt_channels_out_%s[%d]
          }
        }
      }
    }
  }
}
```

Description

This wrapper is rarely used because the connection between the ScanHost ports and the EDT/OCC controllers is automated when performed together. Also, the DFT signals scan_en,

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

edt_update, edt_clock, and shift_capture_clock are automatically inferred on the output port of the ScanHost node. You can use add_dft_control_points *dft_signal_name* to connect them where you need them to normally go.

Note

The Connections/ChainGroup wrapper is mandatory when process_dft_specification is called without the -no_insertion switch and the scan_signals_bypass property is on.

Arguments

- bus_clock_in : *port_pin_name*;

  A property that can be used to specify the source of the bus clock for the ScanHost node. This connection is typically inferred automatically. If unspecified and a DesignInstance wrapper is found below the ScanHost wrapper and this instance was created with an output bus clock port, the bus clock input of the ScanHost node will connect to it unless the ignore_bus_clock_out property is set to off in the DesignInstance/Connections wrapper. If a node below the ScanHost wrapper has its Connection/bus_clock_in property specified, it will be used by the ScanHost node also. Otherwise, it will connect to the object specified by the Datapath/Connections/bus_clock_in property, which defaults to “ssn_bus_clock”. The only time you will specify this property is when you have a clock tree synthesis boundary inside the core, and you are transferring bus data from one domain to the other using a BusFrequencyDivider/BusFrequencyMultiplier node pair. See the description of the BusFrequencyDivider wrapper for a complete description of those nodes and their purpose.

- test_clock : *port_pin_name*;

  A property that specifies a port or pin object to make explicit connections from the test_clock port. It is equivalent to using “add_dft_control_points *port_pin_name* -dft_signal test_clock -type dynamic_control_point”. You will use this property only if you inserted EDT controllers inside a set of blocks, and you now want to insert a ScanHost node in a parent module to service them. The property allows specifying several destination pin or port objects, and a broadcast connection will be made to all of them.

- test_clock_bypass_in : *port_pin_constant_name*;

  A property that defines the connection source for the test_clock signal when the ScanHost is not active. This property is only used when the ScanHost/scan_signals_bypass property is on. The <*port_pin_constant_name*> can be either a port or pin object, or the constant 0, which ties the port to 0. The default is DftSignal(test_clock), which is typically the correct value.

- edt_clock : *port_pin_name*, ... ;

  A property that specifies a port or pin object to make explicit connections from the edt_clock port. It is equivalent to using “add_dft_control_points *port_pin_name* -dft_signal edt_clock -type dynamic_control_point”. You use this property only if you inserted EDT controllers inside a set of blocks, and you now want to insert a ScanHost node in a parent module to service them. The property allows specifying several destination pin or port objects, and a broadcast connection will be made to all of them.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- edt_clock_bypass_in : *port_pin_constant_name*, ...;

  A property that defines the connection source for the edt_clock signal when the ScanHost is not active. This property is only used when the ScanHost/scan_signals_bypass property is on. The <*port_pin_constant_name*> can be either a port or pin object, or the constant 0, which ties the port to 0.

  The default is DftSignal(edt_clock), which is typically the correct value.

- edt_update : *port_pin_name*, ...;

  A property that specifies a port or pin object to make explicit connections from the edt_update port. It is equivalent to using "add_dft_control_points *port_pin_name* -dft_signal edt_update -type dynamic_control_point". You will use this property only if you inserted EDT controllers inside a set of blocks, and you now want to insert a ScanHost node in a parent module to service them. The property allows specifying several destination pin or port objects, and a broadcast connection will be made to all of them.

- edt_update_bypass_in : *port_pin_constant_name* ;

  A property that defines the connection source for the edt_update signal when the ScanHost is not active. This property is only used when the ScanHost/scan_signals_bypass property is on. The <*port_pin_constant_name*> can be either a port or pin object, or the constant 0, which ties the port to 0.

  The default is DftSignal(edt_update), which is typically the correct value.

- scan_en : *port_pin_name*, ... ;

  A property that specifies a port or pin object to make explicit connections from the scan_en port. It is equivalent to using "add_dft_control_points *port_pin_name* -dft_signal scan_en -type dynamic_control_point". You will use this property only if you inserted EDT controllers inside a set of blocks, and you now want to insert a ScanHost node in a parent module to service them. The property allows specifying several destination pin or port objects, and a broadcast connection will be made to all of them. If you are using the OCC wrapper to insert OCC controllers at the same time as the ScanHost node, they will automatically connect their scan_en port to the ScanHost node. Use the add_dft_control_points command as shown above to connect the scan_en to third-party OCCs.

- scan_en_bypass_in : *port_pin_constant_name* ;

  A property that defines the connection source for the scan_en signal when the ScanHost is not active. This property is only used when the ScanHost/scan_signals_bypass property is on. The <*port_pin_constant_name*> can be either a port or pin object, or the constant 0, which ties the port to 0.

  The default is DftSignal(scan_en), which is typically the correct value.

- shift_capture_clock : *port_pin_name*, ... ;

  A property that specifies a port or pin object to make explicit connections from the shift_capture_clock port. It is equivalent to using "add_dft_control_points *port_pin_name* -dft_signal shift_capture_clock -type dynamic_control_point". You will use this property

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

only if you inserted EDT controllers inside a set of blocks, and you now want to insert an ScanHost node in a parent module to service them. The property allows specifying several destination pin or port objects, and a broadcast connection will be made to all of them. If you are using the OCC wrapper to insert OCC controllers at the same time as the ScanHost node, they will automatically connect their slow_clock port to the ScanHost node. Use the add_dft_control_points command as shown above to connect the shift_capture_clock to third-party OCCs.

- shift_capture_clock_bypass_in : *port_pin_name* ;

  A property that defines the connection source for the shift_capture_clock signal when the ScanHost is not active. This property is only used when the ScanHost/scan_signals_bypass property is on. The <*port_pin_constant_name*> can be either a port or pin object, or the constant 0, which ties the port to 0.

  The default is DftSignal(shift_capture_clock), which is typically the correct value.

- shift_clock : *port_pin_name*, … ;

  A property that specifies a port or pin object to make explicit connections from the shift_clock port. It is equivalent to using "add_dft_control_points *port_pin_name* -dft_signal shift_clock -type dynamic_control_point".

- shift_clock_bypass_in : *port_pin_constant_name* ;

  A property that defines the connection source for the shift_clock signal when the ScanHost is not active. This property is only used when the ScanHost/scan_signals_bypass property is on. The <*port_pin_constant_name*> can be either a port or pin object, or the constant 0, which ties the port to 0.

- capture_clock : *port_pin_name*, … ;

  A property that specifies a port or pin object to make explicit connections from the shift_clock port. It is equivalent to using "add_dft_control_points *port_pin_name* -dft_signal capture_clock -type dynamic_control_point".

- capture_clock_bypass_in : *port_pin_constant_name* ;

  A property that defines the connection source for the capture_clock signal when the ScanHost is not active. This property is only used when the ScanHost/scan_signals_bypass property is on. The <*port_pin_constant_name*> can be either a port or pin object, or the constant 0, which ties the port to 0.

- ssh_is_active : *port_pin_name*, … ;

  A property that specifies a port or pin object to make explicit connections from the ssh_is_active port.

- to_scan_in : *port_pin_name*, ... ;

  A property that specifies a port or pin object to make explicit connections from the scan_in ports. You specify the property inside the ChainGroup wrapper if the ChainGroup wrapper was used inside the ScanHost wrapper otherwise you use the property directly found inside the Connections wrapper. You will use this property only if you inserted EDT controllers

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

inside a set of blocks and you now want to insert a ScanHost node in a parent module to service them. In such a case, you will use one ChainGroup per EDT controllers and connect each group scan_in and scan_out ports to each EDT controllers. When using the ChainGroup(ID) wrapper, the ID must match the ID of a ChainGroup wrapper above the Connections wrapper. The ID is case-sensitive.

- to_scan_in_bypass_in : *port_pin_name*, ... ;

  A property that defines the connection source for the to_scan_in signals when the ScanHost is not active. This property is only used when the ScanHost/scan_signals_bypass property is on.

  The default is edt_channels_in_%s[%d], where %s represents the ID of the parent ChainGroup wrapper.

  If you specify a top-level port equipped with a pad or a boundary scan cell with AuxiliaryInputData logic, the command automatically maps the connection to the correct place. If you specify this property to point to the same location as the bus_data_in property, the AuxiliaryInputEn pin is automatically ORed with the ssn_en DFT signal as shown in Figure 13-124 on page 4528 in the ScanHost reference page.

  When the design level is chip, the port must already exist and be connected to a pad buffer cell. You may also have a boundary scan cell equipped with AuxiliaryInputData logic. When you point to such a top-level port, the tool automatically maps the connection across the pad and optional boundary scan cell.

- from_scan_out : *port_pin_name*, ... ;

  A property that specifies a port or pin object to make explicit connections to the scan_out ports. You specify the property inside the ChainGroup wrapper if the ChainGroup wrapper was used inside the ScanHost wrapper otherwise you use the property directly found inside the Connections wrapper. You will use this property only if you inserted EDT controllers inside a set of blocks, and you now want to insert a ScanHost node in a parent module to service them. In such a case, you will use one ChainGroup per EDT controllers and connect each group scan_in and scan_out ports to each EDT controllers. When using the ChainGroup(ID) wrapper, the ID must match the ID of a ChainGroup wrapper above the Connections wrapper. The ID is case-sensitive.

  If you specify a top-level port equipped with a pad or a boundary scan cell with AuxiliaryOutputData logic, the command automatically maps the connection to the correct place. If you specify this property to point to the same location as the bus_data_out property, the AuxiliaryOutputEn pin is automatically ORed and the AuxiliaryOutputData pin is automatically MUXed with the ssn_en DFT signal as shown in Figure 13-124 on page 4528 in the ScanHost reference page.

- from_scan_out_bypass_out : *port_pin_name*, ... ;

  A property that defines the connection destination for the from_scan_out signals when the ScanHost is not active. This property is only used when the ScanHost/scan_signals_bypass property is on.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The default is edt_channels_out_%s[%d], where %s represents the ID of the parent ChainGroup wrapper.

When the design level is chip, the port must already exist and be connected to a pad buffer cell. You may also have a boundary scan cell equipped with AuxiliaryOutputData logic. When you point to such a top-level port, the tool automatically maps the connection across the pad and optional boundary scan cell.

- delete_pre_existing_functional_source : on | off ;

  A property that enables you to delete a preexisting functional source and make a connection to SSN logic. This value is inherited from DataPath/Connections/ delete_pre_existing_functional_source, which defaults to "off."

Examples

Example 1

The following example shows how to use the ScanHost/Connections wrapper to explicitly connect the scan ports of a set of child blocks containing EDT controllers. Notice how the "read_config_data -from_string" command uses "" instead of {} as the delimiter of the string value. This allows the Tcl variable block1, block2, block3 to be set from design introspection and substituted in the configuration data before the data is read in. You could use similar introspection to extract the scan_in and scan_out count in each block.

Alternatively, you could use the add_config_element and set_config_value commands to specify the connections to an arbitrary amount of sub-block.

```
set spec [create_dft_specification -sri_sib_list {ssn}
set block1 [get_single_name [get_instances -of_modules block1]]
set block2 [get_single_name [get_instances -of_modules block2]]
set block3 [get_single_name [get_instances -of_modules block3]]
read_config_data -in $spec -from_string "
  SSN {
    Datapath(11) {
      ijtag_host_interface  : Sib(ssn);
      output_bus_width : 32;
      ScanHost(1) {
        ChainGroup(1) {
          input_chain_count  : 16;
          output_chain_count : 8;
        }
        ChainGroup(2) {
          input_chain_count  : 14;
          output_chain_count : 10;
        }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
          ChainGroup(3) {
            input_chain_count  : 16;
            output_chain_count : 4;
          }
          Connections {
            scan_en             : ${block1}/scan_en,
                                  ${block2}/scan_en,
                                  ${block3}/scan_en;
            edt_update          : ${block1}/edt_update,
                                  ${block2}/edt_update,
                                  ${block3}/edt_update;
            edt_clock           : ${block1}/edt_clock,
                                  ${block2}/edt_clock,
                                  ${block3}/edt_clock;
            shift_capture_clock : ${block1}/shift_capture_clock,
                                  ${block2}/shift_capture_clock,
                                  ${block3}/shift_capture_clock;
            ChainGroup(1) {
              to_scan_in        : ${block1}/scan_in[%d];
              from_scan_out     : ${block1}/scan_out[%d];
            }
            ChainGroup(1) {
              to_scan_in        : ${block2}/scan_in[%d];
              from_scan_out     : ${block2}/scan_out[%d];
            }
            ChainGroup(1) {
              to_scan_in        : ${block3}/scan_in[%d];
              from_scan_out     : ${block3}/scan_out[%d];
            }
          }
        }
      }
    }

  process_dft_specification
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

RtlCells

A wrapper that is used to specify the name of the ports to use when creating RTL cells.

Usage

```
DftSpecification(module_name,id) {
  RtlCells {
    And2|Or2|ClkAnd2|ClkOr2 {
      input0                    : string ;  // default: a
      input1                    : string ;  // default: b
      output                    : string ;  // default: y
    }
    Buf|Inv|ClkBuf|ClkInv {
      input                     : string ;  // default: a
      output                    : string ;  // default: y
    }
    Mux2|ClkMux2 {
      input0                    : string ;  // default: a
      input1                    : string ;  // default: b
      select                    : string ;  // default: s
      output                    : string ;  // default: y
    }

    ClkGateAnd|ClkGateOr {
      clock_in                  : string ;  // default: clk
      functional_enable         : string ;  // default: fe
      test_enable               : string ;  // default: te
      clock_out                 : string ;  // default: clkg
    }
    ClkShaperResetActive|ClkShaperResetInactive {
      clock_in                  : string ;  // default: clk_in
      reset                     : string ;  // default: rst
      reset_polarity            : active_high | active_low;
      enable0                   : string ;  // default: en0
      enable1                   : string ;  // default: en1
      clock_out                 : string ;  // default: clk_out
    }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
PosedgeClockDff|NegedgeClockDff {
  data_in                    : string ;  // default: d
  clock_in                   : string ;  // default: clk
  data_out                   : string ;  // default: q
}
PosedgeClockDffReset|NegedgeClockDffReset {
  reset                      : string ;  // default: rn
  reset_polarity             : string ;  // active_high | active_low
  data_in                    : string ;  // default: d
  clock_in                   : string ;  // default: clk
  data_out                   : string ;  // default: q
}
PosedgeClockDffSet|NegedgeClockDffSet {
  set                        : string ;  // default: sn
  set_polarity               : string ;  // active_high | active_low
  data_in                    : string ;  // default: d
  clock_in                   : string ;  // default: clk
  data_out                   : string ;  // default: q
}

PosedgeDff|NegedgeDff {
  data_in                    : string ;  // default: d
  clock_in                   : string ;  // default: clk
  data_out                   : string ;  // default: q
}
PosedgeDffReset|NegedgeDffReset {
  reset                      : string ;  // default: rn
  reset_polarity             : string ;  // active_high | active_low
  data_in                    : string ;  // default: d
  clock_in                   : string ;  // default: clk
  data_out                   : string ;  // default: q
}
PosedgeDffSet|NegedgeDffSet {
  set                        : string ;  // default: sn
  set_polarity               : string ;  // active_high | active_low
  data_in                    : string ;  // default: d
  clock_in                   : string ;  // default: clk
  data_out                   : string ;  // default: q
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
        ActiveHighLatch|ActiveLowLatch {
          data_in                    : string ; // default: d
          clock_in                   : string ; // default: clk
          data_out                   : string ; // default: q
        }
        ActiveHighLatchReset|ActiveLowLatchReset {
          reset                      : string ; // default: rn
          reset_polarity             : string ; // active_high | active_low
          data_in                    : string ; // default: d
          clock_in                   : string ; // default: clk
          data_out                   : string ; // default: q
        }
        ActiveHighLatchSet|ActiveLowLatchSet {
          set                        : string ; // default: sn
          set_polarity               : string ; // active_high | active_low
          data_in                    : string ; // default: d
          clock_in                   : string ; // default: clk
          data_out                   : string ; // default: q
        }

        PosedgeSynchronizer {
          data_in                    : string ; // default: d
          clock_in                   : string ; // default: clk
          data_out                   : string ; // default: q
        }
        PosedgeSynchronizerReset {
          reset                      : string ; // default: rn
          reset_polarity             : string ; // active_high | active_low
          data_in                    : string ; // default: d
          clock_in                   : string ; // default: clk
          data_out                   : string ; // default: q
        }
      }
    }
```

Description

A wrapper that is used to specify the name of the ports to use when creating RTL cells. A set of RTL cells is created when the use_rtl_cells property of the DftSpecification wrapper defaults to on or is specified as on. The list of cells is dependent on the need of the instruments being created, and the list requested in the force_creation_of_rtl_cells : *cell_type, ... ;* command.

Clock Shaper Cells

The clock shaper cell is a special clock gater cell implemented with a multiplexer and two latches, enabling the clock to be gated high and low. You can use the cell to let the free running clock pass through, freeze the clock low or high, or create a divided frequency while maintaining a single clock-in to clock-out timing path. Figure 13-126 shows the clock shaper cell, in red, used in an application to create a divided clock and a gated clock at different times based on the values of en0 and en1. Refer to "ScanHost" on page 4484 to see how to use it in that manner within the ScanHost node.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

There are two variants of the clock shaper cell: ClkShaperResetActive and ClkShaperResetInactive. These variants differ in how they behave in inactive mode, as following description explains.

The clock shaper consists of a multiplexer where its select is driven by the clock input and the output of the multiplexer drives the clock output. When m0 is 0 and m1 is 1, the multiplexer selects a 1 when the clock is high and selects a 0 when the clock is low. In that case, the clock output follows the clock input. When both m0 and m1 are low, the clock output is gated low, and when both m0 and m1 are high, the clock is gated high. The active high latch in front of the m0 input ensures that the m0 value changes only when the clock selects the m1 value. The active low latch in front of the m1 input makes sure that the m1 value changes only when the clock selects the m0 value. The latch in front of the m0 pin has an asynchronous reset. For ClkShaperResetActive, the latch in front of the m1 pin has an asynchronous set. This ensures that, after a reset, the clock gater initializes to let the clock go through regardless of whether the clock input starts high or low. For ClkShaperResetInactive, the latch in front of the m1 pin has an asynchronous reset, which enables you to suppress the clock in inactive mode.

Note

If a clock_mux cell exists in memory, the clock shaper RTL cell is created with a clock_mux instance to implement the multiplexer logic. This enables you to synthesize the clock shaper RTL cell if you do not have a real cell for it.

In the application shown later in this topic, which mirrors how the ScanHost node uses the cell, the clock shaper cell is used to sometimes gate the clock low so normal full-frequency pulses can propagate through. It is also used at a different time to synthesize a divided clock. These two applications are shown in the timing diagram in Figure 13-127. When clock_gate_en goes from 0 to 1, the 0 is held by the active-low latch, which prevents the 1 from reaching the m1 input while the clock is high. Once the multiplexer selects the m0 input, the 1 value traverses the active low latches and is ready for when the clock goes back high. This time, because m1 is high, the clock output goes high and follows the clock input.

When you use the clock shaper to create a divided clock, you must hold the clock high for one or more full clock cycle. Do this by making m1 and m0 both high. Again, the active low latch in front of the m1 input makes sure it goes high only when the multiplexer selects the m0 input. The active low latch in series with the active high latch forms a flip-flop and ensures m0 goes high when the clock selects the m1 input. With this logic, both enables, clock_en and next_clock_div, have a full clock period to reach the clock shaper cell. The rising and falling edge of the clk_out, whether it is a gated or divided version of the clk_in, follows the timing of the clk_in to clk_out through the select of the mux. The arrival times of both m0 and m1 never impact the edge placement of the clock output.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-126. Clock Shaper Cell Application Schematic

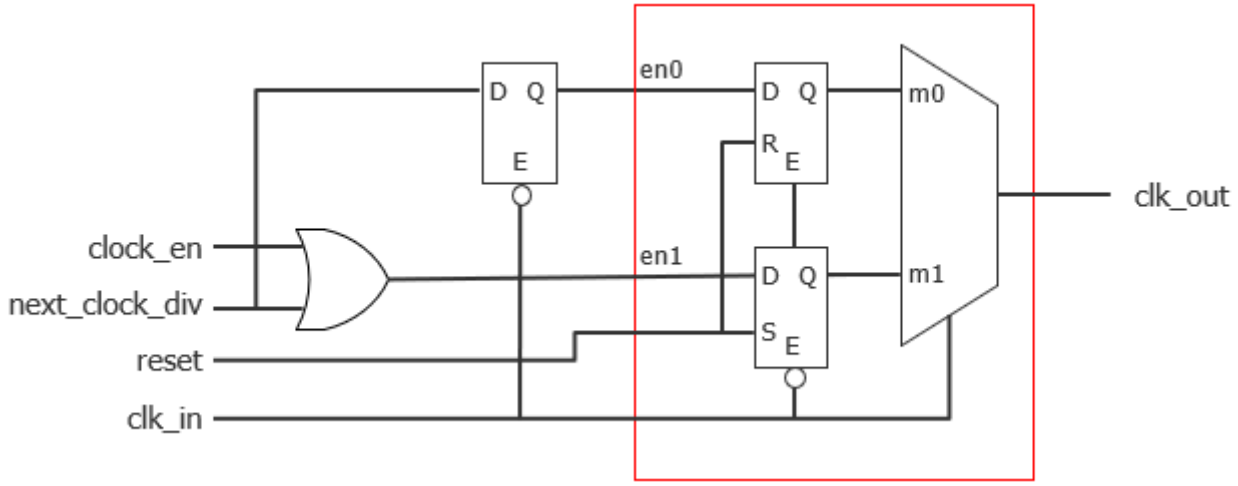
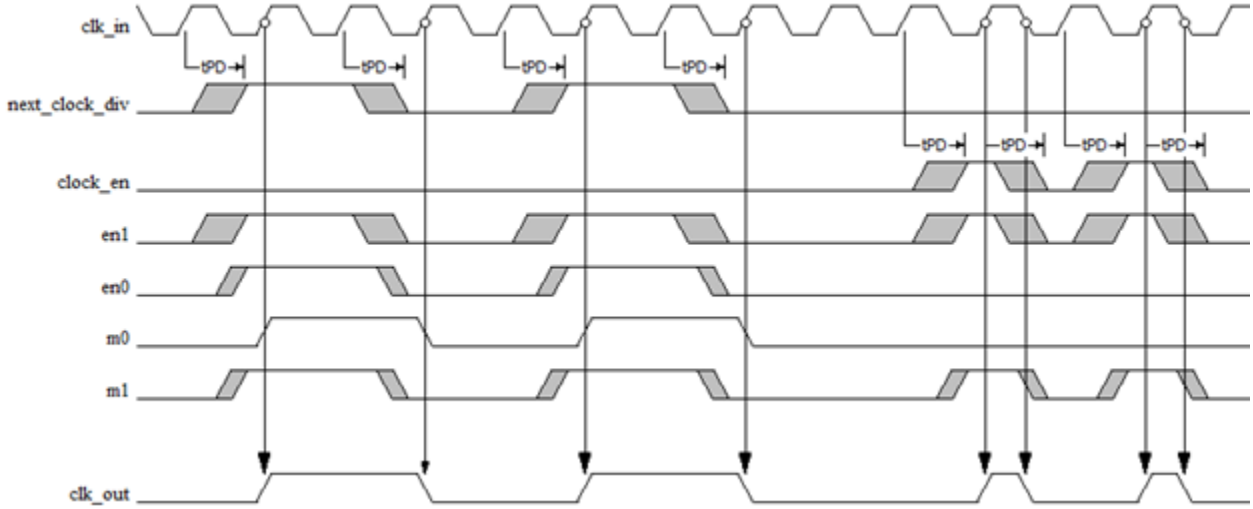


Figure 13-127. Clock Shaper Cell Application Timing



Arguments

- And2|Or2|ClkAnd2|ClkOr2/input0 : *string* ;

  A property that specifies the name of the first input of the And or Or cell. For the Clock version, this input is the input to which the clock signal is connected. The default string is “a”.

- And2|Or2|ClkAnd2|ClkOr2/ input1 : *string* ;

  A property that specifies the name of the second input of the And or Or cells. For the Clock version, this input is the input to which the data signal is connected. The default string is “b”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- And2|Or2|ClkAnd2|ClkOr2/ output : *string* ;

  A property that specifies the name of the output of the And or Or cells. The default string is “y”.

- Buf|Inv|ClkBuf|ClkInv/input : *string* ;

  A property that specifies the name of the input of the Buffer or Inverter cells. The default string is “a”.

- Buf|Inv|ClkBuf|ClkInv/output : *string* ;

  A property that specifies the name of the output of the Buffer or Inverter cells. The default string is “y”.

- Mux2|ClkMux2/input0 : *string* ;

  A property that specifies the name of input0 of the multiplexer cells. The default string is “a”.

- Mux2|ClkMux2/input1 : *string* ;

  A property that specifies the name of input1 of the multiplexer cells. The default string is “b”.

- Mux2|ClkMux2/select : *string* ;

  A property that specifies the name of the select input of the multiplexer cells. The default string is “s”.

- Mux2|ClkMux2/output : *string* ;

  A property that specifies the name of the output of the multiplexer cells. The default string is “y”.

- ClkGateAnd|ClkGateOr/clock_in : *string* ;

  A property that specifies the name of the clock input of the clock gating cells. The default string is “clk”.

- ClkGateAnd|ClkGateOr/functional_enable : *string* ;

  A property that specifies the name of the functional enable input of the clock gating cells. The default string is “fe”.

- ClkGateAnd|ClkGateOr/test_enable : *string* ;

  A property that specifies the name of the test enable input of the clock gating cells. The default string is “te”. The input is used to disable the gating during scan test.

- ClkGateAnd|ClkGateOr/clock_out : *string* ;

  A property that specifies the name of the gated clock output of the clock gating cells. The default string is “clkg”.

- ClkShaperResetActive|ClkShaperResetInactive/clock_in : *string* ;

  A property that specifies the name of the clock shaper clock input pin. The default string is “clk_in”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ClkShaperResetActive|ClkShaperResetInactive/reset : *string* ;

  A property that specifies the name of the clock shaper reset input pin. The reset pin feeds the reset pin of the active high latch and the set pin of the active low latch, as shown in Figure 13-126. The default string is “reset”.

- ClkShaperResetActive|ClkShaperResetInactive/enable0 : *string* ;

  A property that specifies the name of the clock shaper enable0 input pin. The enable0 pin feeds an active high latch that then feeds the m0 input of the multiplexer, as shown in Figure 13-126. The default string is “en0”.

- ClkShaperResetActive|ClkShaperResetInactive/enable1 : *string* ;

  A property that specifies the name of the clock shaper enable1 input pin. The enable1 pin feeds an active low latch that then feeds the m1 input of the multiplexer, as shown in Figure 13-126. The default string is “en1”.

- ClkShaperResetActive|ClkShaperResetInactive/clock_out : *string* ;

  A property that specifies the name of the clock shaper clock output pin. The default string is “clk_out”.

- PosedgeClockDff|NegedgeClockDff/data_in : *string* ;

  A property that specifies the name of the data input of the clock DFF cells. The default is “d”.

- PosedgeClockDff|NegedgeClockDff/clock_in : *string* ;

  A property that specifies the name of the clock input of the clock DFF cells. The default is “clk”.

- PosedgeClockDff|NegedgeClockDff/data_out : *string* ;

  A property that specifies the name of the data output of the clock DFF cells. The default is “q”.

- PosedgeClockDffReset | NegedgeClockDffReset/reset : *string* ;

  A property that specifies the name of the reset of the clock DFF cells. The default is “rn”.

- PosedgeClockDffReset | NegedgeClockDffReset/reset_polarity : *string* ;

  A property that specifies the name of the reset polarity of the clock DFF cells. The default is “active_low”.

- PosedgeClockDffReset | NegedgeClockDffReset/data_in : *string* ;

  A property that specifies the name of the data input of the clock DFF cells. The default is “d”.

- PosedgeClockDffReset | NegedgeClockDffReset/clock_in ; *string* ;

  A property that specifies the name of the clock input of the clock DFF cells. The default is “clk”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- PosedgeClockDffReset | NegedgeClockDffReset/data_out ; *string* ;

  A property that specifies the name of the data output of the clock DFF cells. The default is “q”.

- PosedgeClockDffSet | NegedgeClockDffSet/set : *string* ;

  A property that specifies the name of the set of the clock DFF cells. The default is “sn”.

- PosedgeClockDffSet | NegedgeClockDffSet/set_polarity : *string* ;

  A property that specifies the name of the set polarity of the clock DFF cells. The default is “active_low”.

- PosedgeClockDffSet | NegedgeClockDffSet/data_in : *string* ;

  A property that specifies the name of the data input of the clock DFF cells. The default is “d”.

- PosedgeClockDffSet | NegedgeClockDffSet/clock_in : *string* ;

  A property that specifies the name of the clock input of the clock DFF cells. The default is “clk”.

- PosedgeClockDffSet | NegedgeClockDffSet/data_out : *string* ;

  A property that specifies the name of the data output of the clock DFF cells. The default is “q”.

- PosedgeDff|NegedgeDff/data_in : *string* ;

  A property that specifies the name of the data input of the DFF cells. The default is “d”.

- PosedgeDff|NegedgeDff/clock_in : *string* ;

  A property that specifies the name of the clock input of the DFF cells. The default is “clk”.

- PosedgeDff|NegedgeDff/data_out : *string* ;

  A property that specifies the name of the data output of the DFF cells. The default is “q”.

- PosedgeDffReset | NegedgeDffReset/reset : *string* ;

  A property that specifies the name of the reset of the DFF cells. The default is “rn”.

- PosedgeDffReset | NegedgeDffReset/reset_polarity : *string* ;

  A property that specifies the name of the reset polarity of the DFF cells. The default is “active_low”.

- PosedgeDffReset | NegedgeDffReset/data_in : *string* ;

  A property that specifies the name of the data input of the DFF cells. The default is “d”.

- PosedgeDffReset | NegedgeDffReset/clock_in ; *string* ;

  A property that specifies the name of the clock input of the DFF cells. The default is “clk”.

- PosedgeDffReset | NegedgeDffReset/data_out ; *string* ;

  A property that specifies the name of the data output of the DFF cells. The default is “q”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- PosedgeDffSet | NegedgeDffSet/set : *string* ;

  A property that specifies the name of the set of the DFF cells. The default is “sn”.
- PosedgeDffSet | NegedgeDffSet/set_polarity : *string* ;

  A property that specifies the name of the set polarity of the DFF cells. The default is “active_low”.
- PosedgeDffSet | NegedgeDffSet/data_in : *string* ;

  A property that specifies the name of the data input of the DFF cells. The default is “d”.
- PosedgeDffSet | NegedgeDffSet/clock_in : *string* ;

  A property that specifies the name of the clock input of the DFF cells. The default is “clk”.
- PosedgeDffSet | NegedgeDffSet/data_out : *string* ;

  A property that specifies the name of the data output of the DFF cells. The default is “q”.
- ActiveHighLatch | ActiveLowLatch/data_in : *string* ;

  A property that specifies the name of the data input of the latch. The default is “d”.
- ActiveHighLatch | ActiveLowLatch/clock_in : *string* ;

  A property that specifies the name of the clock input of the latch. The default is “clk”.
- ActiveHighLatch | ActiveLowLatch/data_out : *string* ;

  A property that specifies the name of the data output of the latch. The default is “q”.
- ActiveHighLatchReset | ActiveLowLatchReset/reset : *string* ;

  A property that specifies the name of the reset of the latch. The default is “rn”.
- ActiveHighLatchReset | ActiveLowLatchReset/reset_polarity : *string* ;

  A property that specifies the name of the reset polarity of the latch. Choose either active_high or active_low. The default is “active_low”.
- ActiveHighLatchReset | ActiveLowLatchReset/data_in : *string* ;

  A property that specifies the name of the data input of the latch. The default is “d”.
- ActiveHighLatchReset | ActiveLowLatchReset/clock_in : *string* ;

  A property that specifies the name of the clock input of the latch. The default is “clk”.
- ActiveHighLatchReset | ActiveLowLatchReset/data_out : *string* ;

  A property that specifies the name of the data output of the latch. The default is “q”.
- ActiveHighLatchSet | ActiveLowLatchSet/set : *string* ;

  A property that specifies the name of the set of the latch. The default is “sn”.
- ActiveHighLatchSet | ActiveLowLatchSet/set_polarity : *string* ;

  A property that specifies the name of the set polarity of the latch. Choose either active_high or active_low. The default is “active_low”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ActiveHighLatchSet | ActiveLowLatchSet/data_in : *string* ;

  A property that specifies the name of the data input of the latch. The default is “d”.

- ActiveHighLatchSet | ActiveLowLatchSet/clock_in : *string* ;

  A property that specifies the name of the clock input of the latch. The default is “clk”.

- ActiveHighLatchSet | ActiveLowLatchSet/data_out : *string* ;

  A property that specifies the name of the data output of the latch. The default is “q”.

- PosedgeSynchronizer/data_in : *string* ;

  A property that specifies the name of the data input of the synchronizer cells. The default is “d”.

- PosedgeSynchronizer/clock_in : *string* ;

  A property that specifies the name of the clock input of the synchronizer cells. The default is “clk”.

- PosedgeSynchronizer/data_out : *string* ;

  A property that specifies the name of the data output of the synchronizer cells. The default is “q”.

- PosedgeSynchronizerReset/reset : *string* ;

  A property that specifies the name of the reset of the synchronizer reset cells. The default is “rn”.

- PosedgeSynchronizerReset/reset_polarity : *string* ;

  A property that specifies the name of the reset polarity of the synchronizer reset cells. Choose either active_high or active_low. The default is “active_low”.

- PosedgeSynchronizerReset/data_in : *string* ;

  A property that specifies the name of the data input of the synchronizer reset cells. The default is “d”.

- PosedgeSynchronizerReset/clock_in : *string* ;

  A property that specifies the name of the clock input of the synchronizer reset cells. The default is “clk”.

- PosedgeSynchronizerReset/data_out : *string* ;

  A property that specifies the name of the data output of the synchronizer reset cells. The default is “q”.

Examples

This example specifies the port name for the clock multiplexer cell and the port names of the clock shaper cell with its reset port active polarity. Specifying port names that match the port names of your cells simplifies later swapping the instances of the RTL cells with instances of the real cells.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(moda,rtl) {
  RtlCells {
    ClockMux {
      input0 : I0 ;
      input1 : I1 ;
      select : S0 ;
      output : O ;
    }
    ClkShaperResetActive {
      enable0        : E0 ;
      enable1        : E1 ;
      reset          : RB ;
      reset_polarity : active_low ;
      clock_in       : CK ;
      clock_out      : CKG ;
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TopModuleConnections

A wrapper used in the tiling flow to describe the DFT ports to create on each tile and how to connect them in the chip-level netlist.

Usage

```
TopModuleConnections(module_name) {
  CurrentDesign {
    input_ports          : port_name, ... ;
    output_ports         : port_name, ... ;
    output_port_sources  : input_port_name |
                           output_pin_name | n'b0 | n'bz, ... ;
  }
  ChildBlock(module_name) {
    file_name            : file_path_name ;
    language_format      : 2001 | sv2005 | sv2009 ;
    input_ports          : port_name, ... ;
    output_ports         : port_name, ... ;
    Instance(instance_name) {
      input_port_sources : input_port_name |
                           output_pin_name | n'b0 | n'bz, ... ;
    }
  }
}
```

Description

All ports are pre-created on each tile module in a tiling flow, and those are pre-connected into the chip-level netlist. You may already have an automated way of doing this for all your functional ports, and you may use the same flow to include the DFT ports. If you create your chip-level netlist by hand and it does not include the DFT ports, you can use this configuration syntax to define them and use the process_top_module_connections command to insert and connect them. Use the ChildBlock wrapper to describe the DFT ports of all child physical blocks in the CurrentDesign wrapper to describe the DFT ports on the current design.

Note

The module_name of the TopModuleConnections wrapper must match the module name of the current design.

Arguments

- CurrentDesign/input_ports : *port_name*, ... ;

  An optional property to list all the input ports you want to create and for the tool to connect in the current design. If the port is already present on the current design, the tool checks that it is an input. Otherwise, the tool creates it with the input direction.

  CurrentDesign/input_ports supports simple scalar and one-dimensional buses only.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- CurrentDesign/output_ports : *port_name*, ... ;

  An optional property to list all the output ports you want to create for the tool to connect in the current design. The number and order of the specified output ports must match the sources listed in the output_port_sources property. If the port is already present on the current design, the tool checks that it is an output. Otherwise, the tool creates it with the output direction.

  CurrentDesign/output_ports supports simple scalar and one-dimensional buses only.

- CurrentDesign/output_port_sources : *input_port_name* | *output_pin_name* | *n*'b0 | *n*'bz, ... ;

  An optional property to list the sources of all output ports specified in the CurrentDesign/output_ports property. The source count must match the number of output ports specified in the output_ports list, and the ordering must also be consistent. A connection must refer to a port found within the output_ports list of one of the Child module instances, or an input port of the current design. You can also use *n*'b0 to connect *n* ports to 0, or use *n*'bz to leave *n* ports unconnected.

  For example, if you specify a source as "c3_i1/out1", you must list out1 in the ChildBlock(c3)/output_ports property, and there must be an Instance(c3_i1) wrapper within the ChildBlock(c3) wrapper.

- ChildBlock(*module_name*)

  A repeatable wrapper to list all the child physical modules used in your tiled design. List each child physical block module only once, even if it has several instances.

  Note

  See Example 1 of the process_top_module_connections command to see how the design shown in Example 4 of the OCC wrapper uses only three ChildBlock wrappers, one for C1, C2, and C3, even though C1 and C2 are instantiated multiple times.

- ChildBlock/file_name : ***file_path_name***;

  A required property to specify the file that contains the module definition of the child physical block. The file must be compliant with those formats specified by the language_format property. The tool writes the modified module file to the same location as the original file with the default or specified extension suffix appended to the original filename. See the -extension_suffix switch description in the process_top_module_connections command for more information.

- ChildBlock/language_format : 2001 | sv2005 | sv2009;

  An optional property to specify the file format of the module definition. Verilog 2001 is the default. The tool supports Verilog 2001, SystemVerilog 2005, and SystemVerilog 2009.

- ChildBlock/input_ports : *input_port_name*, ...;

  An optional property to list all the input ports you want to create on the child block and for the tool to connect in the current design. The number and order of the specified input ports must match the Instance/input_port_sources properties. If the port is already present on the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

child block, the tool checks that it is an input. Otherwise, the tool creates it with the input direction.

ChildBlock/input_ports supports simple scalar and one-dimensional buses only.

- ChildBlock/output_ports : *output_port_name*, ...;

  An optional property to list all the output ports you want to create on the child block and for the tool to connect in the current design. If the port is already present on the child block, the tool checks that it is an output. Otherwise, the tool creates it with the output direction.

  ChildBlock/output_ports supports simple scalar and one-dimensional buses only.

- ChildBlock/Instance(***instance_name***)

  A required repeatable wrapper to list all instances of the child block found within the chip-level netlist.

- ChildBlock/Instance/input_port_sources : *input_port_name* | *output_pin_name* | *n*'b0 | *n*'bz, ...;

  An optional property to list the sources of all the input ports specified in the ChildBlock/input_ports property. The source count must match the number of input ports specified in the input_ports list, and the ordering must also be consistent. A connection must refer to a port found within the output_ports list of one of the Child module instances, or a port found in the input_ports list of the current design. You can also use *n*'b0 to connect *n* ports to 0, or use *n*'bz to leave *n* ports unconnected. See Example 2 for an example usage of the n'b0 value.

  For example, if you specify a source as "c3_i1/out1", you must list out1 in the ChildBlock(c3)/output_ports property, and there must be an Instance(c3_i1) wrapper within the ChildBlock(c3) wrapper.

Examples

The ports listed in the input_ports and output_ports properties cannot be complex ports. The tool only supports simple scalar and one-dimensional buses for those ports.

Example 1

The following example specifies the creation of two input ports and two output ports on two child blocks. The chip includes one instance of each of the two child blocks, B1 and B2. Input ports "in1" and "in2" and output ports "out1" and "out2" are created on each module, and the created output pins on one instance drive the created input pins of the other instance.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
TopModuleConnections(chip) {
  ChildBlock(B1) {
    input_ports          : in1, in2;
    output_ports         : out1, out2;
    Instance(B1_I1) {
      input_port_sources : B2_I1/out1, B2_I1/out2;
    }
  }
  ChildBlock(B2) {
    input_ports          : in1, in2;
    output_ports         : out1, out2;
    Instance(B2_I1) {
      input_port_sources : B1_I1/out1, B1_I1/out2;
    }
  }
}
```

See Example 1 of the process_top_module_connections command reference section for a more comprehensive example.

Example 2

The following example adds SSN connections on a wrapper module that includes two instances of a given tile. It uses the CurrentDesign wrapper to define the ports to create on the wrapper module, and 16’b0 in the second instance of the tile as the connection for its from_ssn_bus_data_out[15:0] port. This is because there are no tiles after it to connect to.

```
TopModuleConnections(block1) {
  CurrentDesign {
    input_ports          : ssn_bus_clock_in, ssn_bus_data_in[15:0];
    output_ports         : ssn_bus_data_out[15:0];
    output_port_sources  : T1_I1/ssn_bus_data_out[15:0];
  }
  ChildBlock(T1) {
    file_name            : T1.v;
    language_format      : 2001;
    input_ports          : ssn_bus_clock_in,
                           ssn_bus_data_in[15:0],
                           from_ssn_bus_data_out[15:0];
    output_ports         : ssn_bus_clock_out,
                           ssn_bus_data_out[15:0],
                           to_ssn_bus_data_in[15:0];
    Instance(T1_I1) {
      input_port_sources : ssn_bus_clock_in,
                           ssn_bus_data_in[15:0],
                           T1_I2/ssn_bus_data_out[15:0];
    }
    Instance(T1_I2) {
      input_port_sources : T1_I1/ssn_bus_clock_out,
                           T1_I1/to_ssn_bus_data_in[15:0],
                           16’b0;
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

PatternsSpecification Configuration Syntax

This chapter describes the configuration data syntax used inside Tessent Shell to specify patterns to be applied to DFT components that are inserted into a design.

This chapter covers the BoundaryScan, MemoryBist, MemoryBisr, LogicBist, and SdfInfo wrappers as they exist inside the PatternsSpecification wrapper. Using this syntax, it is possible to specify a wide variety of patterns specifications, in a single wrapper or in multiple PatternsSpecification wrappers, to generate verification testbenches used for simulation and tester patterns used for chip manufacturing.

Note

You can control the default values of many properties with the DefaultsSpecification/PatternsSpecification wrapper and the set_defaults_value command. The default values of properties marked with the notation "*DefSpec" in the usage line are controllable in this manner.

PatternsSpecification **4556**
Patterns **4563**
SimulationOptions 4568
AdvancedOptions 4574
TestStep 4579
TestbenchVerify 4668
ICLNetworkVerify 4669
ProcedureStep 4674
SSNContinuityVerify 4685
EnableGroupInfo **4687**
SdfInfo **4689**
LoadBoardInfo **4692**
TesterInterface 4694
Loopbacks 4697
ACLoopbacks 4699
IjtagRetargetingOptions **4701**
InSystemTest **4703**

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

PatternsSpecification

Defines a series of Patterns wrapper for a given view of a design.

Usage

```
PatternsSpecification(design_name, design_id, pattern_id) {
  usage                           : simulation | manufacturing_test ;
  manufacturing_patterns_formats : format, ... ;   // default: Stil
  compress_pattern_files          : off | on ;
  timeplate                       : timeplate_name ;
  SimulationOptions {
    testbench_module_name      : format_string ;
    current_design_view        : auto | full |
                                 ijtag_graybox | scan_graybox
[scan_graybox_variant_id] ;
    logic_bist_debug           : off | bist_registers_and_clock_verify |
                                 setup_and_clock_verify |
                                 setup_and_clock_verify_one_per_ncp |
                                 monitor_scan_cells ;
    verify_internal_data_pins : off | on ;
  }
  AdvancedOptions {
    procfile_name               : file_path ;
    parameter_file              : file_path ;
    Parameters {
     parameter_name             : parameter_value ;
    }
    ireset_in_test_setup_proc   : on | off ;
    ConstantPortSettings {
      port_name                 : 0 | 1 | - | power | pwr | ground | gnd ;
    }
  }
  Patterns(patterns_name) {
  }
  PatternsGroup(group_name) {
    AdvancedOptions {
      // same as ../AdvancedOptions
    }
    Patterns(patterns_name) {
    }
  }
  EnableGroupInfo(enable_group_info_id) {
  }
  SdfInfo(sdf_info_id) {
  }
  LoadBoardInfo (load_board_info_id) {
  }
  IjtagRetargetingOptions {
  }
  InSystemTest {
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Description

The PatternsSpecification wrapper defines a series of Patterns wrapper for a given view of a design. The Patterns wrapper can be declared directly at the top level or grouped into PatternsGroup wrappers.

The pattern specification uniquely identifies a design view of a design by using the *design_name* and *design_id* values. The create_patterns_specification command sets the *pattern_id* value to signoff or manufacturing based on the usage arguments. The *pattern_id* can later be modified to create various PatternsSpecifications. For example, you can create a patterns specification targeted for wafer probe test, final test, and board level test. The number of tester channels and power available in each test setup typically varies and patterns need to be adjusted accordingly.

Note

For Tessent SiliconInsight usage, the PatternsSpecification also includes a CharacterizationDebugOptions wrapper. Refer to "Characterization Debug Options Hierarchy in the Patterns Specification" in the *Tessent SiliconInsight User's Manual for Tessent Shell* for details.

Arguments

- *design_name*

  A string that identifies the PatternsSpecification for a specific design. This string is automatically populated by the create_dft_specification command to match the name of the current design. Refer to the description of the process_patterns_specification command to understand which PatternsSpecification gets processed.

- *design_id*

  A string that identifies the PatternsSpecification for a specific *design_id* of the current design. If this string is missing from the "get_context -design_id" command and switch, it is automatically populated by the create_patterns_specification command with the value of the tessent_design_id attribute found in the top ICL module. Refer to the description of the process_patterns_specification command to understand which PatternsSpecification the tool processes.

- *pattern_id*

  A string that distinguishes multiple PatternsSpecifications for a specific *design_name* and *design_id* from each other. The create_patterns_specification command populates this string with signoff and manufacturing but this string can be changed to any value to differentiate different patterns specifications from each other. For example, you can create a system_test version that is derived from the manufacturing one and include only the tests that can be operated through the TAP controller.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- usage : simulation | manufacturing_test ;

  A property that specifies whether the PatternsSpecification is to be used to generate simulation testbenches or manufacturing test patterns. The patterns are converted to Verilog testbenches when the usage is specified as “simulation” and to the specified manufacturing_patterns_formats when the usage is specified as “manufacturing_test”. Some settings are checked against this usage specification such as the “reduced_address_count” property in the MemoryBist wrapper.

- manufacturing_patterns_formats : *format*, ... ;

  A property that specifies the format of the patterns file to use when usage is specified as “manufacturing_test”. Valid formats are WGL, STIL, STIL2005, TITDL, FJTDL, MITDL, TSTL2, and SVF. The default format is STIL. For a description of these formats, see the description of the *format_switch* argument to the write_patterns command.

- compress_pattern_files : off | on ;

  A property that specifies whether the patterns files are to be compressed when written to disk. When the property is specified to on, the file has the extension *.gz* to signify that it is compressed.

- timeplate : *timeplate_name* ;

  A property that specifies the timeplate the tool uses to override the default timeplate created when generating the pattern. The timeplate is loaded from a procfile as specified from the PatternsSpecification/AdvancedOptions/procfile_name property. The timeplate and procfile_name properties are both inherited downwards.

- SimulationOptions/testbench_module_name : *format_string* ;

  A property that specifies the top-level module name in the Verilog testbench file. You can use this property to give a different testbench module name for each testbench so that more than one testbench can be compiled in a single work area of a Verilog simulator at the same time.

  The string can also include the %d, %i, %I and %p identifiers which is replaced by the values from PatternsSpecification(%d,%i,%I) and Patterns(%p). Using a unique name for the testbench allows you to use a common signal name file to display signals in your simulator including alias name definitions you may have created. The run_testbench_simulations command uses a separate directory to compile each testbench so keeping the testbench_module_name as TB is optimal and recommended.

- SimulationOptions/current_design_view : auto | full | ijtag_graybox | scan_graybox [*scan_graybox_variant_id*] ;

  A property that specifies the view for the current design. When you specify this property value here, all Patterns/SimulationOptions/current_design_view inherit this top value.

  auto — When you select this option or leave the property unspecified, the ijtag_graybox view is used for the current design if it exists, but only for Patterns/IclNetworkVerify simulations. This is the default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can specify a default value with the "set_defaults_value PatternsSpecification/SignOffOptions/current_design_view *view*" command prior to calling create_patterns_specification. Use this to specify "full" to override the "auto" behavior for ICLNetworkVerify patterns.

full — Force the tool to use the full view for the current design for ICLNetworkVerify patterns.

ijtag_graybox — The tool uses the ijtag_graybox design view. The tool generates an error when you run the process_patterns_specification command if an ijtag_graybox view is unavailable for the current design. The tool uses the full view in such a case.

scan_graybox [*scan_graybox_variant_id*] — The tool uses the scan_graybox design view, and if it exists, it will be used for simulation. The tool generates an error when you run the process_patterns_specification command if a scan_graybox view is unavailable for the current design. The tool uses the full view in such a case.

scan_graybox_variant_id — You can specify which scan_graybox to use with its variant ID if it was generated with "write_design -tsdb -graybox -variant_scan_graybox_id *variant_scan_graybox_id*" command.

- SimulationOptions/logic_bist_debug : off | bist_registers_and_clock_verify | setup_and_clock_verify | setup_and_clock_verify_one_per_ncp | monitor_scan_cells ;

  A property that causes the tool to provide extra data to help you debug LogicBIST simulation mismatches. Refer to "Pattern Mismatch Debugging" in the *Hybrid TK/LBIST User's Manual* for details. The valid values are:

  - off — The tool does not provide extra data for debugging.
  - bist_registers_and_clock_verify — The simulation generates a mismatch showing when the PRPG, MISR, and low-power registers (if present) diverge from their expected values. This can help reduce debug time because it identifies the scan pattern at which the register(s) begin to diverge.
  - setup_and_clock_verify — The tool creates a LogicBIST pattern that verifies that the controller is setup correctly, and then it applies 256 serial patterns to exercise the full NCP count range. During this test, the tool reduces the shift length to decrease the simulation runtime. Because of the reduced shift length, the MISR signature is not examined at the end of the pattern. If a clock does not generate the expected number of pulses for a given NCP, the tool generates a simulation mismatch.
  - setup_and_clock_verify_one_per_ncp — The same as setup_and_clock_verify, except the tool only applies a single scan pattern for each NCP.
  - monitor_scan_cells — When specified, the tool monitors the scan chains, and for each shift cycle, checks that the expected unload values are observed. If an unexpected value is observed, the tool reports the shift cycle and scan cell that maps to the failure.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- SimulationOptions/verify_internal_data_pins ;

  A property that specifies whether the patterns is to verify that data can be properly controlled at the internal DataInPorts and observed from the internal DataOutPorts. When the property is on, the internal DataOutPorts are forced and those values observed using ScanRegisters unloading. The values on internal DataInPorts are controlled using ScanRegisters loading and the resulting value directly observed on the internal DataInPorts. This feature is only usable in simulation and is useful for finding mismatches between the circuits and their ICL descriptions.

  Note that ICL Extraction sets the tessent_design_instance attribute to the pre-synthesis instance names whenever those names are available. During the creation of the data pin verification patterns, the actual design instance names are ignored. The verification pattern generator only uses the tessent_design_instance attribute to obtain the references to the design pins. Consequently, the rtl design files provided to ICL extraction must be used during the simulation of the verification patterns, at least in case that the synthesis modifies the design hierarchy (for example, because of "generate" loops). In order to obtain data pin verification patterns which can be simulated using the synthesized netlist with instance names that have been modified by synthesis, you either have to modify the tessent_design_instance attributes accordingly or run ICL Extraction based on the synthesized netlist from the beginning (without providing rtl files to the tool).

- AdvancedOptions/parameter_file : *file_path* ;

  A property that specifies the name of a file containing parameters used to configure the write_patterns command. For a description of the available parameters, see the "Parameter File Format and Keywords" on page 4997.

  The parameters specified in this file are overridden by those specified in the Parameters wrapper. The following parameters may be overridden by "process_patterns_specification":

  - SIM_TOP_NAME — if the property SimulationOptions/testbench_module_name is set, this parameter is ignored.
  - SIM_CLOCK_MONITOR — overridden.
  - ALL_EXCLUDE_POWER_GROUND — overridden when the property AdvancedOptions/include_all_power_pins is not "auto".
  - ALL_EXCLUDE_UNUSED — overridden when the property AdvancedOptions/include_only_used_pins is not "auto".

- AdvancedOptions/Parameters/*parameter_name* : *parameter_value* ;

  A wrapper that specifies parameter-value pairs used to configure the write_patterns command. A parameter specified in this wrapper overrides the same parameter specified in the specified parameter_file. For a description of the available parameters, see the "Parameter File Format and Keywords" on page 4997.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- AdvancedOptions/procfile_name : *file_path* ;

  A property that specifies a procfile name that contains a test_setup and/or a test_end procedure.

  The file_path must have one of the following formats:

  - A simple filename with no directory path. In this case, the file must exist beside the ICL file.
  - A null string. In this case, no procfile is used.
  - An absolute path starting with a "/".
  - A relative path starting with "./" or "../". In this case, the file must exist relative to the current working directory.

- AdvancedOptions/ireset_in_test_setup_proc : on | off ;

  A property which, when set to "on", suppresses the "iReset" being issued before the first pattern set generated for the Patterns wrapper being processed.

  A test_setup procedure must be provided in the procfile specified by the "procfile_name" property in the same AdvancedOptions wrapper.

  It is your responsibility to put appropriate events into the test_setup procedure that resets the IJTAG network.The effect of the ireset_in_test_setup_proc being on is to add the -ireset_in_test_setup_proc to the open_pattern_set command for the first pattern set generated by for the Patterns wrapper.

- AdvancedOptions/ConstantPortSettings/port_name : 0 | 1 | - | power | pwr | ground | gnd ;

  A data wrapper that specifies the ports that must be held constant throughout the entire pattern set. This list is typically empty and is not populated by the create_patterns_specification command. Input ports that were constrained prior to running extract_icl are automatically reported in the ICL file, and they are automatically forced accordingly when the process_patterns_specification command is called.

- PatternsGroup(*group_id*)

  A wrapper that groups a set of Patterns wrappers. The *group_id* value is a unique string containing only letters, numbers, and underscores that identifies the group. This option makes it convenient to fold that group into the GUI when using the add_config_tab command. It also enables you to select that group in the GUI to validate or process all Patterns wrappers found inside the group.

  You can also do this from the dofile using the process_patterns_specification command with the -config_object option referring to a PatternsGroup wrapper.

Examples

The following example shows a PatternsSpecification wrapper for module "MyDesign", design_identifier "rtl", and pattern_id set to signoff. The AdvancedOptions/Parameters wrapper is used to specify the SIM_PRECISION parameter. You typically do not need to set this

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

parameter as it is automatically set to provide accuracy of each clock period to within 1 percent. For example, a Patterns wrapper using a clock with a period of 4.5 ns and 450 ps automatically has a resolution of 1 ps such that the 450 ps half period of 225 ps can be expressed with more than 2 ps or 1 percent accuracy. It is shown here as an example, in case you want to set a higher precision.

```
PatternsSpecification(MyDesign,rtl,signoff) {
  usage : simulation ;
  AdvancedOptions {
    Parameters {
      SIM_PRECISION : 1ps ;
    }
  }
}
```

Related Topics

create_patterns_specification

process_patterns_specification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Patterns

Defines a complete set of Patterns steps each of which is meant to be run as one entity during simulation or manufacturing test.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(patterns_name) {
    tck_clock_only              : on | off ;
    tck_off_state               : 0 | 1 ;
    load_board_info             : load_board_info_id ;
    timeplate                   : timeplate_name ;
    DftControlSettings {
      icl_inst_path             : 1 | 0 ;
    }
    ClockPeriods {
      pin_name                  : tester[, freq_multiplier]| time ;
                                                        // repeatable
    }
    AdvancedOptions {
    }
    SimulationOptions {
    }
    TestbenchVerify(id) {  // mutually exclusive with ProcedureStep,
    }                      // ICLNetworkVerify, and TestStep wrappers
    ICLNetworkVerify(id) { // mutually exclusive with TestbenchVerify,
    }                      // ProcedureStep, and TestStep wrappers
    ProcedureStep(procedure_step_name) { //repeatable
    }
    SSNContinuityVerify(id){
    }
    TestStep(step_name) { //repeatable
    }
    IjtagRetargetingOptions {
    }
  }
}
```

Description

The Patterns wrapper defines a complete set of Patterns steps each of which is meant to be run as one entity during simulation or manufacturing test. When the PatternsSpecification/usage property is set to manufacturing_test, it is possible to split separate steps into multiple patterns file such that test program events such as VDD bumping or PMU measurements can be inserted in the middle of the patterns. See the AdvancedOptions/split_patterns_file property description in the ProcedureStep and TestStep wrappers for more information about patterns file splitting.

Note

For Tessent SiliconInsight usage, the Patterns wrapper also includes a CharacterizationDebugOptions wrapper. Refer to "Characterization Debug Options Hierarchy in the Patterns Specification" in the *Tessent SiliconInsight User's Manual for Tessent Shell* for details.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- tck_clock_only : on | off ;

  A property that specifies that all instruments in this Patterns wrapper should run with a TCK clock.

  Note

  The tck_clock_only property only has an effect on Patterns containing MemoryBIST tests.

  This property instructs the IJTAG solver to find a way to sensitize clock muxes and OCC on the clock paths to inject a TCK source. If there is no way to inject TCK from a TCKPort, an error is issued. The test can still be run at TCK if the traced functional ClockPorts are set as "tester" in the ClockPeriods wrapper. The patterns describe a signal identical to TCK on those functional ClockPorts, as if they were TCKPorts.

  The TCK injection needs to be present in the ICL descriptions of the OCC modules in order to be understood and used by the IJTAG solver. To make sure the OCC ICL is complete, you must set the OCC wrapper property include_clocks_in_icl_model to "on" before processing the DftSpecification with the process_dft_specification command.

- tck_off_state : 0 | 1 ;

  A property that specifies the off value of the TCK clock. By default, the off value is 0. See the description of the open_pattern_set command for a description of the TCK waveform based on the tck_off_state and the tck_ratio values.

- load_board_info : *load_board_info_id* ;

  A property used to reference a LoadBoardInfo wrapper. The tester interface and loopbacks specified in the pointed LoadBoardInfo wrapper is considered in the patterns and the Verilog testbench is created to reflect those if it is generated.

- timeplate : *timeplate_name* ;

  A property that specifies the timeplate the tool uses to override the default timeplate that is created when generating the pattern. The timeplate is loaded from a procfile as specified from the PatternsSpecification/Patterns/AdvancedOptions/procfile_name property.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The Patterns/timeplate property and Patterns/IjtagRetargetingOptions/tck_period property are mutually exclusive within the same Patterns wrapper. However, you can specify the IjtagRetargetingOptions/tck_period property at the PatternsSpecification level with a Patterns/timeplate property in the same specification.

```
PatternsSpecification(a,b,c) {
  IjtagRetargetingOptions {
    tck_period: 100ns;
  }
  Patterns(a) {
    timeplate: my_timeplate;
  }
  Patterns(b) {
    IjtagRetargetingOptions {
      tck_period: 120ns; //override the default 100ns spec value
    }
  }
}
```

- DftControlSettings/signal : 1 | 0 ;

  A data wrapper that specifies the values of static dft signals of type global_dft_control that were created during the DFT step using the add_dft_signals command. By default, all dft control signals of type global_dft_control are set to their reset value unless you explicitly specify them in the DftControlSettings wrapper.

  To report the available static dft signals that have the global_dft_control type, use the "report_static_dft_signal_settings -all" command when the ICL is elaborated.

  See Example 2 to understand how this wrapper is auto populated by the create_patterns_specification command.

  When the create_patterns_specification creates a Patterns wrapper, it often automatically fills out the DftControlSettings wrapper with some DFT signals. For every active physical block instance, all DFT signals which were registered with a -defaults_value_in_all_test that is different than its -reset_value are set to their specified -defaults_value_in_all_test values. When creating memoryBIST patterns, the nonscan_test DFT signal is also set.

- ClockPeriods/*pin_name* : tester[, *freq_multiplier*] | *time* ;

  A data wrapper that specifies the clocks that are to be started for the given pattern set. When you use the value "tester" for a given clock, the clock starts as a free-running synchronous clock operating at the tester period or at a multiple of the tester period by specifying an optional multiplier as 0.5 or a positive integer from 1 to 10.

  Note

  The <*freq_multiplier*> value is a string and must be specified exactly as "0.5" or as the exact integers from 1 to 10. The tool rejects such values as ".5" or "3.0".

  When the value is of type time, the clock starts as an asynchronous clock with its period set to the specified time value. Any such asynchronous clocks are present in the simulation

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

testbenches but only appear as annotations near the top of the patterns file when the usage is manufacturing_test. The test engineer must start the asynchronous clock with tester-specific commands. This is an example annotation that appears in a STIL file to report the asynchronous clocks:

```
Ann {* Begin_async_clocks *}
Ann {*   clkb = period: 16.00ns *}
Ann {* End_async_clocks *}
```

If a controller used in the current Patterns wrapper issues an iClock command on its clock port and the port traces to a top-level port that is not declared in the ClockPeriods wrapper, you get an error as shown here:

```
//  Error: Unable to trace the iClock
// 'core_rtl_tessent_mbist_c2_controller_inst.BIST_CLK' to a
//  valid clock source. Traced the system clock to clock source
// 'clkb'. The source of an iClock command must be driven by a
// primary input port of ICL type ClockPort that has been declared
// as a clock source with the add_clocks command.
```

To fix this type of error, you can either add missing DftControlSettings to configure the clock multiplexing, or you can add the missing clock port in the ClockPorts wrapper.

Examples

Example 1

The following example illustrates a Patterns wrapper that operates the tester and tck at 100ns. The example creates two asynchronous clocks that are needed by the memory BIST controllers operated in TestStep TS0 and TS1.

```
PatternsSpecification(design_name, design_id, pattern_id) {
  Patterns(patterns_name) {
    tester_period     : 100ns ;
    tck_ratio         : 1 ;
    ClockPeriods {
      Clka : 4.5ns ;
      clkb : 5ns ;
    }
    TestStep(TS0) {
      MemoryBist {
        ...
      }
    }
    TestStep(TS1) {
      MemoryBist {
        ...
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

This example shows how to deal with clock port cloning that may happen during layout. Imagine you had a clock port called clka on your core and after layout this clock port was split into eight ports where the original clock port drives a small percent of the original fanout and the seven new ports clka_1 to clka_7 drive the rest. You need to have this new clock toggling in the testbench whenever clka was toggling. The following example script takes the PatternsSpecification created by create_patterns_specification and clones the clka entry in all ClockPeriods wrapper it is already found. The new entries in the ClockPeriods wrapper tells process_patterns_specification to also start those clocks.

```
set spec [create_patterns_spec]
set new_clka_ports [get_name_list [get_ports clka_*]]
set ClockPeriods_wrappers [get_config_elements ClockPeriods -in $spec -hier]

foreach_in_collection ClockPeriods_wrapper $ClockPeriods_wrappers {
  if {[get_config_value clka -in $ClockPeriods_wrapper -exists]} {
    set period [get_config_value clka -in $ClockPeriods_wrapper]
    foreach new_clka_port $new_clka_ports {
      add_config_element $new_clka_port -in $ClockPeriods_wrapper -value $period
    }
  }
}
```

Related Topics

create_patterns_specification

process_patterns_specification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SimulationOptions

Specifies properties that affect the creation of the simulation testbenches.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(patterns_name) {
    SimulationOptions {
      current_design_view         : auto | full |
                                    ijtag_graybox | scan_graybox
[scan_graybox_variant_id] ;
      testbench_module_name       : format_string ; // default: TB
      monitor_internal_clock_pins : boolean ; // default: on | off
      sdf_info                    : sdf_info_id ;
      emulate_bisr_chains_in_lower_physical_blocks : on | off | auto;
      logic_bist_debug : off | bist_registers_and_clock_verify |
                         setup_and_clock_verify |
                         setup_and_clock_verify_one_per_ncp |
                         monitor_scan_cells ;
      LowerPhysicalBlockInstances {
        instance_name : full | graybox | ijtag_graybox |
                        interface ; //repeatable
      }
      TopModules {
        module_name   : file_path ; // repeatable
      }
      SimulationMacros {
        macro_name    : value ;     // repeatable
      }
      NonContactedTestOptions {
        default_inout_pad_decay_time : time ;
        PadDecayTimes {
          port_name   : time ;     // repeatable
        }
      }
    }
  }
}
```

Description

Specifies properties that affect the creation of the simulation testbenches.

Arguments

- current_design_view : auto | full | ijtag_graybox | scan_graybox \ [*scan_graybox_variant_id*] ;

  A property that specifies the view for the current design. When you specify this property value here, all Patterns/SimulationOptions/current_design_view inherit this top value.

  auto — When you select this option or leave the property unspecified, the ijtag_graybox view is used for the current design if it exists, but only for Patterns/IclNetworkVerify simulations. This is the default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can specify a default value with the "set_defaults_value PatternsSpecification/SignOffOptions/current_design_view *view*" command prior to calling create_patterns_specification. Use this to specify "full" to override the "auto" behavior for ICLNetworkVerify patterns.

full — Force the tool to use the full view for the current design for ICLNetworkVerify patterns.

ijtag_graybox — The tool uses the ijtag_graybox design view. The tool generates an error when you run the process_patterns_specification command if an ijtag_graybox view is unavailable for the current design. The tool uses the full view in such a case.

scan_graybox [*scan_graybox_variant_id*] — The tool uses the scan_graybox design view, and if it exists, it will be used for simulation. The tool generates an error when you run the process_patterns_specification command if a scan_graybox view is unavailable for the current design. The tool uses the full view in such a case.

scan_graybox_variant_id — You can specify which scan_graybox to use with its variant ID if it was generated with "write_design -tsdb -graybox -variant_scan_graybox_id *variant_scan_graybox_id*" command.

This property is also used by LBIST self-test to determine the current design view to be loaded during the self-test simulation run. When the value is set to "auto", the tool uses the ijtag_graybox view if it exists. If the ijtag_graybox view does not exist, the tool uses the full view. When the value is set to "full" or "ijtag_graybox", the tool uses the corresponding view if it exists. If the view does not exist, the tool reports an error. When the value is set to "scan_graybox", the tool ignores this and proceeds as though the value was set to "auto".

- testbench_module_name : *format_string* ;

  A property that specifies the top-level module name in the Verilog testbench file. You can use this property to give a different testbench module name for each testbench so that more than one testbench can be compiled in a single work area of a Verilog simulator at the same time.

  The default is "TB". The string can also include %d, which is replaced by the name of the current design, or %p, which is replaced by the name of the Patterns wrapper. Using a unique name for the testbench enables you to use a common signal name file to display signals in your simulator, including alias name definitions you may have created. The run_testbench_simulations command uses a separate directory to compile each testbench, so keeping the testbench_module_name as TB is optimal and recommended.

- monitor_internal_clock_pins : *boolean* ;

  A property that specifies whether the simulation testbench is to monitor the internal clock pins. When specified to on, those pins that have an iClock defined in the PDL are monitored to make sure the correct frequency reaches them in simulation. The expected period is computed by the iClock command. The ICL clock pin names are converted to design pin names using the value of the tessent_design_instance property that the tool added to the ICL instances during ICL extraction.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

During layout, ungrouping often makes internal clock monitoring unusable as the design hierarchy is required to find the internal clock pins. If this happens, you must either turn this property off or use a Verilog `define macro to turn off the monitoring. See the "Clock Monitoring During Simulation" section under the run_testbench_simulations command for an example of the available macros.

If this property is not specified, it defaults to the value defined in the wrapper ../../SimulationOptions. This means the PatternsSpecification/SimulationOptions wrapper when there is no intermediate PatternsGroup wrapper; otherwise, the PatternsSpecification/PatternsGroup/SimulationOptions wrapper.

- sdf_info : *sdf_info_id* ;

  A property used to reference an SdfInfo wrapper. The SDF file identified in the SdfInfo wrapper is applied to the design when running the simulation of this pattern set.

- emulate_bisr_chains_in_lower_physical_blocks : on | off | auto ;

  A property that specifies whether the tool should create behavioral code in a side Verilog module (see "patterns" on page 4827) to emulate the behavior of the BISR segments within the child physical block instances. The valid values are:

  auto — The effect of the auto setting is dependent upon the view specified in the LowerPhysicalBlockInstances wrapper:

  - If you specify an interface simulation view, auto will resolve to on, enabling BISR chain emulation.

    Note

    Not specifying a lower physical block instance in the LowerPhysicalBlockInstances wrapper is equivalent to specifying an interface simulation view. In this case, auto will resolve to on, enabling BISR chain emulation.

  - If you specify the ijtag_graybox simulation view for a child instance and it is available, auto will resolve to off, disabling BISR chain emulation. The simulation uses the ijtag_graybox view.
  - If you specify the ijtag_graybox simulation view for a child instance and it is not available, the tool reports an error.
  - If you specify the ijtag_graybox simulation view for a child instance that is available, but a grandchild's ijtag_graybox view is not available, the tool uses the grandchild's full simulation view. In this case, auto resolves to off, disabling BISR chain emulation.

  on — BISR chain emulation is enabled for all child physical block instances. The emulation logic overrides the BISR segments inside the child block instance even if the full view of the child physical block instance has been loaded.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

off — BISR chain emulation is disabled for all child physical block instances.

The tool uses hierarchical forces and observe commands to emulate the behavior of the BISR segments on the port of the child physical block instance. This enables you to simulate the BISR operation at the top level if you are using interface or scan graybox models for the child physical block instances.

When compiling the testbench into the simulator, you can specify the string "+define+Full_<PhysicalBlockModuleName>" to strip out the behavioral BISR emulation code associated with the block name, and let the simulation go through the full or IJTAG graybox view of the child physical block.

- logic_bist_debug : off | bist_registers_and_clock_verify | setup_and_clock_verify | setup_and_clock_verify_one_per_ncp | monitor_scan_cells ;

  A property that causes the tool to provide extra data to help you debug LogicBIST simulation mismatches. Refer to "Pattern Mismatch Debugging" in the *Hybrid TK/LBIST User's Manual* for details. The valid values are:

  off — The tool does not provide extra data for debugging.

  bist_registers_and_clock_verify — The simulation generates a mismatch showing when the PRPG, MISR, and low-power registers (if present) diverge from their expected values. This can help reduce debug time because it identifies the scan pattern at which the register(s) begin to diverge.

  setup_and_clock_verify — The tool creates a LogicBIST pattern that verifies that the controller is setup correctly, and then it applies 256 serial patterns to exercise the full NCP count range. During this test, the tool reduces the shift length to decrease the simulation runtime. Because of the reduced shift length, the MISR signature is not examined at the end of the pattern. If a clock does not generate the expected number of pulses for a given NCP, the tool generates a simulation mismatch.

  setup_and_clock_verify_one_per_ncp — The same as setup_and_clock_verify, except the tool only applies a single scan pattern for each NCP.

  monitor_scan_cells — The simulation generates a mismatch that shows when the scan chain is observed unloading an expected value. In addition, the tool reports the shift cycle and scan cell that maps to the failure.

- LowerPhysicalBlockInstances/*instance_name* : full | graybox | ijtag_graybox | interface ;

  A data wrapper that specifies which view of the lower physical block instances is to be used for this simulation. By default, the interface view (the one containing only the port definition of the instance) is used unless the IJTAG solver detects that the IJTAG patterns go through a block instance that is not listed in this wrapper, in which case the IJTAG graybox view is selected if available in the TSDB; otherwise, it uses the full view.

  As documented in the emulate_bisr_chains_in_lower_physical_blocks property above, there is also a special treatment for unspecified child physical blocks in the MemoryBisr wrapper when the emulate_bisr_chains_in_lower_physical_blocks property is set to auto. When an IJTAG graybox view is available for a given child physical block and all its

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

descendants, the IJTAG graybox view is automatically selected and the emulate_bisr_chains_in_lower_physical_blocks property inferred to off.

When a child physical block instance is specified as “full”, any child physical or sub-block found below it that is not specified in this wrapper also uses the full view. When a child physical block instance is specified as “ijtag_graybox”, any child physical or sub-block found below it that is not specified in this wrapper also uses the IJTAG graybox view, unless it is not found in its TSDB, in which case the full view is used.

If the pattern set operates controllers or instruments found inside some lower physical block instances, the full view of those instances must be used. The Patterns wrapper created by the create_patterns_specification command automatically adds the correct instances that correspond to the controller that it added in the given pattern set. You only need to edit this property if you chose to modify the group of controllers the given patterns wrapper operates on.

For the ICLNetworkVerify wrapper, you can use the graybox value when a scan graybox view is available for a lower physical block instance. Selecting this value generates a simulation testbench that automatically excludes all ICL instances that are not used to set up the scan modes (ICL module attribute keep_active_during_scan_test is false).

This property is also used by LBIST self-test to determine the views to be loaded for any child blocks. When the value is set to “full” or “ijtag_graybox”, the tool uses the corresponding view for the child block if it exists. If the view does not exist, the tool reports an error. If the value is unspecified or set to “graybox” or “interface”, the tool uses the ijtag_graybox view if it exists. If the ijtag_graybox view does not exist, the tool uses the full view.

Note

When the full view is used by LBIST self-test for either a parent or child instance view, and is not explicitly specified, LBIST reports a warning.

- TopModules/*module_name* : *file_path* ;

  Many modules can be added to the simulation of the parent Patterns(*pattern_name*) wrapper in which the SimulationOptions/TopModules resides. For each module *module_name*, the file *file_path* in which this module is found is required and only Verilog files are allowed. During the simulation of the Patterns, the file specified by *file_path* is compiled at the same time as the testbench and then, the module *module_name* is loaded during simulation.

  This is equivalent to doing the following on the run_testbench_simulations command line:

```
run_testbench_simulations -select <pattern_name> \
  -extra_top_modules [list <module_name> …] \
  -extra_verilog_files [list <file_path> …] \
  …
```

  Since the TopModules can be specified on a per-Patterns basis, it enables finer control of the simulation process but without having to call the run_testbench_simulations command for each Patterns(*pattern_name*).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- SimulationMacros/*macro_name* : *value* ;

  Each *macro_name* and optional *value* specified in this wrapper translate to the command line argument '+define+…' when the testbench and top modules are compiled during simulation.

  For example, for the following wrapper inside a parent Patterns(*pattern_name*)/ SimulationOptions wrapper:

  ```
  SimulationMacros {
    enable_faults;
    fault_type : 2;
  }
  ```

  Translate to the following command line arguments when compiling the testbench and top modules:

  ```
    +define+enable_faults
    +define+fault_type=2
  ```

  This is equivalent to doing the following on the run_testbench_simulations command line:

  ```
  run_testbench_simulations -select <pattern_name> \
    -simulation_macro_definitions [list enable_faults fault_type=2] \
    …
  ```

  Since the SimulationMacros can be specified on a per-Patterns basis, it enables finer control of the simulation process but without having to call the run_testbench_simulations command for each Patterns(*pattern_name*).

- NonContactedTestOptions/default_inout_pad_decay_time : *time* ;

  A property that specifies the default inout port decay time that is used in leakage_non_contacted testing. Unique port decay times can be specified in the PadDecayTimes wrapper.

- NonContactedTestOptions/PadDecayTimes/*port_name* : *time* ;

  A repeatable property that specifies an inout port name and pad decay time. These attributes are used in leakage_non_contacted testing. If unspecified, the port decay time defaults to the default_inout_pad_decay_time.

Related Topics

create_patterns_specification

process_patterns_specification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

AdvancedOptions

Specifies special advanced properties that affect various aspects of the pattern set.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(patterns_name) {
    AdvancedOptions {
      parameter_file              : file_path ;
      Parameters {
        parameter_name : parameter_value ; // repeatable
      }
      include_all_power_pins               : on | off | auto ;
      include_only_used_pins               : all_contacted | on | off
                                             | auto ;
      exclude_initial_ireset_from_patterns : on | off ;
      skip_test_setup             : on | off ;
      skip_test_end               : on | off ;
      procfile_name               : filename ;
      ireset_in_test_setup_proc   : on | off ;
      no_initialization           : on | off ;
        // defaults to ../../AdvancedOptions/procfile_name
      ConstantPortSettings {
        // cumulates and overrides constant port settings in
        // ../../AdvancedOptions
        port_name                 : 0 | 1 | - | power | pwr | ground
                                    | gnd;
      }
    }
  }
}
```

Description

The AdvancedOptions wrapper is used to specify special advanced properties that affect various aspects of the pattern set.

See the LoadBoardInfo wrapper description and load_board_info property for more control over load board contacted ports and loopbacks.

Arguments

- parameter_file : *file_path* ;

  A property that specifies the name of a file containing parameters used to configure the write_patterns command. For a description of the available parameters, see the "Parameter File Format and Keywords" on page 4997. When the property is not specified, the value of the ../../AdvancedOptions/parameter_file property is used.

  The parameters specified in this file are overridden by those specified in the Parameters wrapper. The following parameters may be overridden by "process_patterns_specification":

  - SIM_TOP_NAME — if the property SimulationOptions/testbench_module_name is set, this parameter is ignored.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

  - SIM_CLOCK_MONITOR — overridden.
  - ALL_EXCLUDE_POWER_GROUND — overridden when the property AdvancedOptions/include_all_power_pins is not “auto”.
  - ALL_EXCLUDE_UNUSED — overridden when the property AdvancedOptions/include_only_used_pins is not “auto”".

- Parameters/*parameter_name* : *parameter_value* ;

  A wrapper that specifies parameter-value pairs used to configure the write_patterns command. A parameter specified in this wrapper overrides the same parameter specified in the specified parameter_file. For description of the available parameters, see the “Parameter File Format and Keywords” on page 4997, The list of parameters found in this wrapper are added to any parameter specified in the ../../AdvancedOptions/Parameters wrapper.

- include_all_power_pins : on | off | auto ;

  A property that specifies whether or not to include power and ground ports in the patterns file. The power and ground ports are specified using the “set_attribute_value -name *function* -value power | ground” command. When set to auto, the power and ground ports are included in the simulation testbenches but excluded from the manufacturing patterns file because the power and ground pins are not driven by the tester channels but directly from the load board.

  Depending on its value, this property takes control of the value of the parameter ALL_EXCLUDE_POWER_GROUND, which can be set directly in the Parameters wrapper.

  - The ALL_EXCLUDE_POWER_GROUND parameter keyword is used only when the include_all_power_pins property is set to auto (the default) or is not set.
  - The ALL_EXCLUDE_POWER_GROUND parameter keyword is overridden when the include_all_power_pins property is set to on or off.

- include_only_used_pins : all_contacted | on | off | auto ;

  A property that specifies whether to include all non-power or ground ports in the patterns file or only the one that are actually used in the current pattern set. The used port list is the list of ports explicitly forced or sampled at least once in the pattern set.

  Depending on its value, this property takes control of the value of the parameter ALL_EXCLUDE_UNUSED, which can be set directly in the Parameters wrapper.

  - The ALL_EXCLUDE_UNUSED parameter keyword is used only when the include_only_used_pins property is set to auto (the default) or is not set.
  - The ALL_EXCLUDE_UNUSED parameter keyword is overridden when the include_only_used_pins property is set to on or off.
  - The ALL_EXCLUDE_UNUSED parameter keyword is overridden when the include_only_used_pins property is set to all_contacted and, if the load_board_info is used and describes a TesterInterface, the “set_write_patterns_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

-existing_used_ports" is called to contain all ports that are contacted (control and/or observation is not "none").

See LoadBoardInfo wrapper description and load_board_info property for more control over load board contacted ports and loopbacks.

- exclude_initial_ireset_from_patterns : on | off ;

  A property that specifies whether or not to include an asynchronous reset of the IJTAG network in the manufacturing patterns. When set to "off", the manufacturing patterns begin with an asynchronous reset of the IJTAG network. When set to "on", the pattern begins the execution while assuming that the IJTAG network is in a reset state.

  When this property is set to "on", the IJTAG network must have been restored to its reset state otherwise the execution of the current pattern fails. In order to bring back the IJTAG network to a reset state, a previous pattern must have been generated with the TestStep/AdvancedOptions/network_end_state : reset and executed before the current pattern is executed.

  This property is typically used when running the memoryBIST pre-repair patterns and memoryBISR fuse programming patterns while implementing the memory repair flow. It is important to disable the initial IJTAG reset in these patterns in order to preserve any memory repair information that was calculated in previous patterns.

  Note that this property removes the initial IJTAG reset from the manufacturing patterns only. When generating simulation testbenches, the initial IJTAG reset is applied in order to ensure a valid IJTAG network state during simulation.

- skip_test_setup : on | off ;

  A property that specifies that the optional test_setup procedure that was loaded from the identified procfile using the Patterns/profile_name property is to be skipped from the current patterns file. If you do that, the pattern set is no longer self contained. In manufacturing test, the test_setup pattern must be applies to the device once and the state of the device that was obtained by the test_setup procedure, and on which the current pattern set depends on, must be maintained by the tester before running the current pattern set. This typically involves maintaining the power to the device and the value of input pins when switching from one pattern set to the next.

- skip_test_end : on | off ;

  A property that specifies that the optional test_end procedure that was loaded from the identified procfile using the Patterns/profile_name property command is to be skipped from the current patterns file

- procfile_name : *file_path* ;

  A property that specifies a procfile name that contains a test_setup and/or a test_end procedure. This property, when unspecified. takes its value from the ../../AdvancedOptions/profile_name property found in the PatternsSpecification or PatternsGroup wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The *file_path* must have one of the format:

- It must be a null string. In this case, no profile is used.
- It must be an absolute path stating with a "/".
- It must be a relative path starting with "./". In this case, the file must exist relative to the current working directory.
- It must be a simple filename with no directory path. In this case, the file must exist beside the ICL file.

- ireset_in_test_setup_proc : on | off ;

  A property which, when set to "on", suppresses the "iReset" being issued before the first pattern set generated for the Patterns wrapper being processed.

  A test_setup procedure must be provided in the procfile specified by the "procfile_name" property in the same AdvancedOptions wrapper.

  It is your responsibility to put appropriate events into the test_setup procedure that resets the IJTAG network.

  The effect of the "ireset_in_test_setup_proc" being "on" is to add the "-ireset_in_test_setup_proc" to the "open_pattern_set" command for the first pattern set generated by for the Patterns wrapper. See the open_pattern_set command for more information.

- no_initialization : on | off ;

  A property that specifies creating the initialization cycle in the pattern file. The default is off. This property, when unspecified, takes its value from the property specified in the following wrapper:

  ```
  PatternsSpecification/AdvancedOptions
  ```

- ConstantPortSettings/port_name : 0 | 1 | - | power | pwr | ground | gnd ;

  The command takes the following actions based on the value assigned to a port:

  - The ports identified with 0 or 1 has their value set to 0 or 1 as in the testbench and in the pattern.
  - The ports identified with '-' are ignored.
  - The ports identified with 'power', 'pwr', 'ground' or 'gnd' have their attribute 'function' set to 'power' or 'ground', which allows specifing them as such despite those ports not being listed in the ICL in the 'tessent_power_port_list' or 'tessent_ground_port_list' attributes. Such ports can be added automatically to the ConstantPortSettings wrapper by setting their 'function' attribute prior to running the create_patterns_specification command.

  A data wrapper that specifies the ports that must be held constant throughout the entire pattern set. This list is typically empty. The ConstantPortSettings wrapper can be used in the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BSDL-only flow (also refer to BoundaryScan for further information). Input ports that were constrained prior to running extract_icl are automatically reported in the ICL file and they are automatically forced when process_patterns_specification is called. Port settings listed in this wrapper accumulate and replace the port settings defined in ../../AdvancedOptions/ ConstantPortSettings wrapper found in the PatternsSpecification and PatternsGroup wrappers.

If you have used add_dft_signals scan_en to define top level ports as DFT signals of type scan_en, the create_patterns_specification command automatically adds the scan enable ports to the AdvancedOptions/ConstantPortSettings wrapper such that they are automatically asserted off during the pattern generation. If you have other scan enable ports not identified with the add_dft_signals, you need to add them yourself in the create PatternsSpecification using this script:

```
set spec [create_patterns_specification]
# Get a collection object to the
# 'ConstantPortSettings' wrapper
set ConstantPortSettings \
  [get_config_value  -in $spec -object \
      AdvancedOptions/ConstantPortSettings]

# Add entries to 'ConstantPortSettings'
# wrapper for some ports that must be
# forced to '0'.
set scan_en_port_list {se1 se2}
foreach scan_en $scan_en_port_list {
  add_config_element $scan_en \
      -in $ConstantPortSettings -value 0
}
```

Examples

The following example uses the include_only_used_pins property to only include the used ports. The generated stil file excludes the unused ports. By default, all ports would have been listed in the patterns file even if they had no activity.

```
PatternsSpecification(my_design, final, manufacturing) {
  usage : manufacturing_test ;
  manufacturing_patterns_formats : stil ;
  Patterns(mbist1) {
    AdvancedOptions {
      include_only_used_pins : on ;
    }
  }
}
```

Related Topics

create_patterns_specification

process_patterns_specification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TestStep

Inserts a test step in the pattern set.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(patterns_name) {
    TestStep(step_name) {
      AdvancedOptions {
        execution_target : always | in_system_test_only | tester_only ;
        force_voltage(pin_name) : voltage ; // repeatable
        split_patterns_file     : on | off ;
        network_end_state       : keep | reset | initial ;
        IjtagRetargetingOptions {
          merge_irunloop_only   : on | off ;
        }
      }
      BoundaryScan {
      }
      MemoryBist {
      }
      MemoryBisr {
      }
      LogicBist {
      }
    }
  }
}
```

Description

The TestStep wrapper is used to insert a test step in the pattern set. The test step includes the selection of Boundary Scan tests, MemoryBIST or LogicBIST tests.

Note

For Tessent SiliconInsight usage, the TestStep wrapper also includes a CharacterizationDebugOptions wrapper. Refer to "Characterization Debug Options Hierarchy in the Patterns Specification" in the *Tessent SiliconInsight User's Manual for Tessent Shell* for details.

Arguments

- *step_name*

  A string that uniquely identifies the step within a pattern set. All procedure_step_name and test_step_name strings must be unique within a Patterns wrapper. The string is only allowed to contain letters, numbers, and underscores.

- AdvancedOptions/execution_target : always | in_system_test_only | tester_only ;

  A property that specifies whether the current test step should be retargeted only by an In-System Test (IST) controller, only by a tester, or by both of these. Use this property to avoid pattern duplication and to make pattern customization more robust.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The default is “always”, meaning that the current test step is solved. The “in_system_test_only” option restricts the current test step to top-level host scan interfaces identified as IST controllers. Use the “tester_only” option when the top host scan interface is a tester and hence there is an external I/O source.

- AdvancedOptions/force_voltage(*pin_name*) : *voltage* ;

  A repeatable property that specifies that the current test step needs a special voltage to be applied on a specific port of the device. This is typically used in TestStep where MemoryBisr is used to program the fusebox. The property does not actually force the voltage. It only generates a special annotation near the top of the patterns file to let the test engineer know that test program commands are needed to supply the needed voltage on the given port. As soon as a single force_voltage property is specified, the split_patterns_file : on; property is implied to allow the insertion of the test program commands needed to apply the voltage values.

  Below is an example of the special annotation inside a stil file:

```
Ann {* Before applying this pattern, the following actions are required: *}
Ann {*  - Apply +0.25V on port PA *}
Ann {*  - Apply -0.25V on port PB *}
```

- AdvancedOptions/split_patterns_file : on | off ;

  A property that causes the pattern set to be split into multiple files. The file containing the previous patterns is closed and a new one is created for the current step. The filename is the name of the Patterns wrapper initially. When a step with split_patterns_file set to on is reached, the new filename consists of the name of the Patterns wrapper concatenated with an underscore to the name of the TestStep wrapper as shown here:

```
<pattern_name>_<test_step_name>
```

  The split patterns file contains a special annotation near the top of the file that reads like this:

```
Ann {*  This pattern file requires some actions taken on the ATE. *}
Ann {*  This pattern assumes it is following another one: *}
Ann {*  the power and clocks must be kept alive between *}
Ann {*  the two patterns and all pins kept their state. *}
```

- AdvancedOptions/network_end_state : keep | reset | *initial* ;

  A property that specifies the action to take at the end of the TestStep. The options for this property are named the same and perform the identical functions as the close_pattern_set command's arguments. For details, see the close_pattern_set command.

- AdvancedOptions/IjtagRetargetingOptions/merge_irunloop_only : on | off ;

  A property that specifies if iApply statements of concurrent iCalls are allowed to be merged during the processing of an iMerge block. This property and its options perform the identical function as “set_ijtag_retargeting_options -merge_irunloop_only”. Refer to the set_ijtag_retargeting_options command for more information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example defines two TestStep wrappers in series. The first one runs memory BIST controllers while the second ones runs a MemoryBisr controller.

```
PatternsSpecification(mychip, gate, signoff) {
  Patterns(P1) {
    TestStep(TS1) {
      MemoryBist {
        Controller(corea.ctl1) {
        }
        Controller(corea.ctl2) {
        }
      }
    }
    TestStep(TS2) {
      MemoryBisr {
        Controller(ctl1) {
        }
      }
    }
  }
}
```

Related Topics

create_patterns_specification

process_patterns_specification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BoundaryScan

Specifies the boundary scan tests to be generated.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(patterns_name) {
    TestStep(step_name){
      BoundaryScan {
        bsdl_file                  : file_path ;
        bsdl_package_directories   : dir_path | auto ;
        enable_group_info          : enable_group_info_id, ... ;
        bonding_configuration      : name ;
        pin_action_verbosity       : on | off ;
        active_control_cells       : unlimited | integer ;
        extest_train_pulse_count   : auto | integer ;
        instruction_reset_value    : binary ;
        RunTest(test) {
        }
      }
    }
  }
}
```

Description

The BoundaryScan wrapper specifies the boundary scan tests to be generated.

As illustrated in its syntax, the BoundaryScan specification consists of RunTest wrappers and properties that affect all RunTest wrappers.

The repeatable RunTest wrapper defines the tests to be generated in the current pattern set. At least one RunTest wrapper is required, and no more than one BoundaryScan wrapper may be specified per TestStep wrapper.

The tool identifies the boundary scan register during extraction with a set of annotations to the STIL pattern set. See "Symbolic Variables Specific to Boundary Scan Patterns" in the Tessent IJTAG User's Manual for complete information.

Arguments

- bsdl_file : *file_path* ;

  A property that specifies the path and filename of the Boundary Scan Description Language (BSDL) file to be used for BSDL-only flow pattern generation. In the BSDL-only pattern generation flow, only the BSDL file is required. You do not need to specify the BSDL file if it can be found in an instrument container in the TSDB directory. Packages can optionally be specified in the BSDL file and included with the bsdl_package_directories argument. For more information, refer to "BSDL-Only Flow" in the *Tessent BoundaryScan User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In addition to an absolute or relative path to the BSDL files, you can use the following two syntactical forms:

- dft_inserted_design(***<module_name>***_***<design_id>***)/***<bsdl_leaf_file_name>***
- instrument(***<module_name>***_***<design_id>***_*<instrument_type>*)/ ***<bsdl_leaf_file_name>***

  See "create_patterns_specification" on page 614 for complete information.

Note

In the BSDL-only flow, it may not be possible to use the run_testbench_simulations command since the data needed to run this command may not be available.

- bsdl_package_directories : *dir_path* | auto ;

  A property that specifies a single path, or a list of paths to the package include directories. If *dir_path* is not specified, or if auto is specified, the path is assumed to be the same directory where the BSDL file is located.

- enable_group_info : *enable_group_info_id*, ... ;

  A property that specifies one or more EnableGroupInfo wrappers to use for the boundary scan patterns. If more than one EnableGroupInfo wrapper is specified, the tool automatically extends the test to multiple sequential tests that enable each EnableGroupInfo wrapper separately.

  All enable cells not part of the listed enable groups, whether they are part of a not listed enable group or part of no enable group, are added to a new artificial group and tested all simultaneously.

  If an EnableGroupInfo(enable_group_info_id) wrapper is not found in the PatternsSpecification wrapper, the tool generates an error listing the enable_group_info_ids that are available. If the user specifies "enable_group_info : *;", the tool automatically adds all available enable_group_info_ids to this property.

- bonding_configuration : *name* ;

  A property that selects which bonding configuration the current patterns are generated for. You must specify this property if your Tessent BoundaryScan hardware was generated with one or more BondingConfigurations. The name string must refer to a valid BondingConfigurations(name). Tessent Shell searches for the corresponding BSDL file <design_name>_<name>.bsdl in the TSDB.

  You also use this property to specify SegmentSelections in the Embedded BoundaryScan flow. See the EmbeddedBoundaryScan wrapper "BondingConfigurations" on page 4112 for more information.

  This property cannot be used if bsdl_file is specified.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- pin_action_verbosity : on | off ;

  A property that enables you to specify whether your simulator displays comments each time a pin is compared to a specific value during testbench simulation. Be aware that adding comments for each output comparison increases pattern storage size.

- active_control_cells : unlimited | *integer* ;

  A property that specifies how many enable boundary-scan cells can be active simultaneously during the clamp and output tests. Valid integer values are 1 or greater.

  A large value generates a shorter overall test. Using a small value reduces the noise of simultaneously switching outputs created by test fixtures and package inductance.

  If the enable_group_info property is specified and the integer is smaller than the number of control cells in an enable_group, the enable group is split up and run sequentially in smaller groups of integer cells simultaneously.

- extest_train_pulse_count : auto | *integer* ;

  A property that specifies whether to add the EXTEST_TRAIN test to the dot6_ac_* run tests. Valid integer values are 0 or greater. The default is “auto,” meaning that for the Boundary Scan and Embedded Boundary Scan flows, the tool takes the value from the AIO_EXTEST_Train_Execution attribute in the BSDL file. The tool assigns the value of the property to 0 if that attribute is not present.

  The dot6_ac_* run tests include the EXTEST_TRAIN instruction if the value is a positive integer. The EXTEST_TRAIN instruction is not tested if the value is 0. Set the value to 0 for the legacy behavior.

- instruction_reset_value : binary ;

  This property is for the BSDL-only flow to specify the initial instruction register value when certain bits (for example, user bits) need a defined state for BoundaryScan operation. You can only use this property in a chip-level flow and when there is no ICL description available.

  Note

  This property corresponds to the ResetValue of the instruction ICL scan register. If you load a design ICL description, specify the value in that description instead of using this property.

Examples

The following example shows the PatternsSpecification of a standard set of 1149.1 tests with a reduction in the patterns storage size due to turning off pin action verbosity and a reduction in switching activity due to limiting the number of simultaneously active control cells.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
PatternsSpecification(CHIP,signoff) {
   Patterns(std_test) {
      TestStep(TS1) {
         BoundaryScan {
            pin_action_verbosity: off ;
            active_control_cells: 1 ;
            RunTest(test_logic_reset){}
            RunTest(inst_reg) {}
            RunTest(id_Reg) {}
            RunTest(bypass_reg) {}
            RunTest(bscan_reg) {}
            RunTest(input) {}
            RunTest(sample) {}
            RunTest(disbaled_outputs) {}
            RunTest(highz_mode) {}
            RunTest(output_clamp) {}
         }
      }
   }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

RunTest

Defines a TAP or IO test to be run.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(patterns_name) {
    TestStep {
      BoundaryScan {
        RunTest(test_name) {
          AdvancedOptions {
            checker_board           : on | off ;
          }
          NonContactedPinOptions {
            LeakageOptions {
              logic_level           : both | iih | iil ;
              number_of_cycles      : int ; // default: 0
            }
            TriStateEnableOptions {
              test_path            : both | local | global ;
              logic_level          : both | high | low ;
              number_of_rti_cycles : integer ; // default: 1
            }
          }
          CellCompares {
            cell_number   : 0 | 1 | X ;        // repeatable
          }
          CellSettings {
            cell_number   : 0 | 1 ;            // repeatable
          }
          MaskedPins {
            pin_name      : normal | masked ;  // repeatable
          }
          PinSettingsOverride {
            pin_name      : 0 | 1 | Z ;        // repeatable
          }
          PinComparesOverride {
            pin_name      : 0 | 1 | X | Z ;    // repeatable
          }
        }
      }
    }
  }
}
```

Description

The RunTest wrapper defines a TAP or IO test to be run. Using the sub-wrappers in the RunTest wrapper, you can customize pin or cell options that apply to this test only. This wrapper needs to be repeated once for every test type you want to run.

Arguments

- *test_name*

  A string that specifies the type of test to run.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The valid values for *test_name* and a description of each test type follow:

Value	Description
bscan_reg	Ensures the boundary scan data register responds properly to the TAP state machine, and that the boundary scan register can be loaded. This algorithm uses the SAMPLE instruction, shifting data through the boundary scan chain only. Testing whether the boundary scan chain can actually capture and update data to the pins is performed as part of the input and output algorithms, and is therefore not included as part of the bscan_reg algorithm.
bypass_reg	Ensures that the data register state machine is functioning properly and that the bypass register can be loaded. This algorithm scans through the bypass register each time, comparing the bypass register capture value to 0. The algorithm also steps through all possible state transitions associated with a data register scan operation. If the TAP does not track states of the FSM (finite state machine) properly, the 11110 pattern does not appear on TDO in subsequent data register scan operations. The algorithm performs a final check of the bypass operation codes by scanning a 11111 pattern through the bypass register using every instruction opcode documented in the BSDL file as selecting the Bypass register.
clamp	This test verifies the CLAMP instruction. It consists of creating various clamp states and verifying that the output and the bidirectional pins are in the correct states. The clamp test exercises each boundary-scan cell with each pin driven in the high and the low state. In addition, the clamp test performs a scan-through test of the bypass register. The execution is conditional on the existence of the TAP CLAMP instruction code.
controlr	This test verifies that all controlr cells are reset to their disabled state after a TAP controller reset. Execution is conditional upon the presence of at least one controlr cell.
disabled_outputs	This test disables all pads in the EXTEST mode by scanning a deactivating value in all their enable cells. Tests the output for high impedance. Execution is conditional on the presence of at least one control cell or one open drain/source pad.
dot6_ac_00	This test measures AC parameters Vhyst_edge and Thyst on contacted 1149.6 input pins by applying a valid logic 0 to the input pins and expecting to capture a logic 0. This test applies AC waveforms to contacted inputs and uses the EXTEST_PULSE code.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Value	Description
dot6_ac_01	This test measures AC parameters Vhyst_edge and Thyst on contacted 1149.6 input pins by applying an invalid logic 0 to the input pins and expecting to capture a logic 1. This test applies AC waveforms to contacted inputs and uses the EXTEST_PULSE code. This test is meant for manufacturing patterns and never passes simulation because it assumes that the test receiver reacts correctly to an invalid 0 or 1.
dot6_ac_10	This test measures AC parameters Vhyst_edge and Thyst on contacted 1149.6 input pins by applying an invalid logic 1 to the input pins and expecting to capture a logic 0. This test applies AC waveforms to contacted inputs and uses the EXTEST_PULSE code. This test is meant for manufacturing patterns and never passes simulation because it assumes that the test receiver reacts correctly to an invalid 0 or 1.
dot6_ac_11	This test measures AC parameters Vhyst_edge and Thyst on contacted 1149.6 input pins by applying a valid logic 1 to the input pins and expecting to capture a logic 1. This test applies AC waveforms to contacted inputs and uses the EXTEST_PULSE code.
dot6_ac_input	This test verifies the AC operation of the 1149.6 receivers. It performs the following tasks: Applies a checkerboard pattern to contacted AC input/bidirectional pins. Disables all output/bidirectional pads. Applies a valid AC waveform to contacted input and inout pins. Tests the “no transition” detection capability, if present, by applying two consecutive test patterns while keeping the AC pin level constant. If the pad contains either a Low-Pass filter or an embedded High-Pass filter, the absence of transition in the second pattern makes its corresponding boundary-scan cell capture its default value. Uses both EXTEST_PULSE and EXTEST_TRAIN codes. **Note:** inputs with loopbacks are not tested with this test, but rather in the dot6_ac_output test.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Value	Description
dot6_ac_output	This test verifies the AC operation of the 1149.6 transmitters and loopbacks. It performs the following tasks: Enables all AC output pads. Ignores and turns off all non-AC pads, if possible. Uses both AC and DC loopbacks if present. Uses both EXTEST_PULSE and EXTEST_TRAIN codes. Alternatively disables and enables ACSelect cells. Runs the test on one Enable Group at a time. Strobes AC waveforms on all AC contacted outputs.
dot6_ac_select_cells	This test verifies that the netlist properly implements the AC grouping described in the BSDL. Only one ACSelect cell is enabled at a time, and its action is verified by checking that all AC output pins in its fanout properly toggle during the RunTestIdle state when the EXTEST_PULSE instruction is loaded. This test is necessary only during netlist verification in the design flow. In manufacturing, the dot6_ac_output test ensures that all AC select cells are properly operating by turning them all on or off at once.
dot6_dc_00	This test measures DC parameters Vthreshold and Vhyst_level on contacted 1149.6 input pins by applying a valid logic 0 to the input pins and expecting to capture a logic 0. All AC pad drivers are disabled for this test.
dot6_dc_01	This test measures DC parameters Vthreshold and Vhyst_level on contacted 1149.6 input pins by applying an invalid logic 0 to the input pins and expecting to capture a logic 1. All AC pad drivers are disabled for this test. This test is meant for manufacturing patterns and never passes simulation because it assumes that the test receiver reacts correctly to an invalid 0 or 1.
dot6_dc_10	This test measures DC parameters Vthreshold and Vhyst_level on contacted 1149.6 input pins by applying an invalid logic 1 to the input pins and expecting to capture a logic 0. All AC pad drivers are disabled for this test. This test is meant for manufacturing patterns and never passes simulation because it assumes that the test receiver reacts correctly to an invalid 0 or 1.
dot6_dc_11	This test measures DC parameters Vthreshold and Vhyst_level on contacted 1149.6 input pins by applying a valid logic 1 to the input pins and expecting to capture a logic 1. All AC pad drivers are disabled for this test.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Value	Description
dot6_dc_input	This 1149.6 test verifies the behavior of all 1149.6 AC input pins when operating in EXTEST mode. Just as the non-AC (in other words, DC) pins “input” test, this pattern applies zero and one logic values at the AC pins and checks that their 1149.6 test-receiver's associated bscan cells capture the same value. In addition, whenever the target tester supports it, the test also verifies that all AC pins can reliably detect invalid input voltage levels sitting in between the “zero” and “one” threshold voltages.
dot6_dc_output	This 1149.6 test is equivalent to the existing output IO test for normal DC pads. Just as in the output test, the dot6_dc_output test applies checkerboard values, activates output cells one enable group at a time, and uses load board loopbacks as well as IO internal loopbacks. In addition, this test also covers 1149.6-specific features such as coupling capacitors discharge time, ACSelect cells, and output inversion during the RunTestIdle state of the TAP.
highz	This test verifies the operation of the HIGHZ instruction code. The pattern loads the HIGHZ code into the TAP, driving all output and bidirectional pins into high-impedance states. Note that this test actually executes the following three tests: disabled_outputs, highz_mode, and controlr.
highz_mode	This test disables all pads via the TAP HIGHZ instruction code. It tests all outputs for high impedance. The execution is conditional to the existence of a TAP HIGHZ instruction code.
id_reg	Ensures that the ID register is responding properly to the TAP data register state machine. This algorithm compares the capture values of the device ID and status registers whenever a data register scan operation is performed. The algorithm steps through all possible state transitions associated with a data register scan operation. The execution is conditional to the existence of a TAP IDCODE instruction code.
iih	Presets input pins so that the tester can drive the pins to their high values and measure the current of these pins. The tester stops the test to measure current, then resumes the test. This same pattern can be used for other parametric tests such as VIH.
iil	Presets input pins so that the tester can drive the pins to their low values and measure the current of these pins. The tester stops the test to measure current, then resumes the test. This same pattern can be used for other parametric tests such as VIL.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Value	Description
input	This test verifies that the boundary-scan cells perform a proper capture of input data using standard IEEE-1149.1 timing. It tests all input, observe-only, and clock boundary-scan cells when the TAP instruction register is loaded with the EXTEST instruction code. In addition, it also tests the input direction of all bidirectional cells. The test turns off all pad drivers, including the bidirectional ones, and applies alternate EVEN and ODD checkerboard patterns to all input and bidirectional contacted pins that have tester control capabilities. Cells are pre-loaded with their pin state's opposite value. All non-X cell captured values are compared at TDO including those resulting from the following: Cells capturing themselves or a constant value in EXTEST mode. When a pin is tied to a constant value by the load board. The presence of an internal or external pull-up or pull-down resistor at the output of an uncontrollable bidirectional pin.
inst_reg	This test verifies that the instruction register can be loaded. The two least significant bits are compared to "01" on every instruction register scan operation. This algorithm steps through all possible state transitions associated with an instruction register scan operation. If the TAP does not track states of the FSM (finite state machine) properly, the standard 01 bits of the instruction register do not appear in a subsequent instruction register operation.

leakage_non_contacted	This test drives a value to an inout port and observes the value after a specified number of cycles.
	You can only use the leakage_non_contact test if there are non-contacted inout ports. Describe these non-contacted inout ports in the PatternsSpecification/LoadBoardInfo/TesterInterface wrapper with port properties of control:none and observation:none.
	Note: The Verilog testbench passes only if you specify the pad decay time in the SimulationOptions/NonContactedTestOptions/PadDecayTimes wrapper with a time greater than that specified in the NonContactedPinOptions/LeakageOptions/number_of_cycles property. You can calculate the number of cycles and the pad decay time using the following formula and requirement:
	`cycles = number_of_cycles + 2.5`
	`pad_decay_time > cycles * (tester_period * tck_ratio)`

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Value	Description

tristate_enable_non_contacted	This test verifies if the force_disable and local enable signals can disable the pad driver of a bidirectional non-contacted pad. The RunTest/ NonContactedPinOptions/TriStateEnableOptions wrapper defines the types of tests to run (global or local) and their parameters. You can only use this wrapper with the tristate_enable_non_contacted test.
	Define the pad decay times in the SimulationOptions/ NonContactedTestOptions/PadDecayTimes wrapper for the testbench. You can calculate the number of cycles and the pad decay time using the following formula and requirement:
	`cycles = number_of_rti_cycles + 1.5`
	`pad_decay_time > cycles * (tester_period * tck_ratio)`

output	This test verifies that the boundary-scan cells perform a proper capture of output data using standard IEEE 1149.1 timing. It also checks the preload functionality of the SAMPLE/ PRELOAD instruction. In addition, it tests output and bidirectional boundary-scan cells when the TAP instruction register is loaded with the EXTEST instruction. Output and bidirectional pad drivers are turned on sequentially on a per group basis so as not to exceed the values specified with the active_control_cells property. All enabled output and bidirectional cells are loaded alternatively with EVEN and ODD checkerboard patterns. The output and bidirectional pins are strobed if they are both contacted and observable. All non-X values captured by the boundary-scan cells are compared at TDO, including the ones from the following: Cells capturing themselves or a constant value in EXTEST mode. Enabled bidirectional boundary-scan cells that capture the pin state in EXTEST mode. Disabled bidirectional boundary-scan cells that have an internal or external pull-up/pull-down resistor. Input pins driven by a load board loopback from an output or bidirectional pin. Input cell captured value for all enabled bidirectional pads controlled by the combination of one output and one input or sample boundary-scan cell.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Value	Description
output_clamp	This test runs the output test with an added short clamp test. During the output test, after each load of the boundary-scan register and each strobing of the enabled output and input pins, the following is performed: The TAP instruction register is reprogrammed with the CLAMP instruction code. The test verifies whether the BYPASS register is selected. The output pins are strobed again to verify that they are to retain the initial output state value in CLAMP mode. The TAP instruction register is then reprogrammed with the EXTEST instruction, and the output test resumes from there.
sample	This test verifies that the boundary-scan cells perform a proper capture of input data, like the input test, with the exception of the bidirectional pins. This algorithm uses the SAMPLE instruction code instead of the EXTEST instruction code which means the pad drivers of bidirectional pins cannot be turned off.
test_logic_reset	This test verifies that the TAP is being reset properly. Note that the boundary scan chain contains safe data values, and that either the *EXTEST* or *SAMPLE* instruction is active.
voh	Presets output pins so that the tester can drive the pins to their high values and measure the voltage of these pins. The tester stops the test to measure voltage, then resumes the test. This same pattern can be used for other parametric tests such as IOH.
vol	Presets output pins so that the tester can drive the pins to their low values and measure the voltage of these pins. The tester stops the test to measure voltage, then resumes the test. This same pattern can be used for other parametric tests such as IOL.

- AdvancedOptions/checker_board : <u>on</u> | off ;

  A property that enables or disables applying checkerboard vectors. Valid settings are as follows:

on	Enables checkerboard vectors which alternate logic zeroes and ones from one pin to the next, following the order of the boundary-scan register. This increases the chances of detecting short-circuits between two adjacent pins. Default IO test vectors are checkerboard.
off	Disables checkerboard, therefore applying boundary-scan load vectors made of all zeroes or all ones. This leads to easier debugging and simplifies usability.

  This property affects the following tests: input, output, clamp, sample, dot6_ac_input, dot6_dc_input, dot6_ac_output, dot6_dc_output, and dot6_ac_select_cells.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If the value is set to off, individual boundary-scan load vectors are still named EVEN and ODD in the output testbench. However, EVEN means logic 0 for all IO pins and ODD means logic 1 for all IO pins.

- NonContactedPinOptions/LeakageOptions/logic_level : both | iih | iil ;

  A property that specifies the logic levels to be tested with the leakage_non_contacted test. This property can only be used with the leakage_non_contacted test. Specifying iih tests the leakage current with a value of 1 driven on the inout port. Specifying iil tests the leakage current with a value of 0 driven on the inout port. The default value of "both" with test the leakage current with both a 0 and 1 value driven.

- NonContactedPinOptions/LeakageOptions/number_of_cycles : *int* ;

  A property that specifies the number of wait cycles between driving the test value and checking the value on a non-contacted inout port. This property can only be used with the leakage_non_contacted test. The default value is "0". The specified wait time should be smaller than the pad decay time specified for the ports that are tested. The pad decay time is specified in the PatternsSpecification/Patterns/SimulationOptions/ NonContactedTestOptions/PadDecayTimes wrapper.

- NonContactedPinOptions/TriStateEnableOptions/test_path : both | local | global ;

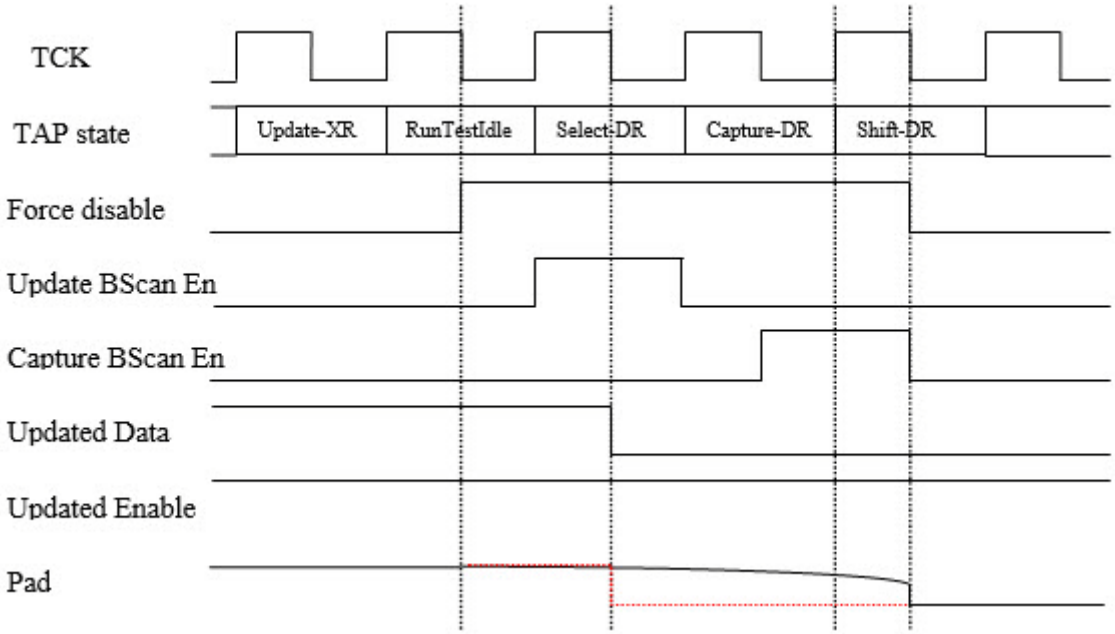
  A property that specifies whether the global or the local test is created. By default, both tests are created.

  The global test charges the pads with the value specified in the TriStateEnableOptions/ logic_level property, and loads the data bscan cell with the opposite value. The update element is not strobed, so the value is not applied to the pad, as shown in Figure 13-128 below. The force_disable signal is asserted, tri-stating the pad, and remains active until the pad state is captured. The updated data, set to the opposite state, is loaded in the SelectDR TAP state, which is done 1.5 cycles before the capture occurs. If the force_disable signal is not able to disable the pad driver because of a faulty circuit, the pad is driven with the opposite value and that value is captured, as shown with the red dotted line.

  The timing in Figure 13-128 reflects the default setting of "1" for TriStateEnableOptions/ number_of_rti_cycles. By using larger settings, the number of cycles between turning off the pad driver and capturing the value can increased. This can be used to test the amount of time the pad can hold a charged value. The pad decay time, or the time a pad can hold a value, is defined in the PatternsSpecification/Patterns/SimulationOptions/ NonContactedTestOptions/PadDecayTimes wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-128. Timing Diagram of the Global tristate_enable_non_contacted Test

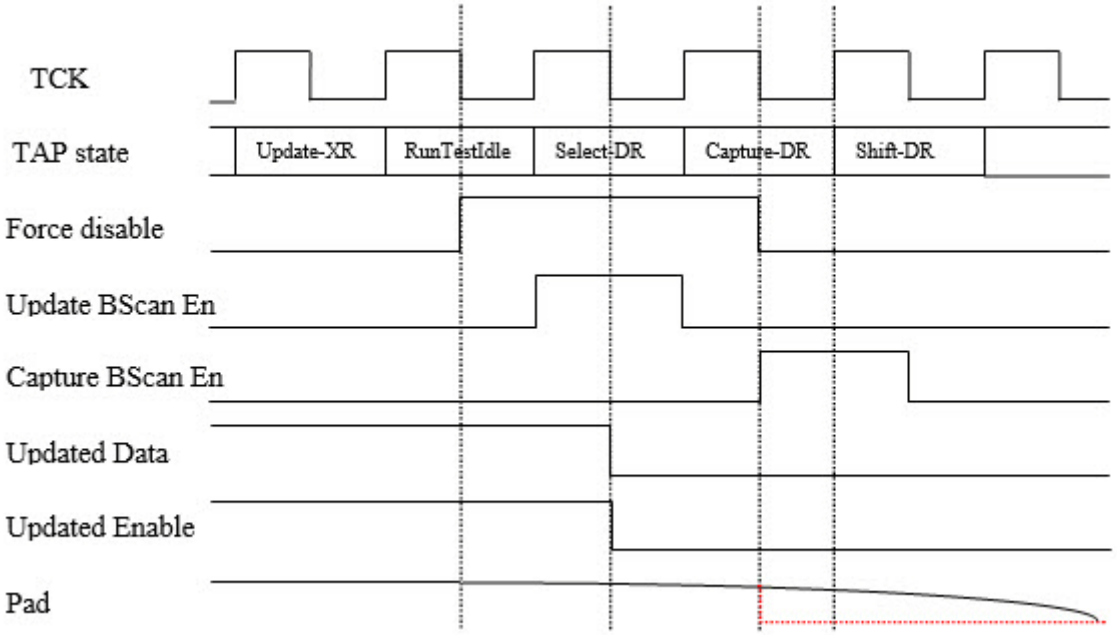


The local test charges the pads with the value specified in the TriStateEnableOptions/logic_level property, and loads the data bscan cell with the opposite value. Additionally, the enable cell is loaded with the pad disable value. The update element is not strobed, so the value is not applied to the pad, as shown in Figure 13-129 below. The force_disable signal is asserted, tri-stating the pad as is done with the global test. The updated data (opposite) and enable (disable value), is loaded in the SelectDR TAP state, which is done 1.5 cycles before the capture occurs. The force_disable is released in the CaptureDR TAP state, leaving only the enable cell, loaded with the pad disable value, to tri-state the pad. If the enable cell is not

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

able to disable the pad driver because of a faulty circuit, the pad is driven with the opposite value and that value is captured, as shown with the red dotted line in the figure:

Figure 13-129. Timing Diagram of the Local tristate_enable_non_contacted Test



- NonContactedPinOptions/TriStateEnableOptions/logic_level : both | high | low ;

  A property that specifies the logic level that is used to pre-charge the pads for local and global tristate_enable_non_contacted tests. By default, tests are created for both high and low levels.

- NonContactedPinOptions/TriStateEnableOptions/number_of_rti_cycles : *integer* ;

  A property that specifies the number of cycles the tristate_enable_non_contacted global test remains in the RunTestIdle TAP state. This determines how long the pad remains undriven before the value is captured 1.5 cycles later. The default value is 1, or one cycle.

- CellCompares/*cell_number* : 0 | 1 | X ;

  A property that enables you to override one or more boundary-scan cells' expected captured values for all scan operations performed during the test. The cell_number is a valid boundary-scan cell number as listed in the BOUNDARY_REGISTER Attribute section of the BSDL file of the design. A setting of X instructs Tessent Shell to ignore the captured and shifted out value of this boundary-scan cell, and a setting of 0 or 1 instructs it to compare the captured and shifted out value of this boundary-scan cell to this logical value.

- CellSettings/*cell_number* : 0 | 1 ;

  A property that enables you to override the value to be shifted into one or more specific boundary-scan cells for all scan operations performed during the test. The cell_number is a valid boundary-scan cell number as listed in the BOUNDARY_REGISTER Attribute

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

section of the BSDL file of the design. A setting of 0 instructs Tessent Shell to shift in and update the logic 0 value in the specified boundary-scan cell, and a setting of 1 instructs Tessent Shell to shift in and update the logic 1 value in the specified boundary-scan cell.

For the specified cells, the CellSettings wrapper overrides the value for all shift-in and update operations that is performed during execution of the test specified by test_name of the RunTest wrapper. Comparisons of captured pin values are not adjusted accordingly and has to be comprehended with the appropriate CellCompares and PinComparesOverride settings. Typically, this wrapper is used to force a static value on an output or bidirectional pin of the DUT.

- MaskedPins/*pin_name* : masked | normal ;

  A property that specifies that one or more pins are masked during the IO tests. Any failure related to these pins or to associated boundary-scan cells is not reported during execution of the test. The pin_name is a valid port name on the current design.

- PinSettingsOverride/*pin_name* : 0 | 1 | Z ;

  A property that specifies a constant value for one or more input or bidirectional pins. The pin_name is a valid name of a top-level port (primary input or bidirectional pin). A value of 0, 1, or Z indicates that a logic 0, logic 1, or Z value, respectively, is forced on the pin. This property setting overrides the pin value that the test normally applies.

- PinComparesOverride/*pin_name* : 0 | 1 | X | Z ;

  A property that specifies for Tessent Shell to compare one or more output or bidirectional pins to a given value. The pin_name is a valid name of a top-level port (primary output or bidirectional pin). A value of 0 or 1 indicates that a logic 0 or logic 1 value, respectively, is compared on the primary pin. A value of X indicates that no value is compared on the primary pin. This property setting overrides the compare value that the test normally applies.

  It is recommended to use this wrapper to list all output2 pins that cannot be tri-stated to a compare value of X for highz, highz_mode, disabled_outputs and controlr tests.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example shows a configuration that tests the TAP controller hosting the boundary scan, and a separate set of patterns to test the boundary scan:

```
PatternsSpecification(CHIP,signoff) {
   Patterns(TAP_test) {
      TestStep(TS1) {
         BoundaryScan {
            RunTest(test_logic_reset){}
            RunTest(inst_reg) {}
            RunTest(id_Reg) {}
            RunTest(bypass_reg) {}
         }
      }
   }
   Patterns(bscan_test) {
      TestStep(TS1) {
         BoundaryScan {
            RunTest(bscan_reg){}
            RunTest(input) {
               AdvancedOptions {
                  checker_board: off ;
               }
            }
            RunTest(input) {
               AdvancedOptions {
                  checker_board: on ;
               }
            }
            RunTest(output_clamp) {}
            RunTest(sample) {}
            RunTest(disabled_outputs) {
            RunTest(highz_mode) {
               PinComparesOverride {
                  OUT5 : X ;
                  OUT6 : X ;
               }
            }
         }
      }
   }
}
```

Related Topics

TestStep

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

LogicBist

Specifies the LogicBIST tests to be generated.

Usage

```
PatternsSpecification(design_name,usage) {
  Patterns(patterns_name) {
    TestStep(test_step_name) {
      LogicBist {
        CoreInstance(icl_instance_name) {
          run_mode                : hw_default | run_time_prog |
                                    burn_in | self_test;
          burn_in_time            : tvalue ;
          mode_name               : mode_name ;
          pattern_id              : pattern_id ;
          misr_compares           : int ;
          begin_pattern           : int | last_pattern ;
          end_pattern             : int | last_pattern ;
          warmup_pattern_count    : int | auto ;
          shift_clock_select      : shift_clock_src | ltest_clock | tck ;
          cooldown_pattern_count  : int; // default: 0
          post_scan_activity      : on | off;
          execution_phase         : enum ;
                            // legal: setup_run_check | setup_run |
                                      check | setup | run | run_check
          required_setup_registers : enum ;
                            // legal: none | all | chain_mask | prpg |
                                      low_power_static | low_power_mask ;
          DiagnosisOptions {
            extract_flop_data  : on | off ;
          }
          SelfTestOptions {
            shift_length       : int ; // Default: 2
            capture_length     : int ; // Default: 1
            low_power          : int | off ;
            chain_mask   : {edt_instance_name chain_id} ... ; //repeatable
          }
        }
      }
    }
  }
}
```

Description

The LogicBIST wrapper specifies the LogicBIST tests to be generated.

As shown in the syntax above, the LogicBIST specification consists of one or more CoreInstance wrappers that contain the properties that determine how the Hybrid TK/LBIST core is to be run.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- CoreInstance(*icl_instance_name*)

  A required property that specifies the ICL instance to the core containing the Hybrid TK/LBIST controller. If the controller is in the root module, then you specify ".".

- run_mode : hw_default | run_time_prog | burn_in | self_test;

  A property that specifies whether the Hybrid TK/LBIST controller runs with the default settings or with scanned-in settings. When set to run_time_prog, the default, the controller is scanned through the IJTAG network to set up the controller according to the properties specified in the CoreInstance wrapper. When set to hw_default, the controller uses hard-coded values that were determined during the DFT process. When set to burn_in, the controller runs continuously for the amount of time specified by the burn_in_time property. When set to self_test, the controller runs in the self-test mode with settings from the SelfTestOptionsWrapper. The following other options are also used in the self-test mode:

  - begin_pattern
  - end_pattern
  - warmup_pattern_count
  - shift_clock_reset

  The following options are not allowed in the self-test mode and should not be specified when the run mode is set to self_test:

  - mode_name
  - pattern_id
  - misr_compares
  - execution_phase

- burn_in_time : *tvalue* ;

  An optional switch and time-delay value pair that specifies how long the LogicBIST wafer-level burn-in test should run. You can specify the time values using the following units. The default is 1s.

  - s - seconds
  - ms - milliseconds
  - us - microseconds
  - ns - nanoseconds
  - ps - picoseconds

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- mode_name : *mode_name* ;

  A property that selects which LogicBIST mode should be used from the logic_test_cores directory within the TSDB. During fault simulation, you can change the mode by using the set_current_mode command. If the TSDB only contains one mode, the tool uses that mode. If the TSDB contains more than one mode, you must specify which mode to use.

- pattern_id : *pattern_id* ;

  An optional property that specifies a pattern ID string. Use this option when you have multiple patDB files within the same mode directory within the TSDB. (See write_tsdb_data -pattern_id for more information.) This option allows you to test the same core in the same pattern with different pattern IDs. The tool ensures that the correct patDB is used for each TestStep/LogicBist/CoreInstance.

  During process_patterns_specification, the tool uses the specified pattern ID to open the correct patDB. If there are multiple patDBs and you do not specify a pattern ID, the tool issues an error.

- misr_compares : *int* ;

  A property that enables you to specify the number of times the MISR is compared during the LogicBIST run. The number of MISR compares cannot exceed the number of patterns being run, as specified by the begin_pattern and end_pattern properties. If DiagnosisOptions/ extract_flop_data is on, the misr_compares property specifies the number of times the contents of the scan chains are scanned. The default is 1.

- begin_pattern :*int* | last_pattern ;

  A property that specifies the first LogicBIST pattern to apply. The default is 0.

- end_pattern : *int* | last_pattern ;

  A property that specifies the last LogicBIST pattern to apply. The default is last_pattern.

- warmup_pattern_count : *int* | auto ;

  A property that specifies the first *N* patterns of a LogicBIST pattern to be masked in order to allow the voltage in the chip to stabilize. When a warm-up value is specified, the PRPG is seeded with pattern *begin_pattern* - *N* and the MISR holds the begin_pattern seed until the warm-up patterns are completed. The default is 0.

  The “auto” value resolves to the number of warm-up patterns specified during fault simulation, or to 0 if the number of warm-up patterns was not specified during fault simulation.

- cooldown_pattern_count : *int*;

  A property that specifies the last N patterns of a LogicBIST pattern to be masked in order to allow the voltage in the chip to stabilize. When you specify a cool-down value, the LBIST controller continues to run for N additional patterns with the MISR holding its value until the controller completes running. The default is 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- post_scan_activity : on | off;

  A property that specifies that after the end_pattern has been applied, the controller continues running, and the MISR holds, until the controller MISR is compared. This property can only be enabled when execution_phase is set to setup or setup_run. The default is off.

- execution_phase : *enum* ;

  An optional property that specifies the execution phase of the TestStep within the Patterns wrapper. The default value is setup_run_check. Other legal values are setup_run, check, setup, run, and run_check. When splitting execution into two phases, use setup_run for the first pattern set, and check for the second. The setup_run and check phases are typically used in InSystemTest context. However, for manufacturing test, the patterns can already map the failing cycle on the tester to an ICL register, so this functionality is not meaningful.

  Note

  This property only applies for Patterns when run_mode is set to run_time_prog.

  The LBIST PDL configures the phase of the test to be executed. An LBIST test has three phases:

  - setting up the test by using short setup or long setup, depending on the required initialization
  - running the LBIST scan patterns using stimuli from PRPG and responses captured in the MISR
  - checking the MISR signature

  Using an iProc argument, you can split this sequence into two phases: setup_run and check.

  For more information about using this property, see "Example 4" on page 4605.

- required_setup_registers

  Specifies the list of LBIST setup registers that the test patterns always target. Use this option, for example, to estimate the maximum expected InSystemTest DMA program size of your LBIST implementation. The tool increases the total size of the resulting patterns by considering the worst-case configuration.

  Registers specified with this property are always present in the test patterns, even if they could be optimized out by the tool. When the tool retargets patterns to the InSystemTest controller, it ignores registers with the default hardware value. This property prevents this for the specified registers.

  Note

  This property is used only for patterns retargeted to InSystemTest controllers.

- shift_clock_select : shift_clock_src | ltest_clock | tck ;

  A property that specifies the clock input for the Hybrid TK/LBIST controller.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- DiagnosisOptions/extract_flop_data : on | off ;

  A property that enables the diagnostic mode in which the state of the circuit is frozen after the application of the last pattern and is then the contents of each flip-flop that is serially scanned out via the IJTAG network. The default is off.

  Note

  The IjtagRetargetingOptions/physical_block_iapply_partitioning property must be set to "off" if the DiagnosisOptions/extract_flop_data property is set to "on".

- SelfTestOptions/shift_length : *int* ;

  A property that specifies the shift length for the self-test patterns. This property can only be used when run_mode is set to self_test. The default value is 2.

- SelfTestOptions/capture_length : *int* ;

  A property that specifies the capture length for the self-test patterns. This property can only be used when run_mode is set to self_test. The default value is 1. The OCC configuration, whether in fast or slow capture, does not impact self-test because the design scan cells are not used.

- SelfTestOptions/low_power : *int* | off ;

  A property that is used when run_mode is set to self_test to enable or disable the low_power hardware, if available. Enabling is specified with a given threshold percentage, as is done with the "set_power_control -switching_threshold_percentage" command.

- SelfTestOptions/chain_mask : {*edt_instance_name chain_id*} ... ;

  A repeatable property that specifies any chain masks for the self-test patterns, similar to the add_chain_masks command. This property can only be used when run_mode is set to self_test.

  Note

  The edt_instance_name in this property is the Verilog instance name, not the ICL instance name.

Examples

Example 1

The following example shows the PatternsSpecification for a top-level Hybrid TK/LBIST controller in the module CHIP. The single pattern consists of two TestSteps that require the clocks clk100, clk150, and clk200 to be active at the periods shown. The first TestStep runs a total of 8 patterns, beginning with pattern 0. After the last pattern (7) runs, the MISR signature is scanned out and compared. The second TestStep runs only pattern 7, but after the pattern is complete the flip-flop data is scanned out and compared.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
PatternsSpecification (CHIP,signoff) {
  Patterns (LogicBist) {
    ClockPeriods {
      clk100 : 10ns ;
      clk150 : 6.66667ns ;
      clk200 : 5ns ;
    }
    TestStep (serial_load) {
      LogicBist {
        CoreInstance(CHIP) {
          run_mode      : run_time_prog ;
          begin_pattern : 0 ;
          end_pattern   : 7 ;
        }
      }
    }
    TestStep (diagnostic) {
      LogicBist {
        CoreInstance(CHIP) {
          run_mode      : run_time_prog ;
          begin_pattern : 7 ;
          end_pattern   : 7 ;
          DiagnosisOptions {
            extract_flop_data : on ;
          }
        }
      }
    }
  }
}
```

Example 2

The following example runs the Hybrid TK/LBIST controller from pattern 512 to 524. With three misr_compares the pattern is broken up into three runs, where the first runs patterns 512 to 515, the second runs patterns 516 to 519, and the last runs patterns 520 to 524. In addition, because warmup_pattern_count is set to 8, the 3 misr_compares runs are effectively running patterns 504 to 515, 508 to 519, and 512 to 524, but those eight warm-up patterns do not contribute to the calculated MISR signatures.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
PatternsSpecification(top,gate,signoff) {
  Patterns(LogicBist_m8051_1) {
    ClockPeriods {
      refclk : 10.00ns ;
    }
    TestStep(serial_load) {
      LogicBist {
        CoreInstance(m8051_wrap_inst_m8051_inst1) {
          run_mode : run_time_prog ;
          begin_pattern : 512 ;
          end_pattern : 524 ;
          misr_compares : 3 ;
          warmup_pattern_count : 8 ;
        }
      }
    }
  }
}
```

Example 3

The following example runs the Hybrid TK/LBIST controller in wafer-level burn-in mode for 24 hours (86400 seconds).

```
PatternsSpecification(top,gate,signoff) {
  Patterns(LogicBist_m8051_1) {
    ClockPeriods {
      refclk : 10.00ns ;
    }
    TestStep(serial_load) {
      LogicBist {
        CoreInstance(m8051_wrap_inst_m8051_inst1) {
          run_mode : burn_in ;
          burn_in_time : 86.4e+3s;
        }
      }
    }
  }
}
```

Example 4

The following example shows the LogicBist patterns wrapper generated on a design with usage “manufacturing” for two physical blocks: m8051 and piccpu: Each block has an LBIST controller.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Patterns(LogicBist) {
   TestStep(serial_load_0) {
     LogicBist {
       CoreInstance(m8051) {
         run_mode : run_time_prog;
         warmup_pattern_count : 16;
         begin_pattern : 0;
         end_pattern : last_pattern;
       }
       CoreInstance(piccpu) {
         run_mode : run_time_prog;
         warmup_pattern_count : 16;
         begin_pattern : 0;
         end_pattern : last_pattern;
       }
     }
   }
 }
```

The following example shows the previous example modified to split the execution phases into setup_run and check separately. It also includes an MBIST pattern and InSystemTest wrapper.

```
PatternsSpecification(top,gate,signoff) {
    Patterns(LogicBist_piccpu_m8051_merged) {
      TestStep(LogicBist_piccpu_m8051_setup_run) {
        LogicBist {
          CoreInstance(piccpu) {
            run_mode : run_time_prog;
            execution_phase: setup_run;
          }
          CoreInstance(m8051) {
            run_mode : run_time_prog;
            execution_phase: setup_run;
          }
        }
      }
      TestStep(LogicBist_piccpu_check) {
        LogicBist {
          CoreInstance(piccpu) {
            run_mode : run_time_prog;
            execution_phase: check;
          }
        }
      }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
        TestStep(LogicBist_m8051_check) {
          LogicBist {
            CoreInstance(m8051) {
              run_mode : run_time_prog;
              execution_phase: check;
            }
          }
        }
      }
      Patterns(MemoryBist_P1) {
        ...
      }
      InSystemTest {
        Controller(top_gate_tessent_in_system_test_c0_inst) {
          TestProgramGroup(0, 2) {
            pattern: LogicBist_piccpu_m8051_merged;
          }
          TestProgram(3) {
            pattern: MemoryBist_P1;
          }
        }
      }
    }
```

The Verilog files generated for the previous example are as follows:

```
tsdb_outdir/patterns/top_gate.patterns_signoff/
     InSystemTest_0_2_LogicBist_piccpu_m8051_merged.v
     InSystemTest_3_MemoryBist_P1.v
```

Example 5

The following example runs the Hybrid TK/LBIST controller in self-test mode for 10 patterns and 5 warmup patterns, with a shift_length of 10, a capture_length of 2, a low_power switching threshold percentage set to 15%, and a chain_mask specified on the first chain for the given EDT instance.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
PatternsSpecification(piccpu,gate,signoff) {
  Patterns(LogicBist_piccpu) {
    ClockPeriods {
      clk : 10.00ns;
    }
    TestStep(self_test) {
      LogicBist {
        CoreInstance(piccpu) {
          run_mode : self_test;
          warmup_pattern_count : 5;
          begin_pattern : 0;
          end_pattern : 9;
          SelfTestOptions {
            shift_length : 10;
            capture_length : 2;
            low_power : 15;
            chain_mask : piccpu_rtl_tessent_edt_lbist_c0_inst, 1;
          }
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

MemoryBisr

Controls BISR circuitry in your design.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(pattern_name) {
    TestStep(name) {
      MemoryBisr {
        run_mode  : bisr_chain_access | autonomous | fuse_box_access ;
        Controller() {
        }
      }
    }
  }
}
```

Description

The MemoryBisr wrapper is used for controlling BISR circuitry in your design.

At the block or sub-block design level, the Controller wrapper and run_mode property are used to verify the operation of BISR chains even though the chains may not yet be connected to a BISR controller.

At the chip design level, all parameters are used to verify the operation of BISR controller itself and its connections to the BISR chains and fuse box.

Arguments

- run_mode : bisr_chain_access | autonomous | fuse_box_access ;

  A property used for BISR verification at the chip top, block, or sub-block level specifying the mode of operation of the BISR circuitry. The property must be specified because there is no default value for it. At the block and sub-block level, only the bisr_chain_access mode is valid. All modes are valid at the chip top level when the chains are connected to a BISR controller. The following table describes the three available modes.

bisr_chain_access	Enables access to BISR chains. At the block/sub-block level, BISR chains are typically directly connected to top-level ports. At the chip top level, the BISR controller enables access to the BISR chains using the TDI and TDO top-level TAP ports. Several options are available to transfer repair information from BIRA modules to BISR registers, and serially load/unload user-defined values in selected BISR chains. These options are explained in the BisrChainAccessOptions wrapper description.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

autonomous	In this mode, the controller operation does not require any data patterns from the tester. Several autonomous options are available to compress/decompress repair information, program/read fuses, and manipulate BISR chains. These options are explained in the AutonomousOption wrapper description.
fuse_box_access	Specifies that the BISR controller enables access to the fuse box using the TDI and TDO top-level 1149.1 TAP ports. The two valid operations in this mode are *program* and *read*. These are explained in the FuseBoxAccessOptions wrapper description.

Examples

Examples for the various BISR run modes can be found at the end of the BisrChainAccessOptions, AutonomousOption, and AutonomousOption wrapper descriptions.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Controller

Specifies the inclusion of a memory BISR controller into the test patterns.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(pattern_name) {
    TestStep(name) {
      MemoryBisr {
        Controller(icl_instance_name) {
          power_domain_group_labels      : none | pdg_label, ... ;
          test_time_multiplier           : real ; // default: 1
          fuse_box_write_duration        : time ; // default: 7ms
          fuse_box_read_duration         : time ; // default: 200ns
          fuse_box_init_duration         : time ; // default: 1us
          inhibit_buffer_to_fuse_tranfer : on | off ;
          AutonomousOptions  {
          }
          BisrChainAccessOptions  {
          }
          FuseBoxAccessOptions  {
          }
        }
      }
    }
  }
}
```

Description

The Controller wrapper specifies the inclusion of a memory BISR controller into the test patterns and allows specifying properties that are applicable to all run modes selectable in the MemoryBist wrapper such as the selection of power domain groups. Options specific to the selected run mode are part of the corresponding wrapper.

Arguments

- *icl_instance_name*

  Specifies the instance name of a module found in an ICL model. The ICL instance name corresponds to the instance of the module in the design at the time the BISR logic was inserted. The ICL instance name provides a fixed reference to this logic even if the circuit hierarchy might change during the layout process. At the block/sub-block level when the chain are not yet connected to a controller, the icl_instance_name is specified as “.”. and identifies the block/sub-block itself. At the chip top level, it identifies the BISR controller module. The list of available memory BIST controller instances can be found using this command:

  ```
  get_icl_instances -filter
  {tessent_instrument_type==mentor::memory_bisr &&
  tessent_instrument_subtype==controller}
  ```

  The create_patterns_specification command automates the creation of the Controller wrapper for each instance in the design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- power_domain_group_labels : none | *pdg_label*, ... ;

  A property that specifies the power domain groups on which to perform the BISR operations. The default value is an asterisk (*) which is equivalent to enumerating all power domain group labels, including the default group assigned to all memories that were not explicitly assigned a label. This special group is designated with a hyphen (-).

  You can only specify the value of none when running the BISR controller from the IJTAG interface in autonomous power-up emulation run mode. In this configuration, all power domain groups are disabled. The BISR controller performs the eFuse to read buffer transfer and completes the operation without loading any BISR chains.

  Note

  The value of "none" is a reserved string and you cannot specify it as a power domain group label.

  Note the following usage conditions:

If these settings apply:	**Then the power_domain_group_labels setting must be:**
MemoryBisr/run_mode : bisr_chain_access	An asterisk (*) or One or more power domain group label(s)
MemoryBisr/run_mode: autonomous MemoryBisr/Controller/AutonomousOptions/operation : verify_fuse_box or MemoryBisr/Controller/AutonomousOptions/operation : power_up_emulation or MemoryBisr/Controller/AutonomousOptions/operation : clear_bisr_chain	An asterisk (*) or One or more power domain group label(s)
MemoryBisr/Controller/AutonomousOptions/operation : power_up_emulation	none or An asterisk (*) or One or more power domain group label(s)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If these settings apply:	Then the power_domain_group_labels setting must be:
MemoryBisr/run_mode: autonomous MemoryBisr/Controller/AutonomousOptions/operation : self_fuse_box_program	An asterisk (*) or all power domain group labels

- test_time_multiplier : *real* ;

  A property that specifies a multiplier to extend or shorten the memory BISR run time. The final run time is equal to the original test time (as calculated by Tessent Shell) multiplied by the test_time_multiplier factor. The default is 1.

  This property is typically used when the test clock runs slower than expected by BISR. As a result, the memory BISR controller may not have completed its test within the allocated time when polled by the ATE. The test_time_multiplier property thus extends the computed test time; this is usually as a temporary measure (until the clock periods described in the Patterns/ClockPeriods wrapper is used).

- fuse_box_write_duration : *time* ;

  A property that controls the duration of the pause cycle when the BISR controller is writing into the fuse box. This property is rarely used and overrides the value already specified by the AdvancedOptions/write_duration property of the fuse box interface. The fuse box data sheet should provide the fuse write duration delay. The time value is a real number followed by the unit without a space in between. Valid time units are s, ms, us, ns, and ps. The default time unit is ns. The fuse box write duration default is 7ms.

- fuse_box_read_duration : *time* ;

  A property that controls the duration of a read access from the fuse box. The time value is a real number followed by the unit without a space in between. Valid time units are s, ms, us, ns, and ps. The default time unit is ns. The fuse box read duration default is 200ns.

- fuse_box_init_duration : *time* ;

  Specifies the initial delay required prior to accessing the fuse box. The time value is a real number followed by the unit without a space in between. Valid time units are s, ms, us, ns, and ps. The default time unit is ns. The fuse box init duration default is 1 us.

- inhibit_buffer_to_fuse_transfer : on | <u>off</u> ;

  A property that prevents the automatic programming of the fuse values that are stored in a buffer located in the fuse box interface at the end of a SelfFuseBoxProgram or FuseBoxAccess programming test step. If the inhibit_buffer_to_fuse_transfer property is set to “on” inside a self_fuse_box_program or fuse_box_access programming TestStep wrapper, the contents of the fuse box interface buffer is not transferred to the fuse box (specifically, fuse programming is not executed). Setting this property to on enables

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

additional fuse values to be written to the buffer in subsequent test steps before final programming is executed.

Examples

Examples for the various BISR options can be found at the end of the BisrChainAccessOptions, AutonomousOption, and AutonomousOption wrapper descriptions.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

AutonomousOption

Sets options that control the autonomous mode of the BISR controller.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns (pattern_name) {
    TestStep(name) {
      MemoryBisr {
        Controller(instance) {
          AutonomousOptions {
            operation           : mode ;
            enable_bira_capture : on | off ;
            select_read_buffer  : on | off | auto ;
            max_repair_count    : integer | all ;
          }
        }
      }
    }
  }
}
```

Description

The AutonomousOptions wrapper sets options that control the autonomous mode of the BISR controller. In Autonomous mode, the controller operation does not require any data patterns from the tester and the state of the GO and DONE outputs is updated at the end of the operation.

Several autonomous options are available to compress and decompress repair information, program and read fuses, and manipulate BISR chains. The content of the AutonomousOptions wrapper is used when you specify the MemoryBisr/run_mode property as “autonomous”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- operation : *mode* ;

  A property that specifies one of the following operation modes:

calculate_bisr_chain_length	In this mode, the BISR controller determines the length of the selected BISR chains by shifting a single “1” through the chains. The BISR controller counts the number of shifts needed for the “1” to propagate through the chains and initializes the BISR chain length registers with the calculated value. These registers are also initialized using the same method when running the power_up_emulation operation. The fuse box is not read from for this operation, therefore if hard incremental repair is to be performed, a power_up_emulation operation should be done initially to properly initialize the eFuse test insertion flags. The initialization of the BISR chain length registers must be executed before running the following autonomous mode operations: rotate_bisr_chain, self_fuse_box_program, and verify_fuse_box.
clear_bisr_chain	Clears all flip-flops in the BISR chain without calculating the BISR chain length. This enables quick initialization of the memory repair inputs when performing memory BIST verification.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

load_bisr_chain	Enables the BISR registers to capture the repair information contained in the corresponding BIRA registers. This operation is typically used before executing a self_fuse_box_program or when implementing soft repair in a design that contains memories with parallel repair interfaces. **NOTE: If the design contains memories with serial repair interfaces or within Shared Bus modules, this operation does only transfer the Redundancy Analysis information to the BISR registers. The repair information is not transferred to the memories with serial repair interfaces or memories inside Shared Bus modules since there is no BISR chain rotation during this operation. If a soft repair implementation is desired in a design that contains memories with serial repair interfaces or within Shared Bus modules, the rotate_bisr_chain operation with enable_bira_capture:on should be used instead.** During the Manufacturing flow, the values from the Redundancy Analysis modules are captured in the BISR registers using a load_bisr_chain operation, and then compressed and programmed in the fuse box using a self_fuse_box_program operation. Since a BISR chain rotation is performed during the self_fuse_box_program operation, the repair information is transferred to the memories with serial repair interfaces or those inside Shared Bus modules during this flow.
power_up_emulation	Simulates a power-up event. During this mode, the BISR controller first calculates the BISR chain length using the method outlined for calculate_bisr_chain_length and then reads and decompresses the repair information from the fuse box into the selected BISR chain(s). All inputs to the BISR controller are provided by IJTAG in this mode. The functional inputs of the BISR controller (clock, repair enable and PDGenable signals) are not used in that mode and can be in any state.
rotate_bisr_chain	Performs a BISR chain rotation autonomously. This operation is used to shift the contents of the BISR chain inside memories with internal repair registers. When the enable_bira_capture property is set to on, a BIRA-to-BISR transfer is performed prior to the BISR chain rotation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

self_fuse_box_program	When the BISR chain contains the repair information for all memories, the BISR controller rotates the BISR chain, compresses its content, and writes the compressed values to the fuse box. If the GO and DONE signals of the BISR controller are high when the test step completes, the initial BISR chain content is preserved. When multiple power domain groups are present, their BISR chains are processed in a single controller run. The power_up_emulation operation should always be performed before the self_fuse_box_program operation.
verify_fuse_box	Verifies the decompressed content of the fuse box against the content of the BISR chain, one bit at a time. The BISR chain is shifted as the decompressed content of the fuse box is shifted in and compared with the BISR chain output. The BISR controller GO signal goes high at the end of the test if the decompressed data from the fuse box exactly matches the content of the BISR chain.

- enable_bira_capture : on | off ;

  A property that enables BISR registers to capture the repair information contained in the corresponding BIRA registers in the rotate_bisr_chain mode. When set to on, the BIRA value is loaded into the BISR register on the first BISR clock cycle before shifting the BISR chain. When set to off, loading of the BIRA values is disabled.

- select_read_buffer : on | off | auto ;

  This property is only used if the MemoryBisr/memory_repair_loading_method parameter is set to “from_read_buffer”. The select_read_buffer property is used to select between serial or parallel loading of the BISR chains during the power-up emulation automonous mode of the BISR controller. When selecting the read buffer output, the BISR chain initialization is much faster than the serial method since the repair values are driven to the memory repair ports using parallel connections. If enable_bira_capture is enabled the parallel repair values are also captured into the BISR registers for validation purposes. When select_read_buffer is set to “off”, the repair values from the fuse box are shifted serially in the BISR registers.

  The default “auto” value resolves to “on” if the BISR controller does support fast BISR loading and the run_mode for the current test step is “power_up_emulation”. Otherwise, the “auto” value resolves to “off”.

- max_repair_count : *integer* | all ;

  A property that specifies the maximum number of repairs your testbench performs. This property is used only in simulation mode with fault injection. Because you know how many faults you injected and how many repairs this represents, you can specify max_repair_count to reduce the simulation time. Valid integer values are 1 or greater. A setting of “all” means that the runtime is calculated as the time needed to write 50 percent of the fuse box locations, assuming the other 50 percent are zeros. A test_time_multiplier : *real ;* property

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

can be specified with a value varying between 1 and 2 to adjust the test time to adjust the maximum percentage of programmed fuses, although this should rarely be necessary.

Examples

The following example causes the BISR controller to find the repair information for the specified power domain in the fuse box, read it, decompress it and load the result in the BISR chain associated to power domain named “PDG_A”. The BISR chain length is also calculated.

```
TestStep(power_up_emulation_step) {
  MemoryBisr {
    run_mode   : autonomous ;
    Controller(FB_INST1) {
      power_domain_group_labels : PDG_A ;
      AutonomousOptions {
        operation    : power_up_emulation ;
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BisrChainAccessOptions

Specifies the options that control BISR chain operations.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(pattern_name) {
    TestStep(name) {
      MemoryBisr {
        run_mode : bisr_chain_access ;
        Controller(icl_instance_name) {
          BisrChainAccessOptions {
            enable_rotation        : on | off ;
            enable_bira_capture    : on | off ;
            select_read_buffer     : on | off ;
            select_bisr_registers  : internal | external | internal_only ;
            default_read_value     : pattern ;
            default_write_value    : pattern ;
            enable_async_reset     : on | off ;
            BisrRegisterAccessOptions(icl_instance_name) {
              read_value(range)    : pattern ;
              write_value(range)   : pattern ;
            }
          }
        }
      }
    }
  }
}
```

Description

The BisrChainAccessOptions wrapper specifies the options that control BISR chain operations.

This wrapper can be used at all design levels (specifically, top, block, or sub-block level). To use the BisrChainAccessOptions wrapper, you must specify MemoryBisr/run_mode : bisr_chain_access. The bisr_chain_access run mode always performs a shift operation on the entire BISR chain. If you must preserve the contents of the BISR chain, the enable_rotation property should be enabled; otherwise, the shift operation clears the contents of the BISR chain when it completes.

You specify access to individual BISR registers using the BisrRegisterAccessOptions. Any read_value or write_value specified in this wrapper overrides the default_read_value and default_write_value specified in the BisrChainAccessOptions wrapper.

Arguments

- enable_rotation : on | off ;

  A property that enables the rotation of the BISR selected chain(s). This property is typically used in combination with enable_bira_capture : on and select_bisr_registers : external to transfer repair information from BIRA registers to external and internal BISR registers. This property can only be specified when the chains are not yet connected to a controller. When

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

you are at a level where the chains are connected to a controller, rotation of the BISR chains must be performed using the rotate_bisr_chain autonomous mode described in the AutonomousOption wrapper.

- enable_bira_capture : on | off ;

  A property that enables BISR registers to capture the repair information contained in the corresponding BIRA registers. When set to on, the BIRA values are loaded into the BISR registers on the first BISR clock cycle before shifting the BISR chain(s). When set to off, loading of the BIRA values is disabled. When set to on, select_bisr_registers should be set to external. The enable_rotation property can be set to on to enable the transfer of BIRA registers to internal BISR registers, or it can be set to off if the objective is to simply inspect the values contained in the BIRA registers.

- select_read_buffer : on | off ;

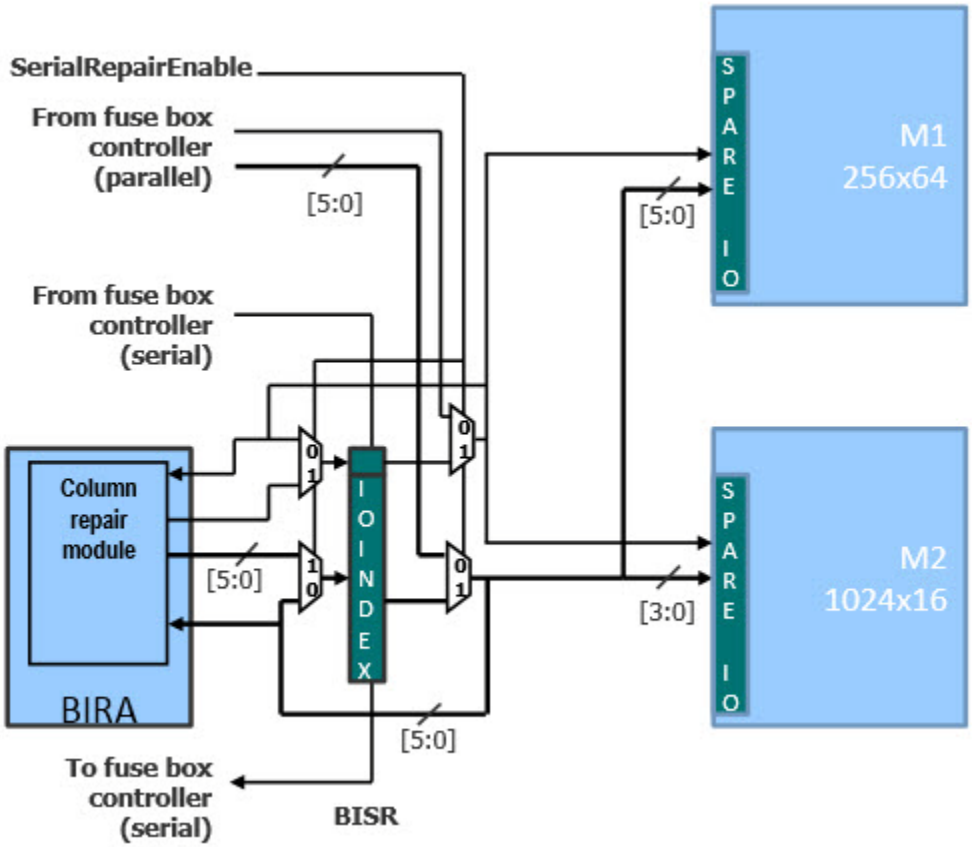
  This property is only used if the MemoryBisr/memory_repair_loading_method parameter is set to “from_read_buffer”.

  When select_read_buffer is set to on, the SerialRepairEnable signal driving the multiplexers shown in Figure 13-130 below is set to “0”. This allows the parallel inputs driven by the fuse box read buffer output to reach the memory repair ports. Additionally, if the enable_bira_capture property is enabled, the parallel repair inputs are also captured inside the BISR registers at the beginning of the shift cycle and scanned out.

  When select_read_buffer is set to off, the SerialRepairEnable signal driving the multiplexers is set to “1”. This allows the repair values from the BISR register outputs to drive the memory repair ports. When enable_bira_capture is enabled in this case, the values from the BIRA engine is captured inside the BISR registers at the beginning of the shift cycle.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-130. Fast BISR Muxing



When select_read_buffer is enabled at a block level, the primary inputs of the block are captured. For block level, an iProc is needed that provides iForcePort statements on the primary inputs of the block to emulate read buffer values. This prevents unknown values from being captured into the BISR registers. The iProc shown below provides an example implementation:

```
iTopProc ReadBuffer{} {
  iForcePort ReadBufferParallelPortName 0
  iApply
}
```

- select_bisr_registers : internal | external | internal_only ;

  A property that specifies whether the internal BISR register of memories with a serial BISR interface, or part of a shared bus, are scanned out during the BISR scan chain access. When set to internal, the property can be used to confirm the values were actually loaded into internal BISR registers in a preceding test step.

  The internal_only value can only be specified with v2020.3 or newer BISR controllers, unless the patterns are generated at the physical block level or when a BISR controller is not present in the design. Otherwise, specifying this value for BISR controllers created with older versions will generate an error. The internal_only value configures the BISR chain the same as when the internal value is specified, however the expected value specified with the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

default_read_value is only applied to the BISR register instances that are associated to memories with a serial repair interface, or for memories that are inside a shared bus assembly. The all_x value is applied for BISR register instances associated to memories with a parallel repair interface and are therefore ignored.

- default_read_value: *pattern* ;

  A property that sets default compare values for all BISR registers of the selected BISR chain(s). The BisrRegisterAccessOptions/read_value property can be used to override the compare pattern for specific BISR registers. By default, the all_x pattern is selected indicating that all bits of all BISR registers are ignored until specified otherwise in the patterns specification. Similarly, the all_one and all_zero patterns indicate that all bits are compared to 1 and 0, respectively. A binary bit pattern can also be specified for comparison.

  The leading_one pattern indicates that the bit of each BISR register closest to its serial output is compared to 1 and all others to 0. The checkerboard and inverse_checkerboard patterns compare alternating 1 and 0 bit patterns, with the pattern beginning at the BISR register bit closest to its serial output. The allocation_bits_on and allocation_bits_off patterns compare the BISR repair register bits associated with memory repair enable bits to a value of 1 or 0 respectively, while ignoring all other bits. These patterns are useful for verifying the correct operation of BISR chains.

  The valid values are:

binary	During an iRead operation, the BISR chain range is compared against the specified bit pattern.
all_x	During an iRead operation, the BISR chain values are not compared.
all_one	Compares all BISR register bits against all 1s during an iRead operation.
all_zero	Compares all BISR register bits against all 0s during an iRead operation.
leading_one	The leading one symbol corresponds to a value of 1 followed by 0s (100...00).
checkerboard	Compares all BISR register bits against a ..0101 pattern during an iRead operation.
inverse_checkerboard	Compares all BISR register bits against a ..1010 pattern during an iRead operation.
allocation_bits_on	Compares the BISR register bits associated with the memory repair enable bits to a value of 1 during an iRead operation.
allocation_bits_off	Compares the BISR register bits associated with the memory repair enable bits to a value of 0 during an iRead operation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- default_write_value : *pattern* ;

  A property that sets default values to be scanned in all BISR registers of the selected BISR chain(s). The BisrRegisterAccessOptions/write_value property can be used to override the write value for specific bits. By default, the all_x pattern is selected indicating that none of the bits need to be set to a particular value. In that case, the last value scanned in during a BISR chain access is used to set each bit. The all_one and all_zero patterns indicate that all bits are set to 1 and 0 respectively. The bits can also be set to a specified binary bit pattern. The all_x pattern is used as the default because BISR chains are usually reset before being exercised for the first time. It is therefore not necessary to explicitly write a value in order to initialize all bits of the BISR chain to 0.

  The leading_one pattern indicates that the bit of each BISR register closest to its serial output is set to 1 and all others to 0. The checkerboard and inverse_checkerboard patterns set alternating 1 and 0 bit patterns, with the pattern beginning at the BISR register bit closest to its serial output. The allocation_bits_on and allocation_bits_off patterns set the BISR repair register bits associated with memory repair enable bits to a value of 1 or 0 respectively, while leaving the remaining bits unmodified. These patterns are useful for verifying the correct operation of BISR chains.

  The valid values are:

binary	During an iWrite operation, the BISR chain range is written with the specified bit pattern.
all_x	During an iWrite operation, the testbench scans in the same values of those previously shifted in this shift register.
all_one	Writes 1’s in all BISR bits during an iWrite operation.
all_zero	Writes 0's in all BISR bits during an iWrite operation.
leading_one	The leading one symbol corresponds to a value of 1 followed by 0s (100..00).
checkerboard	Writes all BISR register bits with a 0101... pattern during an iWrite operation.
inverse_checkerboard	Writes all BISR register bits with a 1010... pattern during an iWrite operation.
allocation_bits_on	Writes the BISR register bits associated with the memory repair enable bits to a value of 1 during an iWrite operation.
allocation_bits_off	Writes the BISR register bits associated with the memory repair enable bits to a value of 0 during an iWrite operation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- enable_async_reset on | off ;

  A property that when enabled, performs an asynchronous reset of the BISR chain at the beginning of the test step.

  If one of the following conditions is met, a shift through of the BISR chain occurs after the asynchronous reset:

  - enable_bira_capture is on
  - enable_rotation is on
  - default_read_value or default_write_value is different than all_x
  - BisrRegisterAccessOptions wrapper read_value or write_value is different than all_x

  If none of those conditions is met, the pattern only performs an asynchronous reset of the BISR chain and no serial shifting of the BISR chain occurs. This method can be used to initialize the BISR registers before running memory BIST patterns, which may reduce simulation time significantly, especially for very long BISR chains.

- *icl_instance_name*

  Specifies the instance name of one or several BISR registers found in an ICL model of the circuit. The selected BISR registers must be part of the selected BISR chain(s) identified by the power_domain_group_labels property. Multiple BISR registers can be specified using wildcards within the name. The ICL instance names correspond to the instance name of BISR registers in the design at the time the BISR logic was inserted. The ICL instance name provides a fixed reference to this logic even if the circuit hierarchy might change during the layout process. To report the available BISR registers, use the report_bisr_repair_register_icl_instances command.

- BisrRegisterAccessOptions(*icl_instance_name*)/read_value(*range*) : *pattern* ;

  A property that sets an expected value on specific bits of the selected BISR registers. In addition to using this property for verification, it can be used in the manufacturing flow. A typical usage of read_value is when observing the BISR chain content that was loaded using enable_bira_capture : on. In this case, the content of the BISR chain is unknown since it contains a repair solution that is calculated inside the device by the BIRA engine. Using read_value property, the repair data can be extracted from the device and compressed off-chip using the CompressBisrChain script.

  The range entry represents a bit index or range of bit indices of BISR registers within the selected BISR chain(s). Bit 0 is closest to the serial output of the BISR register. Entries for range are:

range	Description
integer	Reads the specified bit index of the BISR chain.
integer:*integer*	Reads the range of bit indices specified by the two integers.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

all_bits	Reads all bits of the BISR registers within the selected chain(s).
max_index:*integer*	max_index is replaced by the most significant bit index value of the BISR registers within the selected chain(s). Reads the range of bit indices between the most significant bit and the index specified by integer.
max_index	Reads the most significant bit of the selected BISR chain(s).

The binary value for pattern is a bit pattern of a determined length—for example 8'b10001101. The bit length defined must match the index length defined by range or an error is generated. The all_zero pattern indicates that all bits of the selected BISR registers are compared to 0. Similarly, the all_one and all_x patterns indicate that selected BISR register bits are compared to 1 and x respectively. The leading_one pattern indicates that the bit within a range of BISR registers closest to its serial output is compared to 1 and all others to 0. Values for pattern are:

binary	During an iRead operation, the BISR chain range is compared against the specified bit pattern. The length of range and pattern must match or an error is generated.
all_x	During an iRead operation, the BISR chain values are not compared.
all_one	Compares the BISR register bits against all 1s during an iRead operation.
all_zero	Compares the BISR register bits against all 0s during an iRead operation.
leading_one	During an iRead operation, compares the BISR register bits against a value of 1 followed by 0s (100...00) starting at the bit within the range closest to the BISR chain serial output.

- BisrRegisterAccessOptions(*icl_instance_name*)/write_value(*range*) : *binary* ;

  A property that sets values to be scanned into specific bits of the selected BISR registers. For memories with a serial BISR interface, both the internal and external BISR registers are set to the same value. In addition to using this property for verification, it can be used in the manufacturing flow in cases where the user knows the repair solution, or uses the CompressBisrChain script, then performs the repair externally and loads the repair values through the TAP. The range entry represents a bit index or range of bit indices of BISR registers within the selected BISR chain(s). Bit 0 is closest to the serial output of the BISR register. Entries for range are:

range	Description

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

integer	Writes the specified bit index of the BISR chain.
integer:*integer*	Writes the range of bit indices specified by the two integers.
all_bits	Writes all bits of the BISR registers within the selected chain(s).
max_index:*integer*	max_index is replaced by the most significant bit index value of the BISR registers within the selected chain(s). Writes the range of bit indices between the most significant bit and the index specified by integer.
max_index	Writes the most significant bit of the selected BISR chain(s).

The binary value for pattern is a bit pattern of a determined length—for example 8'b10001101. The bit length defined must match the length index defined by range or an error is generated. The all_zero and all_one patterns indicate that all bits of the selected BISR registers are written with 0s and 1s respectively. The all_x patterns indicate that the selected BISR register bits are written with the same values that were previously shifted in. The leading_one pattern indicates that the bit within the range of BISR registers closest to its serial output is compared to 1 and all others to 0. Values for patterns are:

binary	During an iWrite operation, the BISR chain range is written with the specified bit pattern. The length of range and pattern must match or an error is generated.
all_x	During an iWrite operation, the testbench scans in the same values of that were previously shifted in this BISR chain range.
all_one	Writes 1s in the BISR chain range during an iWrite operation.
all_zero	Writes 0s in the BISR chain range during an iWrite operation.
leading_one	During an iWrite operation, writes a value of 1 followed by 0s (100...00) starting at the bit within the range closest to the BISR chain serial output.

To use the write_value property, the BisrChainAccessOptions/enable_rotation property must be set to off.

Examples

Example 1

The following example has two TestStep wrappers. Both test steps operate on the BISR chain identified with label PDG_A. The leading_one pattern is loaded in all BISR registers in the first step and read back in the second one.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
TestStep(write_leading_one) {
  MemoryBisr {
    run_mode   : bisr_chain_access ;
    Controller(FB_INST1) {
      power_domain_group_labels : PDG_A ;
      BisrChainAccessOptions {
        default_write_value    : leading_one ;
      }
    }
  }
}
TestStep(read_leading_one) {
  MemoryBisr {
    run_mode   : bisr_chain_access ;
    Controller(FB_INST1) {
      power_domain_group_labels : PDG_A ;
      BisrChainAccessOptions {
        default_read_value    : leading_one ;
      }
    }
  }
}
```

Example 2

The following example has two TestStep wrappers. Both test steps operate on the BISR chain identified with label PDG_A. The first step loads a value of 0 in all bits of the BISR chain except for a portion of the specified BISR register. The second step confirms that all values have been loaded correctly and the BISR chain is functional. These steps could be followed by another one running memory BIST to verify that the repair solution loaded in the BISR register is valid.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
TestStep(write_selected_BISR_reg) {
  MemoryBisr {
    run_mode   : bisr_chain_access ;
    Controller(FB_INST1) {
      power_domain_group_labels : PDG_A ;
      BisrChainAccessOptions {
        default_write_value    : all_zero ;
        BisrRegisterAccessOptions (BISR_reg_inst3) {
         write_value(15:8): 8'b10001101 ;
        }
      }
    }
  }
}
TestStep(read_selected_BISR_reg) {
  MemoryBisr {
    run_mode   : bisr_chain_access ;
    Controller(FB_INST1) {
      power_domain_group_labels : PDG_A ;
      BisrChainAccessOptions {
        default_read_value    : all_zero ;
        BisrRegisterAccessOptions (BISR_reg_inst3) {
         read_value(15:8): 8'b10001101 ;
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

FuseBoxAccessOptions

Specifies the options used to read and write (or program) fuses to the fuse box.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(pattern_name) {
    TestStep(name) {
      MemoryBisr {
        run_mode : fuse_box_access ;
        Controller(instance) {
          FuseBoxAccessOptions {
            operation           : program | read ;
            read_address(range) : 0 | 1 | X ;      //repeatable
            write_address       : range ;           //repeatable
          }
        }
      }
    }
  }
}
```

Description

The FuseBoxAccessOptions wrapper specifies the options used to read and write (program) fuses in the fuse box. Fuse addresses are scanned in using the ScanIn and ScanOut ports of the IJTAG network (which is typically a TAP controller at the chip level). To use the FuseBoxAccessOptions wrapper, you must specify MemoryBisr/run_mode : fuse_box_access.

Arguments

- operation : program | read ;

  A property that specifies the type of access method the BISR controller executes on the fuse box. When set to read, the BISR controller executes a read operation on the fuse box according to the read_address property setting. When set to program, the BISR controller writes a logic 1 value to the address specified with the write_address property.

- read_address(*range*) : 0 | 1 | X ;

  A property that reads the content at the specified fuse box address and compares the content against logic 0 or 1. To disable the compare, set the property to X. The default is to compare all addresses to 0 that correspond to the value of an unprogrammed fuse. The range specifies the address within the fuse box. The range can be a single address, an address range, or a comma-separated list of the two formats. All addresses must be an integer from 0 to (number_of_fuses - 1). Its value is either explicitly defined or calculated by default to be ($2^{address_bits}$ -1) where *address_bits* is the width of the address port specified in the FuseBoxInterface wrapper in the Test Core Description (TCD) of the fuse box. The “max_address” and “max_repair_address” strings can be used to specify a range. The “max_address” string represents the last address of the fuse box whereas the “max_repair_address” string represents the last address dedicated to memory repair which is determined by the value of number_of_fuses_for_repair defined in the DftSpecification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

AdvancedOptions/FuseBoxOptions wrapper or calculated by default to be the same as number_of_fuses - 1. For example, the range “max_address: 0” selects all bits of the fuse box whereas “max_repair_address: 0” selects the lower portion of the fuse box dedicated to memory repair. The read_address property can be repeated several times within the FuseBoxAccessOptions wrapper. If an address is specified in more than one property, the last specification prevails. To use this property, the operation property must be set to read.

- write_address : *range* ;

  A property that writes logic 1 into the specified fuse box address which is an integer. The range can be a single address, an address range, or a comma-separated list of the two formats. All addresses must be an integer from 0 to (number_of_fuses - 1). Its value is either explicitly defined or calculated by default to be ($2^{address_bits}$ -1) where *address_bits* is the width of the address port specified in the FuseBoxInterface wrapper in the Test Core Description (TCD) of the fuse box. The “max_address” and “max_repair_address” strings can be used to specify a range. The “max_address” string represents the last address of the fuse box whereas the “max_repair_address” string represents the last address dedicated to memory repair which is determined by the value of number_of_fuses_for_repair defined in the DftSpecification AdvancedOptions/FuseBoxOptions wrapper or calculated by default to be the same as number_of_fuses - 1. For example, the range “max_address: 0” selects all bits of the fuse box whereas “max_repair_address: 0” selects the lower portion of the fuse box dedicated to memory repair. The write_address property can be repeated several times within the FuseBoxAccessOptions wrapper. To use this property, the operation property must be set to program.

Examples

The following example has three TestStep wrappers. The first one verifies that the fuse box has never been written before (that is, all fuses have a value of 0). The second one writes 1 in a few selected locations. The third TestStep reads back a portion of the fuse box and verifies that the locations written in the second TestStep contain the value 1.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
TestStep(read_all0s) {
  MemoryBisr {
    run_mode : fuse_box_access ;
    Controller(FB_INST1) {
      FuseBoxAccessOptions {
        // Not necessary to specify read_address.
        // Default is to read all addresses and compare to 0.
        // Equivalent to specifying read_address(max_address:0):0 ;
        operation : read ;
      }
    }
  }
}
TestStep(write_selected_locations) {
  MemoryBisr {
    run_mode : fuse_box_access ;
    Controller(FB_INST1) {
      FuseBoxAccessOptions {
        operation : program ;
        // Write a 1 to 7 locations
        // 5:9 is equivalent to 5, 6, 7, 8, 9
        write_address : 2, 5:9, 12 ;
      }
    }
  }
}
TestStep(verify_first_16_locations) {
  MemoryBisr {
    run_mode : fuse_box_access ;
    Controller(FB_INST1) {
      FuseBoxAccessOptions {
        operation : read ;
        // Restrict the address range to the first 16 addresses
        // Set the expected value to 0
        read_address(15:0) : 0 ;
        // Override the expected value for the locations written in
        // previous step
        read_address(2, 5:9, 12) : 1 ;
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

MemoryBist

Specifies the use of MemoryBist in the current TestStep.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(pattern_name) {
    TestStep(name) {
      MemoryBist {
        run_mode                          : mode ;
        access_protocol                   : ijtag | parallel ;
        parallel_retention_time           : time | off ;
        reduced_address_count             : on | off ;
        AdvancedOptions {
          retention_test_phase            : all phases | start_to_pause |
                                           pause_to_pause | pause_to_end ;
          memory_reset                    : on | off ;
          preserve_bist_inputs            : on | off ;
        }
        DiagnosisOptions {
          extract_diagnosis_data          : on | off | auto ;
          preserve_fuse_register_values   : on | off ;
        }
        Controller(instance) {
        }
      }
    }
  }
}
```

Description

A wrapper that specifies the use of MemoryBist in the current TestStep.

You can specify this wrapper once for each TestStep wrapper. The MemoryBist specification consists of one AdvancedOptions wrapper, one DiagnosisOptions wrapper, and one or more Controller wrapper(s). These three wrappers can be specified in any order.

The repeatable Controller wrapper is used to define the inclusion of the given controller instance in the current TestStep, and allows specifying additional options that are specific to the instance. The properties specified outside the Controller wrapper apply to all Controllers.

The AdvancedOptions wrapper contains advanced options influencing how memory BIST is carried out. The DiagnosisOptions wrapper defines options related to memory BIST diagnosis.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- run_mode : mode ;

  A property that specifies the run mode to use with the controllers. The valid modes and their meaning are:

hw_default	Specifies that the memory BIST controllers use the default (hardcoded) settings. This run mode requires the least amount of configuration for the memory BIST controllers to launch tests.
run_time_prog	Specifies that the memory BIST controllers run the BIST test using the scanned-in settings prior to test execution. Individual memory BIST controllers are first accessed and configured before they all execute tests in parallel.
increment_failure_limit	Automatically advances the optional failure limit counter. This mode is useful for creating Stop-On-Error test patterns for bitmap applications.
check_repair_needed	Specifies that the repair analysis status registers need to be examined after BIST completes. The status registers report the repair status of the memories with built-in redundancy. This mode requires at least one tested memory to have repair analysis features defined in its Memory library description. This is indicated by the presence of a RedundancyAnalysis wrapper. See "Tessent Core Description" on page 4793.
setup_chain_test	Specifies that a special pattern is created to exercise all chains through the memory BIST controller and interfaces to verify the serial load/unload operation. The pattern shifts the registers composing all configuration/status chains as well as the BIRA registers if they are present. The setup_chain_test can be executed among many controllers simultaneously.

- access_protocol : ijtag | parallel ;

  A property that specifies the access method to configure, execute and monitor the controllers.

  The default ijtag value specifies the controllers are accessed using the IJTAG network. All MemoryBIST pattern generation capabilities in the PatternsSpecification is available. The protocol allows generating simulation testbenches for verification and ATE patterns for manufacturing test.

  Use the parallel protocol to verify the operation of the BAP direct access interface in simulation. You must create the BAP module with the BistAccessPort/DirectAccessOptions/direct_access property set to on in the DftSpecification. The simulation testbench controls and monitors the system-side ports of the BAP modules servicing the controllers specified in the TestStep/MemoryBist wrapper. The testbench executes Go/NoGo tests only. Selecting the parallel value requires the run_mode to be

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

hw_default. The compatible MemoryBIST pattern generation AdvancedOptions is limited to:

- freeze_step
- enable_memory_list
- apply_algorithm
- apply_operation_set

For more information on the implementation and use of the BAP direct access interface, refer to the "Advanced BAP Memory Access" Appendix of the *Tessent MemoryBIST User's Manual For Use With Tessent Shell*.

- parallel_retention_time : *time* | off ;

  A property that specifies the retention pause between algorithm sub-phases when performing parallel static retention testing. The time value is a real number followed by the unit without a space in between. Valid time units are s, ms, us, ns, and ps. The default time unit is ns. Specifying a time value for a controller testing only ROM results in an error.

- reduced_address_count : on | off ;

  A property that enables the memory BIST controller to be run on the four corners of the common memory address space. This property is useful for checking the proper functionality of the BIST controller without having to simulate the test of the entire memory space. Static timing analysis is required in the full chip context to verify that the timing between the controllers and the memories is valid. Note that the full address space should be simulated on the assembly module before you merge it into the chip.

  This property supports the hw_default or run_time_prog run modes.

  Patterns created for signoff usage with the create_patterns_specification command set the reduced_address_count property to on for memory BIST controllers with RAMs only. Memory BIST controllers with ROMs ignore this property and execute a full address count.

  When you enable this property and the memory BIST controller contains a step with multiple memories, the controller runs all algorithm phases of the test on the following four addresses if column_address_count_enable or row_address_count_enable are not enabled in the algorithm:

  - RowMin, ColMin
  - RowMax, ColMin
  - RowMin, ColMax
  - RowMax, ColMax

  If the memory has only row address bits, the reduced address range would be these two locations:

  - RowMin

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- RowMax

Note

When you enable this property and you apply a user-defined algorithm making use of operations that have either column_address_count_enable or row_address_count_enable, the algorithm may not simulate properly. A similar issue occurs if you apply a user-defined algorithm to memories in a controller step that are of different dimensions. It is therefore recommended that you use any of the SMarch* library algorithms to verify the functionality of the BIST controller and its connections to the memories.

- AdvancedOptions/retention_test_phase : all_phases | start_to_pause | pause_to_pause | pause_to_end ;

  A property that specifies whether the memory BIST controller applies the entire selected library algorithm or only a sub-phase of it. Specifying a phase value other than all_phases for a controller only testing ROM results in an error.

Note

The retention_test_phase property is typically not used and is ignored when a time value is specified for the parallel_retention_time property. The parallel_retention_time property automates the sequential execution of the start_to_pause, pause_to_pause, and pause_to_end phases with the specified pause time inserted between them.

The retention_test_phase property enables you to divide the step explicitly into three TestSteps where each TestStep runs a different phase. Valid values are as follows:

all_phases	Indicates that the controller applies the entire selected algorithm.
start_to_pause	Indicates that the controller applies only the portion of the selected algorithm that ends at the first static retention test pause. This value corresponds to the application of the checkerboard pattern.
pause_to_pause	Indicates that the controller applies only the portion of the selected algorithm that runs from the first static retention pause to the second static retention test pause. This value corresponds to the application of the inverse checkerboard pattern.
pause_to_end	Indicates that the controller applies only the portion of the selected algorithm that runs from the second static retention pause and onward.

- AdvancedOptions/memory_reset : on | off ;

  A property that instructs memory BIST controllers to only clear the memories and to skip the remaining phases of the library algorithm. The controllers clear memories by writing a value of 0 to all memory locations. You can only set this property to on when the algorithms used are limited to the following: SMarch, SMarchCHKB, SMarchCHKBci, SMarchCHKBcil, or SMarchCHKBvcd. Specifying an on value when other algorithms are

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

used results in an error. Specifying an off value causes the full algorithm to be executed. The memories are populated with zeros at the end of the algorithm since the last phase executed writes a value of 0 to all memory locations.

- AdvancedOptions/preserve_bist_inputs : on | off ;

  A property that specifies that the memory interface multiplexers continue to select the BIST inputs during and after the test has finished. Memories are under functional control except for the memories undergoing test by the test algorithm. When set to on, the memories remain under BIST control for the duration of the TestStep until the property is set to off in a subsequent TestStep. One application of this property is to maintain the BIST inputs to the memory between pattern TestSteps. Another application of this property is to allow the memory to be refreshed by the controller upon completion of all algorithms in the test.

- DiagnosisOptions/extract_diagnosis_data : on | off | auto ;

  A property that enables the sampling of internal registers of the memory BIST controller after the BIST run is completed. This property is useful when the Stop-On-Nth-Error diagnosis feature is enabled using failure_limit. The internal register values are only meaningful if the controller has stopped at the requested failure number and the DONE status signal is 0. Furthermore, the internal register values require post-processing to provide diagnostic information on the detected failures. The default value of auto resolves to on when failure_limit is not off, otherwise it resolves to off.

- DiagnosisOptions/preserve_fuse_register_values : on | off ;

  A property that specifies if the repair analysis registers preserve their values from a previous test step. This inhibits the loading of the BISR register values into the repair analysis registers when a new test step begins. This property supports the hw_default or run_time_prog run modes.

Examples

Example 1

An example MemoryBIST patterns specification generated with the create_patterns_specification command is shown below. This example runs a memory BIST in reduced address space.

```
PatternsSpecification(top,rtl,signoff) {
  Patterns(MemoryBist_P1) {
    tester_period : 100ns ;
    tck_ratio     : 2 ;
    TestStep(mem_test) {
      MemoryBist {
        run_mode : run_time_prog ;
        reduced_address_count : on ;
        Controller(core_inst1_top_rtl_tessent_mbist_c1_controller_inst) {
          (...)
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Starting with the v2020.4 release, the minimum BIST clock to TCK frequency ratio of 4 is no longer required for accessing internal chains of the MemoryBIST controllers. However, a ratio of 2 might still apply. Refer to "IJTAG Network Performance Optimization" in the *Tessent IJTAG User's Manual* and the description of the ControllerOptions/use_multicycle_paths property for more information. The ratio of 4 continues to apply to controllers created with older versions of Tessent MemoryBIST.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Controller

Specifies the inclusion of a MemoryBIST controller instance in the TestStep.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(pattern_name) {
    TestStep(name) {
      MemoryBist {
        Controller(instance) {
          parallel_retention_group : group_id ;
          AlgorithmSetupOverrides {
          }
          AdvancedOptions  {
          }
          RepairOptions {
          }
          DiagnosisOptions  {
          }
          MemoryInterface(memory_id) {
          }
          DramOptions {
          }
        }
      }
    }
  }
}
```

Description

A wrapper that specifies the inclusion of a MemoryBIST controller instance in the TestStep.

This wrapper is repeated for each controller instance that is to be run in this TestStep. All MemoryBIST controller instances are set up serially but run in parallel.

Arguments

- *instance*

  A string that specifies the ICL instance name of the MemoryBIST controller. The list of available MemoryBIST controller instances can be found using this command:

  ```
  get_icl_instances -filter
  {tessent_instrument_type==mentor::memory_bist &&
  tessent_instrument_subtype==controller}
  ```

  The create_patterns_specification command automates the creation of the Controller wrapper for each instance in the design.

- parallel_retention_group : *group_id* ;

  A property that specifies the group label associated with the controller where *group_id* is an alphanumeric label for the controller group. Underscores are allowed as part of the *group_id* label. Each controller group consists of all controllers with the same

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

parallel_retention_group property value. When performing a parallel static retention test on controller groups, the controller groups are run sequentially to minimize the peak power requirement while maintaining the ability to bring all controllers from all groups to the pause state and perform a single retention pause and, optionally, a single voltage bump for all memories in the chip.

The controller group execution order is determined by the order of appearance of the group labels. The following usage conditions apply:

- This property is meaningful only if the parallel_retention_time property in the MemoryBist wrapper specifies a time greater than zero.
- When performing a parallel static retention test on controller groups, all three algorithm sub-phases (start_to_pause, pause_to_pause, and pause_to_end) are applied. In this case, the AdvancedOptions/retention_test_phase property in the MemoryBist wrapper is ignored.

Examples

The following example assigns MemoryBIST controllers c1 and c4 to group number 1, and controller c3 to group number 2. Controllers c1 and c4 will be executed in the first group, followed by controller c3 in the second group.

```
TestStep(retention_test) {
  MemoryBist {
    run_mode : hw_default ;
    parallel_retention_time : 1s ;
    Controller(core_inst1_top_rtl_tessent_mbist_c1_controller_inst) {
      parallel_retention_group : 1 ;
      (...)
    }
    Controller(core_inst1_top_rtl_tessent_mbist_c3_controller_inst) {
      parallel_retention_group : 2 ;
      (...)
    }
    Controller(core_inst1_top_rtl_tessent_mbist_c4_controller_inst) {
      parallel_retention_group : 1 ;
     (...)
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

AlgorithmSetupOverrides

The AlgorithmSetupOverrides wrapper groups the DataGenerator and AddressGenerator wrappers, which define the initial values of the address and data pattern prior to the execution of the test algorithm.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(pattern_name) {
    TestStep(name) {
      MemoryBist {
        Controller(instance) {
          AlgorithmSetupOverrides {


            load_counter_a_start_count : int ; // default: 0
            AddressGenerator {
              AddressRegisterA | AddressRegisterB {
                load_bank_address   : binary_address | auto ;
                load_column_address : binary_address | auto ;
                load_row_address    : binary_address | auto ;
              }
            }
            DataGenerator {
              load_write_data             : binary_data | auto ;
              load_expect_data            : binary_data | auto ;
              invert_data_with_row_bit    : r[x] | none ;
              invert_data_with_column_bit : c[x] | none ;
            }
          }
        }
      }
    }
  }
}
```

Description

The AlgorithmSetupOverrides wrapper groups the DataGenerator and AddressGenerator wrappers, which require initialization prior to execution of the algorithm microprogram. The specified values override the values defined in the selected algorithm.

Using the AlgorithmSetupOverrides wrapper requires selecting an algorithm using the apply_algorithm property inside the MemoryBist/Controller/AdvancedOptions wrapper of the PatternsSpecification. Additional usage conditions apply to the AddressGenerator wrapper. Changing the initial value of any address register or of CounterA require setting the AdvancedOptions/incremental_test_mode property to on.

For more information, see the "Functional Debug Memory Access" appendix in *Tessent MemoryBIST User's Manual for Use with Tessent Shell*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- load_counter_a_start_count : *int* ;

  An optional property that specifies the startcount or starting count value for the CounterA general purpose module. The default value is “0’. The integer specified for load_counter_a_start_count must be in the range [0:2^n-1], where *n* is the number of bits in CounterA, specified by the counter_a_bits property in the DftSpecification/MemoryBist/Controller/AlgorithmResourceOptions wrapper.

- AddressGenerator/AddressRegisterA | B/load_bank_address : *binary_address* | auto ;

  An optional property that specifies a binary value to be loaded into the *BankAddress* for the named AddressRegisterA or AddressRegisterB. The value loaded into the *BankAddress* is the initial value of the bank address prior to execution of the microprogram. The value set with this property overrides the load_bank_address value defined in the memory library Algorithm wrapper for the named AddressRegisterA or AddressRegisterB. The width of *binary_address* must be equivalent to the number of bank address bits specified in the Core/Memory/AddressCounter wrapper of the memory library file. The property defaults to a *binary_address* equivalent to the lowest starting bank address of all memories tested in the controller step.

- AddressGenerator/AddressRegisterA | B/load_column_address : *binary_address* | auto ;

  An optional property that specifies a binary value to be loaded into the *ColumnAddress* for the named AddressRegisterA or AddressRegisterB. The value loaded into the *ColumnAddress* is the initial value of the column address prior to execution of the microprogram. The value set with this property overrides the load_column_address value defined in the memory library Algorithm wrapper for the named AddressRegisterA or AddressRegisterB. The width of *binary_address* must be equivalent to the number of column address bits specified in the Core/Memory/AddressCounter wrapper of the memory library file. The property defaults to a *binary_address* equivalent to the lowest starting column address of all memories tested in the controller step.

- AddressGenerator/AddressRegisterA | B/load_row_address : *binary_address* | auto ;

  An optional property that specifies a binary value to be loaded into the *RowAddress* for the named AddressRegisterA or AddressRegisterB. The value loaded into the *RowAddress* is the initial value of the row address prior to execution of the microprogram. The value set with this property overrides the load_row_address value defined in the memory library Algorithm wrapper for the named AddressRegisterA or AddressRegisterB. The width of *binary_address* must be equivalent to the number of row address bits specified in the Core/Memory/AddressCounter wrapper of the memory library file. The property defaults to a *binary_address* equivalent to the lowest starting row address of all memories tested in the controller step.

- DataGenerator/load_write_data : *binary_data* | auto ;

  An optional property that specifies a binary value to load into the *WriteData* register. The value loaded into the *WriteData* register is the initial value of the write data prior to

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

execution of the microprogram. The value loaded with this property overrides the load_write_data value defined in the memory library DataGenerator wrapper.

- DataGenerator/load_expect_data : *binary_data* | auto ;

  An optional property that specifies a binary value to load into the *ExpectData* register. The value loaded into the *ExpectData* register is the initial value of the expect data prior to execution of the microprogram. The value loaded with this property overrides the load_expect_data value defined in the memory library DataGenerator wrapper.

- DataGenerator/invert_data_with_row_bit : r[*x*] | none ;

  An optional property that enables you to specify a row address bit that inverts the applied write and expect data registers. The applied write and expect data registers are inverted when the specified row address bit is a "1", and not inverted when the row address bit is a "0". The specified value overrides the value defined in the selected algorithm. The default value of "none" specifies that no row bit is selected to invert the write and expect data registers. The specified r[*x*] index must select a valid row address bit where *x* is in the range of "0" to the number of row address bits, minus one. The number of row address bits is specified in the Core/Memory/AddressCounter wrapper of the memory library file.

- DataGenerator/invert_data_with_column_bit : c[*x*] | none ;

  An optional property that enables you to specify a column address bit that inverts the applied write and expect data registers. The applied write and expect data registers are inverted when the specified column address bit is a "1", and not inverted when the column address bit is a "0". The specified value overrides the value defined in the selected algorithm. The default value of "none" specifies that no column bit is selected to invert the write and expect data registers. The specified c[*x*] index must select a valid column address bit where *x* is in the range of "0" to the number of column address bits, minus one. The number of column address bits is specified in the Core/Memory/AddressCounter wrapper of the memory library file.

Examples

For examples and further information, see the "Functional Debug Memory Access" appendix in *Tessent MemoryBIST User's Manual for Use with Tessent Shell*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

AdvancedOptions

Specifies advanced test options for the selected memory BIST controller instance.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(pattern_name) {
    TestStep(name) {
      MemoryBist  {
        Controller(instance) {
          AdvancedOptions {
            apply_algorithm                    : algo_name | null ;
            apply_operation_set                : from_step | opset_name ;
            enable_memory_list                 : memory_id, ... ;
            freeze_step                        : off | integer ;
            freeze_test_port                   : off | integer ;
            compare_strobe_count               : off | auto |integer ;
            incremental_test_mode              : on | off ;
            test_time_multiplier               : real ;
            test_execution_cycles              : integer | auto ;
            MemoryClusterOptions  {
              configuration_data(interface_id) : binary | auto ;
            }
          }
        }
      }
    }
  }
}
```

Description

A wrapper that specifies advanced test options for the selected memory BIST controller instance.

The AdvancedOptions wrapper includes the MemoryClusterOptions wrapper which provides additional options specific to Shared Bus memory clusters.

Arguments

- apply_algorithm : *algo_name* | null ;

  A property that selects a test algorithm to apply to one or more memory BIST controller steps. The specified algorithm overrides the default algorithm associated with the controller step.

  - The algorithm name must match either a Tessent library built-in algorithm name or a user-defined algorithm name that was built into the memory BIST controller in the DFT insertion phase.
  - For the soft-programmable controller, the algorithm name may also match a user-defined algorithm specified in the MemoryOperationsSpecification that was loaded prior to pattern generation. If the name matches a hard-coded algorithm encoded in

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

the controller and a soft algorithm loaded in the MemoryOperationsSpecification, the tool applies the hard-coded algorithm.

- This property requires setting the MemoryBist/run_mode property to run_time_prog. The hw_default run mode can be used if the memory BIST controller enables the direct access capability for algorithm selection.

The null value should only be used in the context of BISR chain test patterns. This mode instructs the MemoryBIST controller to transfer the values from the BISR registers to the BIRA registers in the MBIST circuit. The controller performs the transfer and goes to the Done state without running any memory test algorithm.

- apply_operation_set : from_step | *opset_name* ;

  A property that selects a hard-coded operation set to apply to one or more memory BIST controller steps. When an operation set name is specified, the operation set overrides the default operation set.

  - The operation set name must match either a Tessent library built-in operation set name or a user-defined operation set name that was built into the memory BIST controller in the DFT insertion phase.
  - This property requires setting the MemoryBist/run_mode property to run_time_prog. The hw_default run mode can be used if the memory BIST controller enables the direct access capability for operation set selection.

  The from_step selection is only applicable with custom algorithms, and selects the default operation set associated with the controller step. Normally, the operation set specified in the custom algorithm definition overrides the default operation set associated with the controller step. The from_step value can be specified with a Tessent library algorithm, but the operation is equivalent to that with default values.

  When the apply_operation_set property is not specified, the default value is the operation set associated with the controller step if apply_algorithm specifies a Tessent library algorithm. Additionally, if apply_algorithm specifies a custom algorithm, then the default value for apply_operation_set is the operation set defined in the custom algorithm.

- enable_memory_list : *memory_id*, ... ;

  An optional property that specifies one or more memories to enable during parallel testing within a controller step. Selection of more than one memory must be specified as a comma separated list of memory ID's. If the property is not specified, all memories are enabled. A warning is issued if all memories are not enabled for testing in the current pattern. The memory_id is a valid memory ID specified by the Step/MemoryInterface wrapper within the MemoryBist wrapper in the DftSpecification and must belong to the BIST controller step specified by the freeze_step property. The MemoryBist/run_mode property must be set to run_time_prog. The hw_default run mode can be used if the memory BIST controller enables the direct access capability for memory selection.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- freeze_step : off | *integer* ;

  A property that specifies for the memory BIST controller to run only a single Step defined in the DftSpecification. The freeze_step property enables you to diagnose specific failures. Controller steps are numbered sequentially starting with 0. From the DftSpecification, the first controller Step wrapper is identified as “0”, the second Step wrapper is identified as “1”, and so on. To specify a single controller step, you must set the MemoryBist/run_mode property to run_time_prog. The hw_default run mode can be used if the memory BIST controller enables the direct access capability for step selection.When the freeze_step property is off, the memory BIST controller tests across all steps implemented in the memory BIST controller, and the results are accumulated across the entire BIST run.

- freeze_test_port : off | *integer* ;

  A property that specifies a test port on which to freeze the memory BIST controller so that you can diagnose specific failures. At least one memory in the TestStep wrapper much have multiple test ports. To use this property, you must set the MemoryBist/run_mode property to run_time_prog. The hw_default run mode can be used if the memory BIST controller enables the direct access capability for test port selection.When the freeze_test_port property is off, the memory BIST controller executes the steps for all test port combinations implemented in the MemoryBist controller in the DftSpecification, and the results are accumulated across the entire BIST run.

- compare_strobe_count : off | auto | *integer* ;

  An optional property that enables checking of the strobe count when the memory BIST controller is operating in Go/NoGo mode and is equipped with Enhanced Stop-On-Nth-Error (ESOE) capability. When enabled, the ESOE counter decrements for each compare event applied by the test algorithm. The compare events are indicated by the strobe_data_out property within the Cycle wrapper of the operation definition.

  The strobe count, in conjunction with the DONE status, provides an indication that the controller circuit is operating correctly. Any deviation would point to undesired circuit behavior which could sometimes be observed during circuit characterization. Use the compare_strobe_count property if characterization results, generally presented in the form of Shmoo plots, show isolated PASS regions outside the normal operating conditions of supply voltage and temperature.

  The default off setting specifies that the contents of the ESOE counter is not checked. When set to auto or an *integer* value, the content of the ESOE counter is checked at the end of the test against the *integer* value specified, or against a value automatically calculated by the tool when set to auto. If the tool does not automatically compute the expected compare events applied by the test algorithms, then the counter value must be obtained from simulation, or from known good circuits, and specified as an *integer* to the compare_strobe_count property. When specifying an *integer* value, it is important to note that the ESOE counter does not report the number of compares since it is a down-counter and it is possible for the counter to wrap around zero. Therefore, the expected counter value must be computed as the two’s complement of the module (number of compares/ESOE counter size). Specifying the correct *integer* value is necessary to obtain a PASS status.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following usage conditions apply:

- ESOE diagnosis must be enabled by setting the DftSpecification failure_limit property to 2 or higher. However, a value of 256 or higher is recommended for more reliable results.
- The PatternsSpecification failure_limit property must be off.
- The AdvancedOptions/incremental_test_mode property must be off.

- incremental_test_mode : on | off ;

  An optional property that enables a mode on the Tessent MemoryBIST controller to allow functional system debug. The incremental_test_mode property in the DftSpecification/MemoryBist/Controller/AdvancedOptions wrapper must have been set to "on" to create necessary hardware for this property. For more information, see the "Functional Debug Memory Access" appendix in *Tessent MemoryBIST User's Manual for Use With Tessent Shell*.

- test_time_multiplier : *real* ;

  A property that specifies a multiplier to extend or shorten the memory BIST run time. The final run time is equal to the original test time (as calculated by Tessent Shell) multiplied by the test_time_multiplier factor. The default is 1.

  This property is typically used when the test clock runs slower than expected by BIST. As a result, the memory BIST controller may not have completed its test within the allocated time when polled by the ATE. Specifying a value greater than 1 thus extends the computed test time; this is usually as a temporary measure (until the clock periods described in the Patterns/ClockPeriods wrapper is used).

- test_execution_cycles : *integer* | auto ;

  A property that specifies the number of clock cycles required to execute the algorithm. A setting of auto means that the pattern generation tool automatically computes the number of test execution cycles. In most cases this is sufficient; however, this property gives you the option to specify an absolute number of clock cycles. This accommodates extreme cases where the automatically-generated number is not accurate. Note that, in general, correcting the calculated test time with the test_time_multiplier property is preferable because it is less error-prone.

- MemoryClusterOptions/configuration_data(*interface_id*) : binary | auto ;

  A repeatable wrapper that specifies options related to Shared Bus memory clusters.

  The MemoryClusterOptions wrapper is used in the MemoryBist/Controller wrapper.

  A repeatable property that specifies a binary value to be applied to the interface port identified by *interface_id*. The binary value follows Verilog syntax (for example, 3'b101). If specified, it overrides the default ConfigurationData value from the memory cluster library file (which corresponds to the default "auto" value).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The value specified for this property is used for all controller steps executed in the TestStep. If the value is not compatible with all steps, the AdvancedOptions/freeze_step must also be specified.

Examples

Example 1

The following example illustrates the use of the apply_operation_set property to override the operation set to syncWR. This new operation set is applied to all Steps. The apply_algorithm property is not used which means the default algorithm to each Controller/Step is used.

```
PatternsSpecification(top,rtl,signoff) {
  Patterns(MemoryBist_P1) {
    tester_period : 100ns ;
    TestStep(mem_test) {
      MemoryBist {
        run_mode : run_time_prog ;
        Controller(core_inst1_top_rtl_tessent_mbist_c1_controller_inst) {
          AdvancedOptions {
            apply_operation_set : SyncWR ;
          }
        }
      }
    }
  }
}
```

Example 2

The following example illustrates how to specify specific memories for parallel testing within a controller through the enable_memory_list and freeze_step property. The freeze_step property is not required if the controller only has a single step. For this example, it is assumed the controller has multiple steps, and the memories to be tested are in the second step.

Two memories, M1 and M3, are specified as a Tcl list for set_config_value on the enable_memory_list property. The freeze_step property is specified as “1” to limit testing to this specific controller step that contains the memory interface for memories M1 and M3:

```
SETUP> set_config_value PatternsSpecification(top,rtl,signoff)/
Patterns(P1)/TestStep(run)/MemoryBist/Controller(c1)/AdvancedOptions/
enable_memory_list [list M1 M3]
SETUP> set_config_value PatternsSpecification(top,rtl,signoff)/
Patterns(P1)/TestStep(run)/MemoryBist/Controller(c1)/AdvancedOptions/
freeze_step 1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The specification can be examined with report_config_data on the AdvancedOptions wrapper:

```
SETUP> report_config_data PatternsSpecification(dashboard,rtl,signoff)/
Patterns(P1)/TestStep(run)/MemoryBist/Controller(c1)/AdvancedOptions

AdvancedOptions {
  enable_memory_list : M1, M3;
  freeze_step        : 1;
}
SETUP>
```

The specification properties for enable_memory_list can be retrieved as a Tcl list with the get_config_data command:

```
SETUP> get_config_value PatternsSpecification(dashboard,rtl,signoff)/
Patterns(P1)/TestStep(run)/MemoryBist/Controller(c1)/AdvancedOptions/
enable_memory_list
M1 M3
SETUP>
```

An additional example showing a similar use of the enable_memory_list and freeze_step properties in Functional Debug Memory Access mode can be found within the example PattternsSpecification provided in the “Algorithm Description” topic in *Tessent MemoryBIST User’s Manual For Use With Tessent Shell*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DiagnosisOptions

Provides diagnosis-related options and properties.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(pattern_name) {
    TestStep(name) {
      MemoryBist  {
        Controller(instance) {
          DiagnosisOptions {
            compare_go                    : on | off ;
            compare_go_id                 : on | off ;
            compare_memory_go             : on | off ;
            compare_misr                  : on | off ;
            comparator_id_select          : integer | all | none ;
            include_rom_signature_address : auto | on ;
            StopOnErrorOptions {
              failure_limit        : integer | auto_increment | off ;
              data_compare_time_slots     : all | even | odd ;
            }
          }
        }
      }
    }
  }
}
```

Description

A wrapper that provides diagnosis-related options and properties.

Arguments

- compare_go : on | off ;

  A property that specifies that the controller pass/fail status (GO port) is checked at the end of the test. This operation provides the composite pass/fail results of all memories the controller. You must disable the compare on the global pass/fail (GO) signal for some of the patterns needed in the memory self repair flow as documented in the Tessent MemoryBIST User's Manual.

  When you set this property to on, you typically set the compare_memory_go or compare_go_id property to on when testing at least one RAM, and the compare_misr property to on when testing ROM. By default, the pass/fail status of the ROM is determined from the MISR signature encoded in the memory interface or stored in the TSDB. If the ROM content has changed, the final signature does not match the default signature, and the differences results in a fail status. In this case, disable the compare_go property and provide a new signature or ROM content. Refer to the expected_rom_signature and rom_content_file property descriptions for further information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- compare_go_id : on | off ;

  A property that specifies that the compare status registers (GO_ID bits) are scanned and examined at the end of the test. This operation identifies which comparator failed but not necessarily the step in which it failed. This property is meaningful when at least one RAM is tested by the memory BIST controller. It is ignored on controllers testing only memories of type ROM.

  Note

  If you are not using the automation provided by Tessent SiliconInsight, you can find out which memory instance failed when the Step/comparator_location property was set to shared_in_controller in the DftSpecification by running the test with a freeze_step property set for each of the steps in the memory BIST configuration file.

  When the RepairOptions/check_repair_status property is set to non_repairable, the behavior of the compare_go_id property is disabled for repairable memories with local comparators. The GO_ID registers for non-repairable memories or repairable memories with shared comparators are always sampled when compare_go_id is set to on.

- compare_memory_go: on | off ;

  A property that enables examining the pass/fail status register of the individual memories at the end of the pattern execution. The per-memory status register must be present in the memory BIST controller hardware in order to use this property. Refer to the DftSpecification DiagnosisOptions/go_status property description for information on incorporating the per-memory status register.

- compare_misr : on | off ;

  A property that specifies the ROM MISRs are to be examined at the end of the test. This property is ignored when the controller is testing only memories of type SRAM or DRAM. By default, the MISR signature stored in the TSDB is used to compare the MISR register content. You may specify a different value using the MemoryInterface/ expected_rom_signature property.

- comparator_id_select : all | none | *integer* ;

  A property that specifies the individual status of the comparators in the controller to be routed to the compare status controller port. Valid values are as follows:

all	Selects the global controller comparator status signal.
none	Selects a constant logical 1, which implies the comparison result is forced to consistently pass.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

integer	A value between 1 and the number of shared comparators implemented in the controller. The number of shared comparators equals the maximum memory word size of all memories with the Step/comparator_location property set to shared_in_controller in the DftSpecification. The value of 1 corresponds to bit 0 of the memory data word.

These usage conditions apply:

- This property is only meaningful when the controller was generated with the DiagnosisOptions/comparator_selection_mux property set to on in the DftSpecification.

- include_rom_signature_address : auto | on ;

  A property that specifies whether to read the address that stores the ROM signature when the MISR signature is computed from the ROM content. This property applies to a controller that was generated with the DftSpecification property rom_signature_location set to max_address or min_address. For other settings of the rom_signature_location property, all ROM addresses will be read. The on value instructs the test algorithm to read all addresses of the ROM. The default value of auto resolves to on when the ROM signature was hard coded, or when the ROM signature was omitted and must be compared off-chip. The auto value resolves to exclude the maximum or minimum address when the ROM signature was stored in the ROM.

  The procedure to determine the expected ROM signature and store it into the ROM is described below.

  a. **Calculate the ROM signature without the signature address.**

     The revised ROM content is read from the file path specified by the rom_content_file property. During the test, the signature address is not read. The pass/fail status check is off for this portion of the procedure. After the test, the MISR signature is shifted out.

```
Controller(<icl_inst>) {
  DiagnosisOptions {
    compare_go   : off;
    compare_misr : on;
    include_rom_signature_address : auto;
  }
  MemoryInterface(<mem_id>) {
    rom_content_file : <rom_file_without_signature>;
  }
}
```

     The expected signature, derived from the ROM content file provided at pattern generation, is obtained from the simulation log file. An example of a partial simulation log file is listed below, showing the ROM signatures in hexadecimal format.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# 145100ns:  Starting MemoryBist controller execution :
             blka_rtl_tessent_mbist_c2_controller_inst
# 145100ns:  Checking GO is FAIL before execution
# 145100ns:  Checking DONE is FAIL before execution
# 159400ns:  Checking GO is PASS after execution completion
# 159400ns:  Checking DONE is PASS after execution completion
             (blka_rtl_tessent_mbist_c2_controller_inst)
# 166900ns:  Checking MISR registers of ROM instance m1 with user
             signature 0xDDFBA8531F160025C494E452FE
# 166900ns:  Checking MISR registers of ROM instance m2 with user
             signature 0x8358ECC2A65601BC25C494E452FE
# 166900ns:  Checking MISR registers of ROM instance m4 with user
             signature 0x3B4ED36C41BC
```

b. **Update and verify the ROM content with the new signature.**

Modify the ROM content with the new signature at the maximum or minimum address. Verify the full ROM content by re-executing the test with the signature address included. The test is expected to pass. The final MISR signature will differ from the value programmed into the signature address.

```
Controller(<icl_inst>) {
  DiagnosisOptions {
    compare_go   : off;
    compare_misr : on;
    include_rom_signature_address : on;
  }
  MemoryInterface(<mem_id>) {
    rom_content_file : <rom_file_with_signature>;
  }
}
```

c. **Verify the signature stored in the ROM.**

Finally, re-execute the test by excluding the signature address. This verifies that the signature is read from the ROM and used to compare the MISR signature. The test is expected to pass. The final MISR signature matches the value programmed into the signature address.

```
Controller(<icl_inst>) {
  DiagnosisOptions {
    compare_go   : on;
    compare_misr : off;
    include_rom_signature_address : auto;
  }
  MemoryInterface(<mem_id>) {
  }
}
```

- StopOnErrorOptions/failure_limit : *integer* | auto_increment | off ;

  A property that enables Enhanced Stop-On-Error (ESOE) and specifies the number of failures that the memory BIST controller detects before stopping and capturing all relevant information. This property has two usage scenarios. To learn more about how to use this

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

property, refer to the "Stop-On-Nth-Error Approach" section of the "Tessent MemoryBIST Diagnosis" chapter in the Tessent MemoryBIST User's Manual. Valid values are:

integer	Specifies the number of failures that the controller detects before stopping and capturing. The value is loaded in the ESOE counter and, if present, the failure limit counter.
auto_increment	Specifies that the stop-on-error counter is loaded with the value of the failure limit counter as it was set in a previous step, where run_mode was set to increment_failure_limit.
off	

These usage conditions apply:

- The DftSpecification/MemoryBist property DiagnosisOptions/StopOnErrorOptions/failure_limit was not set to off.
- When you set the failure_limit property to a non-zero value, you should set DiagnosisOptions/extract_diagnosis_data to on within the MemoryBist wrapper.

Note

When the failure limit is reached and the test stops, it is normal for the Done bit to fail because the test did not successfully complete (it aborted at some point).

To run diagnosis on a single memory:

- Set the MemoryBist/AdvancedOptions/freeze_step property to the step number that corresponds to the memory to diagnose.
- Use the MemoryBist/AdvancedOptions/enable_memory_list property to select a single memory. It requires that the AdvancedOptions/selective_parallel_memory_test property in the DftSpecification/MemoryBist wrapper was not set to off.

- StopOnErrorOptions/data_compare_time_slots : all | even | odd ;

  A property that selects the clock cycles in which StrobeDataOut is enabled during a read operation. Use this property when running Stop-on-Nth-Error diagnosis on memories that have StrobeDataOut enabled on consecutive clock cycles during a read operation. The Stop-On-Error diagnosis mode needs two clock cycles upon detecting an error before freezing the state of the memory BIST controller and memory interfaces. This property guarantees that the state of the controller is properly preserved when using operation sets containing a read operation with consecutive StrobeDataOut signals. Valid values are:

all	Enables StrobeDataOut on all clock cycles during Stop-On-Error Diagnosis mode.
even	Enables StrobeDataOut only on the even clock cycles during Stop-On-Error Diagnosis mode.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

odd	Enables StrobeDataOut only on the odd clock cycles during Stop-On-Error Diagnosis mode.

To use this option, the failure_limit property value must be a positive integer or set to auto_increment.

Examples

The following example extracts diagnosis data from a memory, m11, tested in the third controller step (that is, step 2). Because the memory is normally tested concurrently, two other memories, m12 and m13, are disabled when diagnosing the memory of interest:

```
PatternsSpecification(top,rtl,signoff) {
  Patterns(MemoryBist_P1) {
    tester_period : 100ns ;
    TestStep(mem_test) {
      MemoryBist {
        run_mode : run_time_prog;
        reduced_address_count : off ;
        DiagnosisOptions {
          extract_diagnosis_data : on ;
        }
        Controller(core_inst1_top_rtl_tessent_mbist_c1_controller_inst) {
          AdvancedOptions {
            freeze_step : 2 ;
            enable_memory_list : m11 ;
          }
          DiagnosisOptions {
            StopOnErrorOptions {
              failure_limit : auto_increment ;
            }
          }
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

RepairOptions

Provides repair-related options and properties.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(pattern_name) {
    TestStep(name) {
      MemoryBist  {
        Controller(instance) {
            RepairOptions {
              check_repair_status     : non_repairable | on | off ;
              extract_repair_fuse_map : on | off ;
              spare_element_priority  : row | column ;


            }
        }
      }
    }
  }
}
```

Description

The RepairOptions wrapper specifies what to shift out and monitor with repairable memories, as well as the spare allocation method to use. For Memory wrapper details, see "Tessent Core Description" on page 4793.

Arguments

- RepairOptions/check_repair_status : non_repairable | on | off ;

  A property that specifies that the repair analysis status registers are examined after the BIST run is completed. The status registers identify whether the memories with built-in redundancy require no repair, is repairable or is unrepairable. Valid values are:

on	Specifies that the repair analysis status registers are sampled and compared against the zero (0) value.
off	Specifies that the repair analysis status registers are not sampled.
non_repairable	Specifies that only the non-repairable flag of the status register is sampled.

  At least one memory tested by the controller must have repair analysis defined in its Memory library description. This feature is useful for separating the bad or unrepairable devices from the good or repairable devices during manufacturing test. For further details on repair status, refer to the "BIRA Repair Status Bits Checking" topic in the *Tessent MemoryBIST User's Manual For Use With Tessent Shell* manual.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- RepairOptions/extract_repair_fuse_map : on | off ;

  A property that specifies that the repair analysis fuse registers are examined after the BIST run is completed. The fuse registers contain the repair information for memories with built-in redundancy. When set to on, the repair analysis fuse registers are sampled and compared against the zero (0) value. Note that at least one memory tested by the controller must have repair analysis defined in its Memory library description. When set to off, the repair analysis fuse registers are not sampled.

  Note

  The content of the fuse registers are lost after performing this operation. This option should be set to "off" if you plan to perform a BIRA to BISR transfer.

- RepairOptions/spare_element_priority : column | row ;

  A property that specifies the type of spare element to be allocated first upon failures. This is used only for memories with redundant row and column elements. To find a repair solution for a faulty memory initially declared as unrepairable, try changing the repair strategy by selecting a different spare_element_priority setting. Valid values are:

column	Specifies that all single-bit errors are repaired by redundant column or IO elements; multi-bit errors are automatically repaired by spare row elements. If the number of faults exceeds the number of redundant columns or IO elements, the remaining redundant row elements are allocated for these faults. This is the most flexible solution because it automatically allocates redundant row elements for multi-bit failures while allocating redundant column or IO elements for single-bit failures. A redundant column or IO element cannot repair a multi-bit failure. If only a redundant column or IO element is available when a multi-bit error is detected, the repair status is unrepairable.
row	Specifies that redundant rows are allocated for all errors (single-bit and multi-bit). The redundant column/IO elements are only allocated for the remaining faults after all redundant rows are allocated.

Examples

The following example extracts the repair status and repair solution from all repairable memories.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
PatternsSpecification(top,rtl,signoff) {
  Patterns(MemoryBist_P1) {
    tester_period : 100ns ;
    TestStep(mem_test) {
      MemoryBist {
        run_mode : run_time_prog ;
        Controller(core_inst1_top_rtl_tessent_mbist_c1_controller_inst) {
          RepairOptions {
            check_repair_status : on ;
            extract_repair_fuse_map : on ;
          }
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

MemoryInterface

Specifies test options for the memory associated with the selected memory interface.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(pattern_name) {
    TestStep(name) {
      MemoryBist  {
       Controller(instance) {
          MemoryInterface(memory_id) { //repeatable
            expected_rom_signature : binary | auto ;
            rom_content_file       : filename ;
            comparator_id_select   : all | none | integer ;
          }
        }
      }
    }
  }
}
```

Description

A wrapper that specifies memory-specific options.

Arguments

- *memory_id*

  A string that identifies a given memory interface. The id is one of the ids defined in the Step/MemoryInterface and Step/ReusedMemoryInterface wrappers inside the DftSpecification.

- expected_rom_signature : binary | auto ;

  A property that specifies the ROM signature value to be compared to. This property is useful when the ROM content has changed and the ROM signatures were not updated into the instruments directory in the TSDB (see "Tessent Shell Data Base (TSDB)" on page 4807). A value of "auto" selects the signature calculated from the file referenced by rom_content_file. Refer to the rom_content_file argument description for more information on how the ROM MISR signature is specified.

- rom_content_file : *filename* ;

  A property that specifies a ROM content file from which the MISR signature is calculated. If expected_rom_signature is set to "auto", and the specified file is readable, the process_patterns_specification command calculates the MISR signature from the file, and applies the new MISR signature in the simulation testbench or manufacturing test pattern.

  When generating manufacturing test patterns, a ROM signatures file, named *design_name_design_id*.rom_signatures_manufacturing, is stored next to the patterns specification file in the TSDB. The ROM signatures file contains the MISR signatures calculated by the process_patterns_specification command for each combination of the ROM instance and its ROM content file. Note that the ROM signatures file is not created when generating simulation testbenches.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The ROM signature file may be referenced if the ROM content file is unavailable during subsequent runs to generate manufacturing patterns or to perform diagnosis using Tessent SiliconInsight. The tool can automatically process the ROM signatures file and retrieve the stored MISR signatures to create the manufacturing patterns. Guidance on where the MISR signature source is obtained, given the existence and validity of ROM content and signature files and settings for the expected_rom_signature argument, are provided in Table 13-15 below.

Table 13-15. ROM MISR Signature Source

expected_rom_signature Value	rom_content_file Value	ROM signatures file	MISR Signature Source
binary	N/A	N/A	Use expected_rom_signature value
auto	Specified and readable	N/A	Calculate from rom_content_file
auto	Specified and Not Readable	Readable	Retrieve from ROM signatures file. If no match, results in an error.
		Not readable	Results in an error
auto	Not Specified	Readable	Retrieve from ROM signatures file. If no match, use signature calculated from DftSpecification rom_content_file during MBIST insertion.
		Not readable	Use signature calculated from DftSpecification rom_content_file during MBIST insertion.

- comparator_id_select : <u>all</u> | none | *integer* ;

  A property that specifies the individual status of the comparators in the memory interface to be routed to the compare status controller port. Valid values are as follows:

all	Selects the interface's global comparator status signal.
none	Selects a constant logical 1, which implies the comparison result is forced to consistently pass.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

integer	A value between 1 and the number of comparators implemented for this memory. The number of comparators equals the memory word size. The value 1 corresponds to bit 0 of the memory data word.

These usage conditions apply:

- This property is only meaningful when the controller was generated with the DiagnosisOptions/comparator_selection_mux property set to on in the DftSpecification.

Examples

The following example extracts the MISR register of memory m1 after the test and compares the MISR register content against the hex value 24'h010204:

```
PatternsSpecification(top,rtl,signoff) {
  Patterns(MemoryBist_P1) {
    tester_period : 100ns ;
    TestStep(rom_test) {
      MemoryBist {
        run_mode : run_time_prog ;
        Controller(core_inst1_top_rtl_tessent_mbist_c1_controller_inst) {
          DiagnosisOptions {
            compare_misr : on ;
            compare_go   : off ;
          }
          MemoryInterface(m1) {
            expected_rom_signature : 24'h010204 ;
          }
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DramOptions

Provides DRAM AutoRefresh options and properties.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(pattern_name) {
    TestStep(name) {
      MemoryBist  {
        Controller(instance) {
          DramOptions {
            run_time_refresh_interval  : time | off | auto ;
            idle_time_refresh_interval : time | off | auto ;
          }
        }
      }
    }
  }
}
```

Description

The DramOptions wrapper enables you to specify the DRAM AutoRefresh properties that are applied during and after algorithm execution. AutoRefresh can be disabled or enabled with default or user specified durations between consecutive AutoRefresh operations. AutoRefresh is disabled by default.

The run_time_refresh_interval property specifies the duration between AutoRefresh operations during algorithm execution. The idle_time_refresh_interval property specifies the duration between AutoRefresh operations after algorithm test execution is completed.

Arguments

- run_time_refresh_interval : *time* | off | auto ;

  The run_time_refresh_interval property enables you to specify if AutoRefresh operations are to be automatically performed on a DRAM during the execution of the algorithm. Valid values are:

Table 13-16. DramOptions Parameter Values

time	AutoRefresh is enabled and the real number *time* specifies the duration between AutoRefresh operations. Valid values for specifying the units for *time* are as follows: • *s* — specifies the refresh interval in seconds. • *ms* — specifies the refresh interval in milliseconds. • *us* — specifies the refresh interval in microseconds. • *ns* — specifies the refresh interval in nanoseconds. • *ps* — specifies the refresh interval in picoseconds.
off	The AutoRefresh operations are disabled. This is the default configuration.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-16. DramOptions Parameter Values (cont.)

auto	AutoRefresh is enabled and the default duration is used between AutoRefresh operations. The default duration is the lowest value of all the RetentionTimeMax properties in the Memory library files divided by the maximum number of memory rows.

- idle_time_refresh_interval : *time* | off | auto ;

  The idle_time_refresh_interval property enables you to specify if AutoRefresh operations are to be automatically performed on a DRAM after the test execution completes. Valid parameter values are shown in Table 13-16 above.

Examples

The following example shows a DramOptions wrapper that specifies an elapsed time of 250 milliseconds between consecutive AutoRefresh operations for both during and after algorithm execution:

```
PatternsSpecification (top,rtl,signoff) {
  Patterns (MemoryBist_P1) {
    tester_period : 100ns ;
    TestStep (mem_test) {
      MemoryBist {
        Controller (core_inst1) {
          DramOptions {
            run_time_refresh_interval : 250ms ;
            idle_time_refresh_interval : 250ms ;
          }
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

MultiDieInterconnect

Specifies test patterns to generate for multi-die designs.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(patterns_name) {
    TestStep(step_name){
      MultiDieInterconnect {
        pin_action_verbosity       : boolean ; // default: on
        active_control_cells       : int     ; // default: unlimited
        Die(instance_name, ...) { // repeatable
          bsdl_file                : file_path ;
          bsdl_package_directories : directory_path, ... ; // repeatable
        }
        RunTest(test_name) { // repeatable
          // legal: inst_reg bypass_reg id_reg bscan_reg interconnect ;
          test_strategy : strategy ; // legal: odd_and_even n_log_n ;
        }
      }
    }
  }
}
```

Description

The MultiDieInterconnect wrapper specifies the test patterns to be generated for a multi-die design.

The repeatable RunTest wrapper defines the tests to be generated in the current pattern set. At least one RunTest wrapper is required, and no more than one MultiDieInterconnect wrapper can be specified per TestStep wrapper.

Legal values for test_name are:

- inst_reg
- bypass_reg
- id_reg
- bscan_reg
- interconnect

The interconnect test verifies the interconnects on the interposer between the dies and between the dies and the package ports. The other tests verify the basic functionality of the test hardware, as with the PatternsSpecification/Patterns/TestStep/BoundaryScan/RunTest() wrapper.

The test_strategy property is valid only for the interconnect test.

- odd_and_even: Tessent Shell uses a “01010101...” pattern to test the interconnects.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- n_log_n: Tessent Shell uses “01010101…”, then “00110011…”, “00001111…”, and so on.

Both of these strategies test for shorts between neighboring die pins by driving them to opposing values. The n_log_n test drives every pair of die pins at least once to opposing values, and takes more time.

Arguments

- pin_action_verbosity : on | off ;

  A property that enables you to specify whether your simulator displays comments each time a pin is compared to a specific value during testbench simulation. Be aware that adding comments for each output comparison increases pattern storage size.

- active_control_cells : unlimited | *integer* ;

  A property that specifies how many enable boundary-scan cells can be active simultaneously during the clamp and output tests. Valid integer values are 1 or greater.

  A large value generates a shorter overall test. Using a small value reduces the noise of simultaneously switching outputs created by test fixtures and package inductance.

  If the enable_group_info property is specified and the integer is smaller than the number of control cells in an enable_group, the enable group is split up and run sequentially in smaller groups of integer cells simultaneously.

- Die(***instance_name***, ...)

  Specifies the instance names of the dies for which the test patterns are generated.

- Die()/***bsdl_file***

  A property that specifies the path and filename of the Boundary Scan Description Language (BSDL) file to be used for the dies for which the test patterns are generated.

- Die()/***bsdl_package_directories ...***

  A repeatable property that specifies the paths containing the BSDL files to be used for the package for which the test patterns are generated.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Pattern Generation Script

```
# This script generates BoundaryScan patterns for die_b

set_context dft -rtl -design_id rtl

# read the library containing the pad cells
read_cell_library ../../library/mentor/Pads.lib
set_simulation_library_sources -extension v -y  ../../library/verilog

# read and elaborate the design
set_tsdb_output_directory ../tsdb_outdir
open_tsdb ../../die_a/tsdb_outdir
open_tsdb ../../die_b/tsdb_outdir
read_design die_a
read_design die_b
read_verilog ../design/package_top.v
set_current_design package_top
set_design_level chip

# read the BSDL information for the dies
read_bsdl ../../die_a/tsdb_outdir/instruments/die_a_rtl_bscan.instrument/
die_a_all_ports.bsdl
read_bsdl ../../die_b/tsdb_outdir/instruments/die_b_rtl_bscan.instrument/
die_b.bsdl

check_design_rules

# extract the ICL for the package level
extract_icl -write_in_tsdb on

set_system_mode setup

set_context patterns -ijtag -rtl -design_id rtl

# check the design
check_design_rules

set spec [create_patterns_specification]

# select all ports on die_a
read_config_data -in_wrapper $spec/Patterns(MultiDieInterconnect)
-from_string "
AdvancedOptions {
  ConstantPortSettings {
    bypass_inter_die : 0;
  }
}"

report_config_data $spec

# 2.5D interconnect pattern generation
process_patterns_specification

run_testbench_simulations
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Patterns Specification

```
PatternsSpecification(package_top,rtl,signoff) {
  Patterns(ICLNetwork) {
    ICLNetworkVerify(package_top) {
    }
  }
  Patterns(MultiDieInterconnect) {
    tester_period : 100ns;
    SimulationOptions {
      LowerPhysicalBlockInstances {
        die_a_inst1 : full;
        die_a_inst2 : full;
        die_b_inst1 : full;
        die_b_inst2 : full;
      }
    }
    TestStep(MultiDieInterconnect) {
      MultiDieInterconnect {
        Die(die_a_inst1) {
          bsdl_file : instrument(die_a_rtl_bscan)/die_a_all_ports.bsdl;
        }
        Die(die_a_inst2) {
          bsdl_file : instrument(die_a_rtl_bscan)/die_a_all_ports.bsdl;
        }
        Die(die_b_inst1) {
          bsdl_file : instrument(die_b_rtl_bscan)/die_b.bsdl;
        }
        Die(die_b_inst2) {
          bsdl_file : instrument(die_b_rtl_bscan)/die_b.bsdl;
        }
        RunTest(inst_reg) {
        }
        RunTest(bypass_reg) {
        }
        RunTest(id_reg) {
        }
        RunTest(bscan_reg) {
        }
        RunTest(interconnect) {
        }
      }
    }
    AdvancedOptions {
      ConstantPortSettings {
        bypass_inter_die : 0;
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TestbenchVerify

Creates the correct design view data for testbench files you create with the "write_patterns -verilog ..." command.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(patterns_name) {
   TestbenchVerify(id) {
      testbench_file_name : file_path_name ;
      compilation_options : options_string ;
      simulation_options  : options_string ;
    }
  }
}
```

Description

Use the TestbenchVerify wrapper to create simulation data for any simulation testbench generated by the "write_patterns -verilog ..." command.

The "create_patterns_specification -testbench_files *file_path_list*" command automatically creates one unique Patterns wrapper containing a single TestbenchVerify wrapper for each file found in the *file_path_list*. When created in this manner, the "compilation_options" and "simulation_options" are not specified and left for you to populate if needed.

Most of the time, one TestbenchVerify wrapper must reside in its unique Patterns wrapper because the simulation view for the current design and sub-blocks must be unique for each simulation testbench. The "create_patterns_specification -testbench_files *file_path_list*" command automatically populates the whole simulation view under the SimulationOptions/LowerPhysicalBlockInstances for the sub-blocks and SimulationOptions/current_design_view for the current design.

Arguments

- *id*

  The *id* of the TestbenchVerify wrapper can be any alphanumerical string plus underscores.

- testbench_file_name : *file_path_name*

  The *file_path_name* value of the "testbench_file_name" option is the full or relative path to a testbench file generated by the "write_patterns -verilog ..." command.

- compilation_options : *options_string*

  When specified, the *options_string* value are added to each simulator compilation command line created by the run_testbench_simulations command in the simulation script.

- simulation_options : *options_string*

  When specified, the *options_string* value are added to the command line that launches the simulator created by the run_testbench_simulations command in the simulation script.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICLNetworkVerify

Specifies the creation of a structural pattern to verify the proper operation of the IJTAG Network.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(patterns_name) {
    ICLNetworkVerify(id) {
      verify_scan_path          : on | off ;
      verify_internal_data_pins : on | off ;
      VerifyOnlyIclModules {
        name_patterns ; // repeatable
      }
      ExcludeIclModules {
        name_patterns ; // repeatable
      }
      VerifyOnlyIclInstances {
        name_patterns ; // repeatable
      }
      ExcludeIclInstances {
        name_patterns ; // repeatable
      }
      ExcludeIclScanRegisters {
        name_patterns ; // repeatable
      }
      ExcludeIclDataPins {
        name_patterns ; // repeatable
      }
    }
  }
}
```

Description

A wrapper that is used to specify the creation of a structural pattern to verify the proper operation of the IJTAG Network. In simulation, the pattern is useful for identifying mismatches between the ICL and Verilog model views. In manufacturing test, it covers manufacturing defects that can exist in the IJTAG network circuitry.

When the verify_scan_path property is on, the patterns are created by tracing backward from all ScanOutPorts and configuring all ScanMuxes found along the way to exercise all scan configurations. On the first scan load, the predictable captured values of the ScanRegisters are compared and the scan registers are loaded with all 0s with a single 1 closest to ScanIn. The ScanRegister is then parked in pause state and then unloaded to make sure the same data comes out, making sure the ScanRegisters are actually the length documented in the ICL model.

When verify_internal_data_pins is on, the internal DataOutPorts are forced and those values observed using ScanRegisters unloading. The internal DataInPorts are controlled using ScanRegisters loading and the resulting value directly observed on the internal DataInPorts. This feature is only usable in simulation and is useful for finding mismatches between the circuits and their ICL descriptions.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The create_patterns_specification command automatically partitions the ICL verification patterns per child physical block. Within each Patterns wrapper, it selects the IJTAG graybox view of the target child physical block using the SimulationOptions/ LowerPhysicalBlockInstances wrapper when it is available within its TSDB; otherwise, it selects the full view.

Licensing

If you use at least one Tessent signed instrument (MemoryBIST, LogicBIST, TestKompress, MissionMode) at your current design or at any lower hierarchical level, then any one of the respective licenses is sufficient. In all other cases, you need either an IJTAG or a MissionMode license.

Arguments

- verify_scan_path : on | off ;

  A property that specifies whether the patterns are to verify the integrity of the scan paths. When the property is on, the patterns are created by tracing backward from all ScanOutPorts and configuring all ScanMuxes found along the way to exercise all scan configurations. On the first scan load, the predictably captured values of the ScanRegisters are compared and the scan registers are loaded with a pattern consisting of alternating sequences of 0s and 1s, with an increasing sequence length. The TAP controller is then put in the pause state and the ScanRegisters are unloaded in a second scan load to make sure the same data comes out. This ensures that the ScanRegisters length matches the ICL model. While shifting the scan path verification pattern out of the ScanRegisters, the next ICL network configuration is already shifted into the network.

  ```
  load 01001100001111000000011111111...   expect captures values described in ICL
  load the next ICL network configuration  expect 01001100001111000000011111111...
  ```

- verify_internal_data_pins : on | off ;

  A property that specifies whether the patterns are to verify that data can be properly controlled at the internal DataInPorts and observed from the internal DataOutPorts. When the property is on, the internal DataOutPorts are forced and those values observed using ScanRegisters unloading. The values on internal DataInPorts are controlled using ScanRegisters loading and the resulting value directly observed on the internal DataInPorts. This feature is only usable in simulation and is useful for finding mismatches between the circuits and their ICL descriptions. Verifying the internal data pins can substantially increase the pattern count and also the time to generate the verification patterns. The increase depends on the number of DataInPorts and DataOutPorts, and the overall ICL network complexity.

  ICL Extraction sets the tessent_design_instance attribute to the pre-synthesis instance names whenever those names are available. During the creation of the data pin verification patterns, the actual design instance names are ignored. The verification pattern generator only uses the tessent_design_instance attribute to obtain the references to the design pins. Consequently, the RTL design files provided to ICL extraction must be used during the simulation of the verification patterns, at least in the case where synthesis modifies the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

design hierarchy (for example, because of "generate" loops). To obtain data pin verification patterns that can be simulated using the synthesized netlist with instance names that have been modified by synthesis, you either have to modify the tessent_design_instance attributes accordingly, or run ICL Extraction based on the synthesized netlist from the beginning (without providing RTL files to the tool).

- VerifyOnlyIclModules/*name_patterns* ;

  A Data wrapper used to specify that the test should only focus on a set of ICL modules. The *name_patterns* are glob patterns matching ICL module names instantiated below the current ICL design.

- ExcludeIclModules/*name_patterns* ;

  A Data wrapper used to specify that the test should exclude a set of ICL modules. The *name_patterns* are glob patterns matching ICL module names instantiated below the current ICL design. ICL modules having the tessent_ignore_during_icl_verification attribute (see icl_module) set to on are automatically excluded. Use this attribute on ICL modules that are not truly IJTAG-compliant and cannot pass the structural test performed in this test.

  When a child physical block is specified as "graybox" in the SimulationOptions/LowerPhysicalBlockInstances wrapper, the modules with the "keep_active_during_scan_test" ICL module attribute set to false are automatically added to the module exclusion list, as only the Scan resource instrument is present within the scan graybox view.

- VerifyOnlyIclInstances/*name_patterns* ;

  A Data wrapper used to specify that the test should only be performed on a set of ICL instances. The *name_patterns* are glob patterns matching ICL instances found below the current ICL design.

- ExcludeIclInstances/*name_patterns* ;

  A Data wrapper used to specify that the test should exclude a set of ICL instances. The *name_patterns* are glob patterns matching ICL instances found below the current ICL design. ICL instances having the tessent_ignore_during_icl_verification attribute set to on are automatically excluded.

  The list of excluded ICL instances is automatically augmented with all first-level child physical block instances that are not listed within the SimulationOptions/LowerPhysicalBlockInstances wrapper or are explicitly specified with the "interface" view.

- ExcludeIclScanRegisters/*name_patterns* ;

  A Data wrapper used to specify that the test should exclude a set of ICL Scan Registers. The *name_patterns* are glob patterns matching ICL instances found below the current ICL design. ICL Scan Registers having the tessent_ignore_during_icl_verification attribute set to on are automatically excluded. This wrapper is only used when verify_scan_path is set to on.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ExcludeIclDataPins/*name_patterns* ;

  A Data wrapper used to specify that the test should exclude a set of ICL DataPorts. The *name_patterns* are glob patterns matching ICL DataPorts found below the current ICL design. This wrapper is only used when the verify_internal_data_pins option is set to on. DataInPorts and DataOutPorts that have the tessent_ignore_during_icl_verification attribute set to on in their ICL definition are automatically excluded. This switch is only used when the "create_icl_verification_patterns -data_pin_test" option is set to on.

Note

The ICL verification pattern generator attempts to utilize a minimal scan path to reach all ICL instances that are to be tested by ICL verification patterns. This scan path might traverse ICL instances that have been excluded through the ExcludeIclModules or ExcludeIclInstances wrappers. ICL modules with the "tessent_ignore_during_icl_verification" attribute are not considered when searching a minimal scan path. During pattern simulation, the correct view for all design modules along the minimal scan path is automatically loaded.

Examples

Example 1

The following example specifies the creation of the ICL network verification patterns set. The ExcludeIclModules wrapper was used to exclude all ICL Module matching "abc*". Alternatively, the attribute tessent_ignore_during_icl_verification could have been set to true in the ICL module to exclude it automatically.

```
ICLNetworkVerify {
  ExcludeIclModules {
    abc* ;
  }
}
```

Example 2

The following example demonstrates creating one ICLNetworkVerify step using the signoff for the sub-physical regions and using it to replace what is being done in the default manufacturing pattern specification generated.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Patterns(ICLNetwork) {
  ICLNetworkVerify(top) {
    // Testing ICL scan registers outside all physical blocks
    ExcludeIclInstances {
      m8051_wrap_inst_m8051_inst1;
      m8051_wrap_inst_m8051_inst2;
      piccpu_inst1;
    }
  }
  ICLNetworkVerify(top_1) {
    // Testing ICL scan registers,
    // including those in physical block ICL instance
    // 'm8051_wrap_inst_m8051_inst1'
    ExcludeIclInstances {
      m8051_wrap_inst_m8051_inst2;
      piccpu_inst1;
    }
  }
  ICLNetworkVerify(top_2) {
    // Testing ICL scan registers,
    // including those in physical block ICL instance
    // 'm8051_wrap_inst_m8051_inst2'
    ExcludeIclInstances {
      m8051_wrap_inst_m8051_inst1;
      piccpu_inst1;
    }
  }
  ICLNetworkVerify(top_3) {
    // Testing ICL scan registers,
    // including those in physical block ICL instance 'piccpu_inst1'
    ExcludeIclInstances {
      m8051_wrap_inst_m8051_inst1;
      m8051_wrap_inst_m8051_inst2;
    }
  }
```

As shown, the ICLNetworkVerify(top) excludes all sub-physical blocks instances. Then, in turn, each sub-physical block instance is *not* excluded in subsequent ICLNetworkVerify(<*design_name*>_<*n*>) wrappers.

A comment is added outside each ExcludeIclInstances wrapper that explains the objective of this particular ICLNetworkVerify wrapper.

Related Topics

create_icl_verification_patterns

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ProcedureStep

Inserts arbitrary procedure steps into the pattern set.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(patterns_name) {
    ProcedureStep(procedure_step_name) {
      svf_file                        : file_path ;
      maintain_model_state_from_svf : on | off ;
      run_before_dft_control_settings : on | off ;
      iCall(iproc_name_path) { // repeatable
        iProcArguments {
          argument_name : argument_value ; // repeatable
        }
      }
      wait_time      : time ;
      tester_cycles : integer ;
      tck_cycles     : integer ;
      AdvancedOptions {
        execution_target : always | in_system_test_only | tester_only ;
        force_voltage(pin_name)      : voltage ; // repeatable
        split_patterns_file          : on | off ;
        network_end_state            : keep | reset | initial ;
        IjtagRetargetingOptions {
          merge_irunloop_only    : on | off ;
        }
      }
    }
  }
}
```

Description

The ProcedureStep wrapper is used to insert arbitrary procedure steps into the pattern set. The events of the procedure step consist of a test sequence expressed as a SVF (serial vector format) file or as an iProc followed by a pause of arbitrary length.

The example that follows is an ICL module that describes a programmable clock divider. If this ICL was visible when the process_dft_specification command was issued, a TDR was inserted to provide controllability of the DataInPorts from the IJTAG network. The ICL module defines the relationship between the clk_out and clk_in as well as the default division ratio between them. The DataInPort rst can be used to reset the divider and the DataInPort ratio[1:0] can be used to program the division ratio. As described in the Description section of the add_clocks command, an add_clocks command is automatically inferred when elaborating the design in the DFT context.

Note how the “rst” ICL port has the tessent_use_in_dft_specification attribute set to “auto_no_pi”. When unspecified, this attribute defaults to “auto”. With “auto”, the input pin is intercepted or sourced by a DataOutPort of a TDR if the port is seen to not already be sourced by a valid DataOutPort or a primary input port. When you set it to “auto_no_pi”, the pin is equipped with a local IJTAG control even if it is seen to already be sourced by a primary input.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You want to use the “auto_no_pi” value for ports like this “rst” port because they are often sourced by a system level reset port. Even though it is possible to toggle the system reset to reset the clock divider, the effect of the reset pulse is felt by the entire system and you may affect already running parts of the circuit. When you provide local controllability, it becomes possible to reset the clock divider without affecting the rest of the chip. See the description of create_dft_specification for more information about how DataInPorts are handled during DFT insertion.

If you add the tessent_default_procedure_name and optionally the tessent_default_procedure_parameters in the ToClockPort as shown below, the create_patterns_specification automatically adds the ProcedureStep to iCall the procedure with the specified parameters if it sees the clock of the memoryBIST controller going through this ToClockPort. It is critical that all elements you specify in the ToClockPort be consistent. The default proc and parameters must program the ToClockPort to behave as its Source, FreqDivider and FreqMultiplier properties indicates.

If the create_patterns_specification command sees that multiple cascaded modules need to be initialize, it initializes them in separate ProcedureStep such that the element feeding the other is initialized first. See Example 2 where a PLL is initialized first followed by the clock divider in its fanout.

```
Module clk_div {
  Attribute keep_active_during_scan_test = "true" ;
  ClockPort clk_in ;
  ToClockPort clk_out {
    Attribute tessent_default_procedure_name = “init” ;
    Attribute tessent_default_procedure_parameters = “ratio 4” ;
    Source  clk_in ;
    FreqDivider 4 ;
  }
  DataInPort rst {
    //This attribute makes sure the pin is intercepted by
    //create_dft_specification even if the pin is sourced by a PI.
    //Because the rst pin is often shared with many other things
    //in the chip, the local interception makes sure the clock divider can
    //be reset by its iProc without affecting the rest of the design.
    Attribute tessent_use_in_dft_specification = "auto_no_pi" ;
  }
  DataInPort ratio[1:0] ;
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The basic PDL iProcs to program the divider is shown here:

```
iProcsForModule clk_div
iProc init {{ratio 4}} {
  iClock clk_in
  iWrite rst 0b1
  iWrite ratio [expr int(log($ratio)/log(2))]
  iApply
  iWrite rst 0b0
  iApply
  iRunLoop 8 -sck clk_in
  iClockOverride clk_out -freqDivider $ratio
}
```

The iProc accepts a ratio argument that defaults to 4, and uses this value to program the divider. Once the divider is programmed, the iClockOverride command is issued to reflect that the division ratio has been changed. The later iClock commands that trace to this ToClockPort sees the overridden value. The basic PDL iProc shown above is correct in terms of the PDL commands but lacks the Tcl code needed to validate the ratio argument.

The enhanced procedure shown below has the proper validation code. The code at the top validates the ratio value and gives an error message when incorrect. When the iProc is called within a ProcedureStep wrapper, the iProc is called with a parameter that defines the validation_dict argument containing an entry for each parameter. In the example below, the entry “iproc_parameters ratio” is defined and contains the name of the property inside the ProcedureStep that defines its value. You can use this property to attach the error message to the property using the add_config_message command as shown below.

Finally, the validation_dict argument contains an entry called “validate_only”. It is expected that the code “if {[dict get $validation_dict validate_only]} {return}” exists in the iProc such that the iProc can be called with this entry set to 1 during the validation step of the process_patterns_specification command in order to run the validation code without actually running the PDL command. When the argument called “validation_dict” is not present, the iProc is not run when you call the process_patterns_specification command with the -validate_only option. This means that incorrect values of arguments passed to iProcs from a ProcedureStep wrapper is only detected when you run the process_patterns_specification command without the -validate_only option and the error message only shows up in the transcript and is not attached to the property objects in the GUI.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
iProcsForModule clk_div
iProc init {{ratio 4} {validation_dict {validate_only 0}} } {
  #### Validation part ####
  if {$ratio ni {1 2 4 8}} {
    set err "Illegal value '$ratio' for option 'ratio'. Must be 1, 2, 4, or 8. Default is 4."
  if {[dict exists $validation_dict iproc_parameters ratio]} {
      add_config_message -display $err \
          -config [dict get $validation_dict iproc_parameters ratio]
    } else {
      display_message $err
    }
    return -code error
  }
  if {[dict get $validation_dict validate_only]} {
    # Skip PDL part
    return
  }
  #### PDL part ####
  iClock clk_in
  iWrite rst 0b1
  iWrite ratio [expr int(log($ratio)/log(2))]
  iApply
  iWrite rst 0b0
  iApply
  iRunLoop 8 -sck clk_in
  iClockOverride clk_out -freqDivider $ratio
}
```

Arguments

- *procedure_step_name*

  A string that uniquely identifies the step within a pattern set. All of the procedure_step_name and test_step_name names must be unique within a Patterns wrapper. The string is only allowed to contains letters, numbers, and underscores.

- svf_file : *file_path* ;

  A property that specifies a file that defines an SVF sequence to be included in the ProcedureStep. The svf_file and iCall(iproc_name_path) properties are mutually exclusive.

  There are limitations on what can be in the svf_file. See import_patterns_from_svf command for a complete list. If the svf_file includes a reset of the Tessent TAP, then the Tessent TAP must be left in the Run-Test-Idle state (at the end of the svf_file). An example of this use model is if the svf_file contents must be run to provide access to the Tessent TAP.

  The file_path must have one of the following formats:

  - A simple filename with no directory path. In this case, the file must exist beside the ICL file.
  - A null string. In this case, no procfile is used.
  - An absolute path starting with a “/”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

  - A relative path starting with "./" or "../". In this case, the file must exist relative to the current working directory.

- maintain_model_state_from_svf : on | off ;

  An optional property that specifies how the SVF reader operates. When set to off, the reader operates in "blackbox mode", in that it does not apply the SDR (Scan Data Register) and SIR (Scan Instruction Register) commands to the internal state of the IJTAG network, nor does it check if the internal state of the IJTAG network matches the length of the provided SDR/SIR data. The default value is on.

- run_before_dft_control_settings : on | off ;

  The ProcedureStep/TestStep steps are always relative to each other in the order in which they appear in the PatternsWrapper. The settings described in the DftControlSettings wrappers are always applied first no matter where the wrapper is positioned.

  In some situations, you may need to run one or more custom ProcedureStep before the DftControlSettings can be applied; for example, powering up a section of the device. Specifying the run_before_dft_control_settings on the *N* first ProcedureStep wrappers them in their current relative order before the DftControlSettings are applied.

  An error is generated if you have a TestStep or a ProcedureStep with run_before_dft_control_settings set to off above a ProcedureStep with run_before_dft_control_settings set to on.

- iCall(*iproc_name_path*)

  A Wrapper that specifies the hierarchical ICL path to an iProc to be called. For example, an iProc named "reset" which has been registered against an ICL module mod1 that is instantiated as "mod1_inst" in the ICL is referenced as iCall(mod1_inst.reset)". The iCall wrapper and svf_file properties are mutually exclusive.

  When many iCall(*iproc_name_path*) wrappers exists in a ProcedureStep wrapper, the associated iCalls are processed in an 'iMerge -begin … iMerge -end' block so that their operations are performed in parallel. When there is only one iCall in the wrapper, it is not processed within an iMerge block. If the iProcs you want to iCall within a ProcedureStep wrapper contains instructions that are incompatible with the merging procedure (for example, iApply -end_in_pause), make sure to include it by itself inside one ProcedureStep wrapper to avoid it being process in an iMerge block.

  The iProc for the ICL module must have been sourced into the tool prior to executing the process_patterns_specification command; otherwise, an error is generated.

  Note

  If your store your iProcs in a file called XXX.pdl and an ICL module was read in from a file called XXX.icl and the two files existed in the same directory, the content of the file XXX.pdl is collected during ICL extraction and stored into a file called <design_name>. pdl inside the TSDB. This file is auto-loaded when ICL is elaborated, removing the need for you to manually source the iProcs before they can be referenced by the iCall() wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- iCall(*iproc_name_path*)/iProcArguments/*argument_name* : *argument_value* ;

  A Data wrapper that specifies argument-values pairs that specifies one or more arguments to pass to the iProc identified with the iproc_name property. The *argument_name* must exactly match the name of an argument of the actual iProc. Tessent Shell generates an error if the argument does not exist. The order of the arguments is not important as long as they formally match those defined for the iProc.

  The argument_value is automatically passed to the iProc in the right position.

  Tessent Shell generates an error if a required argument of the iProc is missing from the iProcArguments wrapper.

  If an optional argument is not specified, the default value of that argument is passed to the iProc when called.

  Consider the following example containing the declaration of an iProc called “reset”:

  ```
  iProc reset { initial_delay {final_delay 10us} } {…}
  ```

  The following configuration data:

  ```
  iCall(mod1_inst.reset) {
    iProcArguments {
      initial_delay : 1us;
    }
  }
  ```

  results in the following call to the iProc because the value of final_delay is optional and defaults to 10 us:

  ```
  iCall mod1_inst.reset 1us 10us
  ```

  The following configuration data results in an error because the initial_delay argument is required:

  ```
  iCall(mod1_inst.reset) {
    iProcArguments {
      final_delay : 22us;
    }
  }
  ```

  If you have an iProc that uses “args” as an argument, and you want to pass a series of option-value pairs to the args option, you use the following syntax:

  ```
  iProcArguments {
    args : “option1 value1
            option2 value2
            optionN valueN”;
  }
  ```

- wait_time : *time* ;

  A property that specifies a wait loop at the start of the procedure step. The wait_time property is mutually exclusive with the tester_cycles and tck_cycle properties. It expresses a wait loop in terms of time and runs as an iRunLoop -time *time* command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- tester_cycles : *integer* ;

  A property that specifies a wait loop at the start of the procedure step. The tester_cycles property is mutually exclusive with the wait_time and tck_cycle properties. It expresses a wait loop in terms of tester cycles and runs as an iRunLoop -time (*integer* * tester_period) command.

- tck_cycles : *integer* ;

  A property that specifies a wait loop at the start of the procedure step. The tck_cycles property is mutually exclusive with the wait_time and tester_cycles properties. It expresses a wait loop in terms of TCK cycles and runs as an iRunLoop *integer* -tck command.

  There is a relationship between *tck_cycle* and *tester_cycle* shown in the following equation:

  $$tck_cycle = tester_cycle * tck_ratio$$

  When *tck_ratio* is equal to 1, then *tck_cycle* and *tester_cycle* are equal.

- AdvancedOptions/execution_target : always | in_system_test_only | tester_only ;

  A property that specifies whether the current procedure step should be retargeted only by an In-System Test (IST) controller, only by a tester, or by both of these. Use this property to avoid pattern duplication and to make pattern customization more robust.

  The default is “always”, meaning that the current procedure step is solved. The “in_system_test_only” option restricts the current procedure step to top-level host scan interfaces identified as IST controllers. Use the “tester_only” option when the top host scan interface is a tester and hence there is an external I/O source.

- AdvancedOptions/force_voltage(*pin_name*) : *voltage* ;

  A repeatable property that specifies that the current test step needs a special voltage to be applied on a specific port of the device. This property is typically used in TestStep when MemoryBisr is used to program the fusebox. The property does not actually force the voltage. It only generates a special annotation near the top of the patterns file to let the test engineer know that test program commands are needed to supply the needed voltage on the given port. As soon as a single force_voltage property is specified, the split_patterns_file : on; property is implied to allow the insertion of the test program commands needed to apply the voltage values.

  Here is an example of the special annotation inside a stil file:

  ```
  Ann {*  Before applying this pattern, the following actions are
  required:  *}
  Ann {*  - Apply +0.25V on port PA *}
  ```

  Ann {* - Apply -0.25V on port PB *}

- AdvancedOptions/split_patterns_file : on | off ;

  A property that splits the pattern set into multiple files. The file containing the previous patterns is closed and a new one is created for the current step. The filename is the name of the Patterns wrapper initially. When a step with split_patterns_file set to on is reached, the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

new filename consists of the name of the Patterns wrapper concatenated with an underscore to the name of the ProcedureStep wrapper as shown here:

```
<pattern_name>_<procedure_step_name>
```

The split patterns file contains a special annotation near the top of the file that reads like this:

```
Ann {*  This pattern file requires some actions taken on the ATE. *}
Ann {*  This pattern assumes it is following another one: *}
Ann {*  the power and clocks must be kept alive between *}
Ann {*  the two patterns and all pins kept their state. *}
```

- AdvancedOptions/network_end_state : keep | reset | *initial* ;

  A property that specifies the action to take at the end of the ProcedureStep. The options for this property are named the same and perform the identical functions as the close_pattern_set command's arguments. For details, see the close_pattern_set command.

- AdvancedOptions/IjtagRetargetingOptions/merge_irunloop_only : on | off ;

  A property that specifies if iApply statements of concurrent iCalls are allowed to be merged during the processing of an iMerge block. This property and its options perform the identical function as “set_ijtag_retargeting_options -merge_irunloop_only”. Refer to the set_ijtag_retargeting_options command for more information.

Examples

Example 1

The following example shows a Patterns wrapper containing a single ProcedureStep that consists of a call to an iProc called “run” on the ICL instance “core1_i1.instrumenta_i1”. The parameter mode is passed to the iProc with a value equal to “random”. There is a pause of 4 milliseconds at the end of the procedure.

```
PatternsSpecification(mychip, gate, signoff) {
  Patterns(P1) {
    ProcedureStep(my_icall) {
      wait_time     : 4ms ;
      iCall(core1_i1.instrumenta_i1.run) {
        iProcArguments {
          mode : random ;
        }
      }
    }
  }
}
```

Example 2

The following example shows how the Patterns wrapper was auto created by the create_patterns_specification command to perform signoff simulation for the memory BIST controller found inside coreb (See Figure 13). Note how the content of the DftControlSettings, the SimulationOptions/LowerPhysicalBlockInstances and the ProcedureSteps wrappers were created. The command traces all the clocks of the MemoryBist controllers found in the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

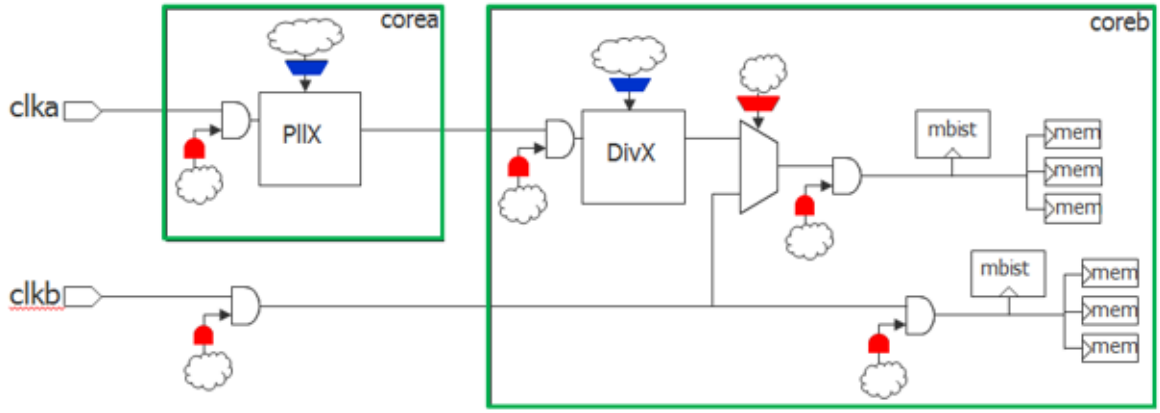
TestStep/MemoryBist wrapper. For every physical block entered, it asserts the dft_control_enable, nonscan_test, and all_test dft control signals it finds in them. For more information about those signals, see the add_dft_control_points, add_dft_clock_enables, and add_dft_clock_mux command descriptions. Normally, the signoff simulation of a given physical block instance is done using the interface view for all other physical block instances. In this example, the clock traced through the corea_i1 instance so the full view of corea_i1 is requested when simulating coreb_i1 in order for it to get its clocks. Finally, note how two ProcedureStep wrapper were auto inserted to initialize the PLL and clock divider seen when tracing the clocks. The clock divider is initialized after the PLL because it requires the output of the PLL to be running before it is initialized. This automatic inferring of the ProcedureStep only happens if you have defined the tessent_default_procedure_name attribute in the ToClockPort of the ICL model for the PLL and clock divider as described above in the Description section.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
PatternsSpecification(chip,rtl,signoff) {
  Patterns(MemoryBist_P2) {
    tester_period : 100ns ;
    DftControlSettings {
      corea_i1.all_test : 1 ;
      corea_i1.nonscan_test : 1 ;
      coreb_i1.all_test : 1 ;
      coreb_i1.nonscan_test : 1 ;
      all_test : 1 ;
      nonscan_test : 1 ;
    }
    ClockPeriods {
      clka : 3.00ns ;
      clkb : 9.50ns ;
    }
    SimulationOptions {
      LowerPhysicalBlockInstances {
        corea_i1 : full ;
        coreb_i1 : full ;
      }
    }
    ProcedureStep(clock_setup1) {
      iCall(corea_i1.pll.init) {
        iProcArguments {
          multiply : 10 ;
        }
      }
    }
    ProcedureStep(clock_setup2) {
      iCall(coreb_i1.clk_div.init) {
        iProcArguments {
          ratio : 4 ;
        }
      }
    }
    TestStep(run_time_prog) {
      MemoryBist {
        ...
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 13-131. Example Chip With Two Cores and Complex Clocking



Related Topics

create_patterns_specification

process_patterns_specification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SSNContinuityVerify

Specifies the creation of an SSN continuity pattern to test the internal path of the ScanHost from input to output.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  Patterns(patterns_name) {
    SSNContinuityVerify(id) {
      verify_datapaths_simultaneously : on | off ; // default: on
      ssn_datapath_ids : id, ... ;
      custom_pattern_sequence : sequence ;
      IncludedSshIclInstances {
        icl_instance ; // repeatable
      }
    }
  }
}
```

Description

A wrapper that is used to specify the creation of a continuity pattern to verify the SSN datapath. SSN datapaths from ssn_bus_in to ssn_bus_out are verified at the chip level.

The SSNContinuityVerify(default) wrapper adds an iReset to the pattern by default.

Arguments

- verify_datapaths_simultaneously : on | off ;

  A property that specifies to test all SSN datapaths simultaneously. If this is off, the tool instead tests SSN datapaths sequentially. This is equivalent to the "create_ssn_continuity_patterns -verify_datapaths_simultaneously" command and switch.

- ssn_datapath_ids : *id*, ... ;

  A property that restricts pattern generation to the specified SSN datapath IDs. The ICL must specify the datapath IDs as values of the tessent_ssn_datapath_ids attribute on the primary datapath bus inputs and outputs. This is equivalent to the "create_ssn_continuity_patterns -ssn_datapath_ids" command and switch.

- custom_pattern_sequence : *sequence* ;

  A property that specifies the continuity pattern data sequence. By default, the continuity pattern data is a random bit sequence. If you specify a shorter sequence than the active SSN datapath, the sequence repeats from the beginning. This is equivalent to the "create_ssn_continuity_patterns -custom_pattern_sequence" command and switch.

- IncludedSshIclInstances/*icl_instance* ;

  A Data wrapper used to specify that the SSN continuity patterns should be restricted to the datapaths that are required to access the instances specified as the <*icl_instance*>s. The specified instances must be valid ScanHost instances by instrument type and subtype. This

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

is equivalent to the “create_ssn_continuity_patterns -include_ssh_icl_instances” command and switch.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

EnableGroupInfo

Creates groups of Boundary Scan enable cells that can then be referenced using the BoundaryScan/RunTest/enable_group_info property.

Usage

```
PatternsSpecification(design_name ,design_id,pattern_id) {
  EnableGroupInfo(enable_group_info_id) {
    enable_cell_numbers : cell_number, ... ;
  }
}
```

Description

The EnableGroupInfo wrapper is used to group enable cells together that are tested simultaneously. The enable_group_info_id property can be used in the enable_group_info property to specify the enable groups to be tested separately and the sequence in which the enable groups are tested.

Arguments

- *enable_group_info_id*

  A string that uniquely identifies the EnableGroupInfo wrapper such that it can be referenced using the BoundaryScan/enable_group_info property. All enable cells that are not part of the listed enable groups, whether they are part of a not listed enable group or part of a no enable group, are added to a new artificial group and all tested simultaneously.

- enable_cell_numbers : *cell_number*, ... ;

  A property used to list the enable Boundary Scan cell numbers in the design to be tested simultaneously. The cell number corresponds to the cell number listed in the BSDL description.

Examples

The following example defines two EnableGroupInfo wrappers with different enable cells listed in each. The BoundaryScan wrapper shows two BoundaryScan wrappers referencing each enable group.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
PatternsSpecification(mychip,final,manufacturing) {
  Patterns(bscan) {
    TestStep(TS1) {
      BoundaryScan {
        enable_group_info : g1 ;
        RunTest(output) {
        }
      }
    }
    TestStep(TS2) {
      BoundaryScan {
        enable_group_info : g2 ;
        RunTest(output) {
        }
      }
    }
  }
  EnableGroupInfo(g1) {
    enable_cell_numbers : 3, 34, 78, 89 ;
  }
  EnableGroupInfo(g2) {
    enable_cell_numbers : 11, 37, 57, 101 ;
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SdfInfo

Defines SDF file usage scenarios that can be referenced by the Patterns/SimulationOptions/sdf_info property.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  SdfInfo(sdf_info_id) {
    SdfFile(path_to_file) {
      instance_names     : instance_name, ... ;
      include_macro_name : macro_name ;
      exclude_macro_name : macro_name ;
    }
  }
}
```

Description

The SdfInfo wrapper is used to define SDF file usage scenarios that can be referenced by the Patterns/SimulationOptions/sdf_info property.

The various SDF files are annotated on the specified instance path or the top-level module when the instance_names property is left empty. You can request the sdf annotation commands to only run or to not run when a given macro name is defined on the simulator command line.

Arguments

- *path_to_file*

  A string that refers to an SDF file. Relative path are expressed relative to the current working directory; they are converted to an absolute path in the testbench such that the simulation can be performed in any directory.

- instance_names : *instance_name*, ... ;

  A property that defines one or many Verilog instance paths that the SDF file applies to. When the property is specified as a null string (the default value), the SDF file is applied to the current design. The instance path names uses the period "." because the hierarchy separator is used in a Verilog simulator.

- include_macro_name : *macro_name* ;

  A property that defines a macro name that is used to surround the sdf_annotate command with a 'ifdef *macro_name*" statement. When you specify this property, the sdf annotation only happens if you pass "+define+macro_name" on the simulator command line. The sdf_annotation command is not surrounded by a 'ifdef statement unless this property is specified. This feature is very useful when you want to simulate with more than one version of an SDF for the same testbenches as shown in the following example.

- exclude_macro_name : *macro_name* ;

  A property used to define a macro name that is used to surround the sdf_annotate command with a 'ifndef *macro_name*" statement. When you specify this property, the sdf annotation

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

can be suppressed by passing “+define+macro_name” on the simulator command line. The sdf_annotation command is not surrounded by a ‘ifndef statement unless this property is specified.

Examples

The following example defines an SdfInfo wrapper that loads an SDF file called “top.sdf_slow” for the current design and another SDF file called “corea.sdf_slow” for two instances of the physical block “corea” when the +define+sdf_slow macro is passed to the simulator. It loads an SDF file called “top.sdf_fast” for the current design and another SDF file called “corea.sdf_fast” for two instances of the physical block “corea” when the +define+sdf_fast macro is passed to the simulator.

```
SdfInfo(chip_corea) {
  SdfFile(../SDFFiles/top.sdf_slow) {
    include_macro_name : sdf_slow ;
  }
  SdfFile(../SDFFiles/corea.sdf_slow) {
    instance_names : tlb.corea_i1, tlb.corea_i2 ;
    include_macro_name : sdf_slow ;
  }
  SdfFile(../SDFFiles/top.sdf_fast) {
    include_macro_name : sdf_fast ;
  }
  SdfFile(../SDFFiles/corea.sdf_fast) {
    instance_names : tlb.corea_i1, tlb.corea_i2 ;
    include_macro_name : sdf_fast ;
  }
}
```

Below is the corresponding Verilog module that is created and simulated in parallel to the chip to load the SDF files.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
module P1_sdf
  initial begin
    `ifdef sdf_slow
      sdf_annotate(/home/ver_eng/SDFFiles/top.sdf_slow,
                   TB.top_inst) ;
    `endif
    `ifdef sdf_slow
      $sdf_annotate(/home/ver_eng/SDFFiles/corea.sdf_slow,
                   TB.top_inst.tlb.corea_i1) ;
      $sdf_annotate(/home/ver_eng/SDFFiles/corea.sdf_slow,
                   TB.top_inst.tlb.corea_i2) ;
    `endif
    `ifdef sdf_fast
      sdf_annotate(/home/ver_eng/SDFFiles/top.sdf_fast,
                   TB.top_inst) ;
    `endif
    `ifdef sdf_fast
      $sdf_annotate(/home/ver_eng/SDFFiles/corea.sdf_fast,
                   TB.top_inst.tlb.corea_i1) ;
      $sdf_annotate(/home/ver_eng/SDFFiles/corea.sdf_fast,
                   TB.top_inst.tlb.corea_i2) ;
    `endif
  end
endmodule
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

LoadBoardInfo

A wrapper used to define the different tester and load board capabilities.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  LoadBoardInfo(load_board_info_id) {
    dot6_ttest : ttest_time ; // default: auto
    TesterInterface {
    }
    Loopbacks {
    }
    ACLoopbacks {
    }
  }
}
```

Description

The LoadBoardInfo wrapper is used to define the different tester and load board capabilities and configurations. It can be referenced using the PatternsSpecification/Patterns/load_board_info property—see "Patterns" on page 4563.

You would need this wrapper specified and referenced when:

- Your tester supports applying a 'Z' pattern state (highZ) to an input-only pin, and you want to use that capability in your BoundaryScan/RunTest dot6_ac_input test pattern.
- Your load board features loopbacks between output and input ports used in patterns, including boundary scan tested ports for BoundaryScan patterns.
- On your load board, some of your tested ports:
  - Have pull resistors.
  - Are uncontrollable by the tester (input and bidirectional ports).
  - Are unobservable by the tester (output and bidirectional ports).

When no LoadBoardInfo is specified, Tessent Shell assumes:

- All input and bidirectional tested ports are controllable.
- The tester cannot disable the drive of an input-only port.
- All output and bidirectional ports can be observed for states 0, 1 and Z.

Arguments

- *load_board_info_id*

  A string that uniquely identifies the LoadBoardInfo wrapper such that it can be referenced using the PatternsSpecification/Patterns/load_board_info property.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- dot6_ttest : *ttest_time* ;

  An optional time value that specifies the minimum time required by the slowest coupling capacitor, among all AC Loopbacks, to fully discharge (under 1 percent of charge). The value of this property (also referred to as TTest in the IEEE 1149.6 standard) is used mainly by DC-related BoundaryScan tests such as dot6_dc_input and dot6_dc_output. They ensure that DC levels are systematically blocked by all coupling capacitors and therefore cannot be captured by the test receiver.

  This property's value defaults to three times the slowest time constant along all AC pads that have an on-chip high-pass or low-pass filter as recommended by the IEEE 1149.6 standard. If no on-chip filter is present or if no BSDL file can be found, then the value defaults to three periods of TCK.

Examples

The following example is described in the next sections.

```
LoadBoardInfo(tester1) {
  TesterInterface {
    Port (TRST) {
      control     : tied_high ;
      observation : none ;
    }
    Port (IN1) {}
    Port (IN_DV1) {
      differential_inverse_of : IN_DV1_N ; //only specify if no BSDL
    }
    Port (IN_AC1) {}
    Port (IN_AC1_DV) {
      differential_inverse_of : IN_AC1_DV_N ; //only specify if no BSDL
    }
  }
  Loopbacks {
    IN1 : OUT1 ;
    IN_DV1 : OUT_DV1 ;
  }
  dot6_ttest : 15.0e-09s ;
  ACLoopbacks {
     IN_AC1 : OUT_AC1 ;
     IN_AC1_DV : OUT_AC1_DV ;
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TesterInterface

The TesterInterface wrapper enables you to specify which design ports are contact to a tester channel.

Usage

```
LoadBoardInfo(load_board_info_id) {
  TesterInterface {
    default_control       : auto | none | two_state_weak |
                            two_state_strong | three_state |
                            tied_high | tied_low ;
    default_observation   : auto | none | two_state | three_state ;
    default_pull_resistor : from_bsdl | none | up | down ;
    Port(port_name) {
      control             : from_default | none | two_state_weak |
                            two_state_strong |three_state |
                            tied_high | tied_low ;
      observation         : from_default | none | two_state |
                            three_state ;
      pull_resistor       : from_default | from_bsdl | none |
                            up | down ;
      differential_inverse_of : port_name ;
     }
  }
}
```

Description

Specifying this wrapper applies the default_control, default_observation and default_pull_resistor values to all ports of the current design that are not specified in a Port wrapper.

Arguments

- default_control : auto | none | two_state_weak | two_state_strong | three_state | tied_high | tied_low ;

  This property defines the control capabilities of the tester on ports that are not specified by a Port wrapper, or those that have a Port wrapper with the control : from_default value specified. The default value is “auto”, which resolves to “two_state_strong” for input ports, “none” for output ports and “three_state” for inout ports.

- default_observation : auto | none | two_state | three_state ;

  This property defines the observation capabilities of the tester on ports that are not specified by a Port wrapper, or those that have a Port wrapper with the observation : from_default value specified. The default value is “auto”, which resolves to “three_state” for output and inout ports and to “none” for input ports.

- default_pull_resistor : from_bsdl | none | up | down ;

  The property defines the polarity of the external weak pull resistor set by the tester on ports that are not specified by a Port wrapper, or those that have a Port wrapper with the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

pull_resistor : from_default value specified. The default value is “from_bsdl”. For more details, see the property description of Port/pull_resistor.

- Port(*port_name*)

  This identifier defines which port the following properties apply to. This must be an existing port of the current design.

  When specifying a differential pair, you must only specify the positive leg’s name. If you have Tessent Boundary Scan DFT inserted, then the negative leg is inferred from the BSDL file, if not, then you should use the differential_inverse_of property to specify the name of the negative leg of this differential port.

- Port/control : from_default | none | two_state_weak | two_state_strong | three_state | tied_high | tied_low ;

  This property defines the control capabilities of the tester on the identified port. The default is “from_default”, which specifies the control capabilities is defined by the default_control property. For an input-only port, setting the value “none” means the port is not contacted by the tester and is not part of Tessent Boundary Scan patterns.

- Port/observation : from_default | none | two_state | three_state ;

  This property defines the observation capabilities of the tester on the identified port. The default is “from_default”, which specifies the observation capabilities is defined by the default_observation property. For an output-only port, setting the value “none” means the port is not contacted by the tester and is not part of Tessent Boundary Scan patterns.

- Port/pull_resistor : from_default | from_bsdl | none | up | down ;

  This property defines the polarity of the external weak pull resistor set by the tester on the port if any. The default value is “from_default”, which specifies the polarity is defined by the default_pull_resistor property. When the value is “from_bsdl”, Tessent Shell tries to infer the value from the disabled result value in the BOUNDARY_REGISTER section of the BSDL file if one can be found. A disabled result value of “weak0” infers this property’s value to “down” and a disabled result value of “weak1” infers this property’s value to “up”.

- Port/differential_inverse_of : *port_name* ;

  This property is used to specify the negative leg of a differential port pair that complements this positive leg of the port. You should only use this property is you do not have Tessent BoundaryScan DFT.

Examples

The tester does not contact the TRST TAP port and four inputs in the following example. IN_DV1 is a differential input, so the control, observation, and pull_resistor properties are applied automatically to the complementary port, IN_DV1_N.

Note

The TRST port must be driven from the load board. This is the case for any global reset or enable port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
LoadBoardInfo(tester1) {
  TesterInterface {
    Port (TRST) {
      control : tied_high ;
      observation : none ;
    }
    Port (IN1) {}
    Port (IN_DV1) {
      differential_inverse_of : IN_DV1_N ; //only specify if no BSDL
    }
    Port (IN_AC1) {}
    Port (IN_AC1_DV) {
     differential_inverse_of : IN_AC1_DV_N ; //only specify if no BSDL
    }
  }
  [...]
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Loopbacks

The Loopbacks wrapper specifies one or more connection loopbacks from source ports to destination ports.

Usage

```
LoadBoardInfo(load_board_info_id) {
  Loopbacks {
    destination_port : source_port ; // repeatable
  }
}
```

Description

The Loopbacks wrapper specifies one or more connection loopbacks from source ports to destination ports of the current design. Loopbacks specify direct connections, as opposed to ACLoopbacks, which connect source ports to destination ports through a coupling capacitor.

Usage Conditions

Note the following:

- *source_port* and *destination_port* must be output and input direction respectively. Inout ports are not allowed.
- An 1149.1 source must loop back to an 1149.1 destination and an 1149.6 source must loop back to an 1149.6 destination. Interconnecting 1149.1 and 1149.6 ports together is not allowed.
- To loop back differential pairs, you can specify either of:
  - The output positive leg and the input positive leg
  - The output negative leg and the input positive leg

  You can also specify only the positive leg's port names. In this case, the negative leg is inferred from the BSDL file or taken from the differential_inverse_of properties of the specified ports.
- The *destination_port* must be the positive leg.

  The connection between the other two legs; that is, between the positive leg of the source and the negative leg of the destination, is inferred.

Arguments

- *destination_port* : *source_port* ;

  This port pair describes a direct connection made in the simulation testbench and assumed by the test patterns.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

In the example, OUT1 and OUT_DV1 are looping back to uncontacted ports IN1 and IN_DV1 respectively. Because IN_DV1 and OUT_DV1 are differential ports, their negative leg is also automatically looped back.

```
LoadBoardInfo(tester1) {
  [...]
  Loopbacks {
    IN1 : OUT1 ;
    IN_DV1 : OUT_DV1 ;
  }
  [...]
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ACLoopbacks

The ACLoopbacks wrapper specifies one or more connection loopbacks from AC source ports to AC destination ports.

Usage

```
LoadBoardInfo(load_board_info_id) {
  ACLoopbacks {
    destination_port : source_port ; // repeatable
  }
}
```

Description

The ACLoopbacks wrapper specifies one or more connection loopbacks from AC source ports to AC destination ports of the current design. ACLoopbacks specify port connections through a coupling capacitor, as opposed to Loopbacks, which connect directly source ports to destination ports.

Usage Conditions

Note the following:

- *source_port* and *destination_port* must be output and input direction respectively. Inout ports are not allowed.
- 1149.1 ports are not allowed.
- To loop back differential pairs, you can specify either of:
  - The output positive leg and the input positive leg
  - The output negative leg and the input positive leg

  You can also specify only the positive leg's port names. In this case, the negative leg is inferred from the BSDL file or taken from the differential_inverse_of properties of the specified ports.
- The *destination_port* must be the positive leg.

  The connection between the other two legs; that is, between the positive leg of the source and the negative leg of the destination, is inferred.

Arguments

- *destination_port* : *source_port* ;

  This port pair describes a connection through a coupling capacitor made in the simulation testbench and assumed by the test patterns.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

In the example, OUT_AC1 and OUT_AC1_DV are looping back to uncontacted ports IN_AC1 and IN_AC1_DV respectively. Because IN_AC1_DV and OUT_AC1_DV are differential ports, their negative leg is also automatically looped back. The resulting Verilog coupling capacitors waits 15 ns before setting the loopback signal going into the inputs to Z.

```
LoadBoardInfo(tester1) {
  [...]
  dot6_ttest : 15.0e-09s ;
  ACLoopbacks {
    IN_AC1 : OUT_AC1 ;
    IN_AC1_DV : OUT_AC1_DV ;
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

IjtagRetargetingOptions

Documents and controls the IJTAG retargeting options to be used in the patterns of the current specification.

Usage

```
PatternsSpecification(design_name,design_id,pattern_id) {
  IjtagRetargetingOptions {
    iapply_target_annotations                    : off | dense | full ;
    compare_constant_capture_values              : on | off | auto ;
    compare_constant_data_out_ports              : on | off | auto ;
    ignore_non_reset_end_state_implicit_compares : off | on | auto ;
    constrain_update_en_only                     : auto | on ;
    default_scan_out_strobe_point : before_rising_edge_of_tck  |
                                    before_falling_edge_of_tck |
                                    auto ;
    physical_block_iapply_partitioning       : off | all |
                                               top_physical_blocks ;
    interface_type                           : auto | ijtag | tpsp ;
    max_operations_per_iapply                : integer ;
    merge_irunloop_only                      : on | off ;
    network_end_state_warnings               : on | off ;
    disable_pio_operations                   : on | off ;
    ignore_iwrite_when_value_already_present : on | off ;
    allow_skip_rti                           : on | off ;
    generated_timeplate_name        : timeplate_name; // Default: gen_tp%
    inject_cycles                   : inject_cycles_list ;
    extra_control_setup_hold_cycles : extra_cycles ; //Mutually exclusive
                                                      //with inject_cycles
    extra_tms_setup_hold_cycles     : extra_cycles ; //Mutually exclusive
                                                      //with inject_cycles
    tck_ratio           : auto | 1 | 2 | 4 | 6 | 8 | … | 1024 ;
    tck_period                      : tck_period ; // Default: 100ns
    annotation_parameter_values     : parameters ;
  }
  Patterns(patterns_name) {
    IjtagRetargetingOptions {
      // The patterns reference this content. The default values are
      // inherited from PatternsSpecification/IjtagRetargetingOptions.
    }
  }
}
```

Description

The IjtagRetargetingOptions wrappers provide granular control of the IJTAG retargeting options and replace the use of the set_ijtag_retargeting_options command before you run the process_patterns_specification command. The top-level PatternsSpecification/IjtagRetargetingOptions wrapper acts as a default to all Patterns in the current specification. The Patterns/IjtagRetargetingOptions wrapper acts as an override of the general PatternsSpecification settings.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The Patterns/IjtagRetargetingOptions and Patterns/TestbenchVerify wrappers are mutually exclusive within the same Patterns wrapper.

The create_patterns_specification command (when not run with the "-testbench_files" switch) populates the values of the PatternsSpecification/IjtagRetargetingOptions wrapper properties from retargeting options that have been specified using the set_ijtag_retargeting_options command. After the patterns specification is created, you can add or change property values in the IjtagRetargetingOptions wrapper as needed.

When you run the process_patterns_specification command, the tool checks for consistency between the properties specified in the IjtagRetargetingOptions wrapper and options specified by the set_ijtag_retargeting_options command.

You can override the consistency checks completed by the process_patterns_specification command with either the "-import_command_options" or "-ignore_command_options_inconsistencies" switch.

Arguments

- IjtagRetargetingOptions/<*retargeting_options*>

  An optional set of properties that specify the IJTAG retargeting options available in the IjtagRetargetingOptions wrapper. Refer to the Usage section for a list of the properties and their default values, as well as the property descriptions listed in the set_ijtag_retargeting_options command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

InSystemTest

Specifies the creation of InSystemTest patterns inside the PatternsSpecification wrapper.

Usage

```
PatternsSpecification (design_name, design_id, pattern_id) {
   InSystemTest {
      memory_file_icl_register_annotation : on | off
      Controller(icl_instance_name) {
         apb_secondary_clk_ratio: 1..16 ;  // default: 1
         TestProgram(index) {
         // TestProgram is a repeatable wrapper, where index starts at 0
            pattern : pattern_wrapper_name ;
         }
         TestProgramGroup(start_index, end_index) {
         // TestProgramGroup is a repeatable wrapper
            pattern: pattern_wrapper_name ;
         }
      }
   }
}
```

Description

InSystemTest is an optional wrapper specified within the PatternsSpecification used to generate patterns for an IST controller. By default—that is, usage is set to simulation—the tool generates both a Verilog testbench and memory contents.

If you specify manufacturing_test as the usage, for the direct memory access architecture, the tool generates the memory contents file. For the CPU-based access architecture, it populates the mentor::in_system_test::PatternsSpecification dictionary.

The Controller wrapper is repeatable, depending on how many IST controllers you defined with the DftSpecification/InSystemTest wrapper.

The *icl_instance_name* is an existing ICL instance name. For details about Tessent MissionMode, refer to the *Tessent MissionMode User's Manual*.

Arguments

- memory_file_icl_register_annotation : on | off

  An optional property that adds annotations as comments in the memory (*.mem*) file that shows the mapping between the bitstream and the ICL scan registers. The default is off.

  The main purpose of this property is to help you understand the memory file contents. This is mainly useful for CpuInterface, but is also available for direct memory access. A typical use for CpuInterface is for the user-written CPU program to diagnose which ICL register failed by matching the *.mem* binary value with the register using this annotation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Caution

Do not manually patch the memory file. Changing either write data or expect data inconsistently can cause unpredictable results.

For the CpuInterface testbench memory file, there are two less ICL register bits per write and expect line compared to the data width because the leading two bits are opcodes for the IST controller and are not supplied to the ICL network. For the DirectMemoryAccess memory file, there are as many ICL registers per write and expect line as the data width. For padding bits that do not correspond to any ICL register, the "@" symbol indicates padding.

When the write data is all 0s, the file represents the data as an optimized (*WRITE0) instruction in the *.mem* file to reduce memory size. In this case, there is no "W" (write) annotations.

For an example of an annotated memory file, see Example 2.

- apb_secondary_clk_ratio: 1..16

  An optional property that specifies the clock divider ratio between controller clock PCLK and the to_ijtag_tck output to the downstream IJTAG network. The tool allows this property only when the controller type is ApbSecondary. The integer range of the clock divider ratio is 1 to 16. The default is 1.

- TestProgram (*index*)

  A repeatable wrapper that specifies a single test program to be run by the IST controller. For the CPU-based access controller, you can specify any number of test programs. For the direct memory access controller, you can specify a maximum of max_test_program_count wrappers.

  The *index* is an integer value starting at 0 that represents the test program number. For direct memory access, you can specify *index* from 0 to max_test_program_count - 1. For CPUs, any number of test programs are allowed.

  Note

  The value of tck_ratio may depend on the maximum TCK clock frequency that can be applied to the circuit. See the "IJTAG Network Performance Optimization" section in the *Tessent IJTAG User's Manual* showing how to maximize the frequency of the IJTAG network test clock.

- TestProgram/pattern : *pattern_wrapper_name*

  The pattern:*pattern_wrapper_name* property specifies the existing pattern wrapper name for the controller to apply. Properties such as test_period, tck_ratio, constant port settings, and clock periods for the test program are sourced from the specified pattern.

- TestProgramGroup(*start_index*, *end_index*)

  A repeatable wrapper that specifies consecutive test programs to be executed by the IST controller. Valid values are from 0 to max_program - 1 and must not conflict with any other

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TestProgram() or TestProgramGroup() indices. The range (*start_index*, *end_index*) should match the number of TestSteps or ProcedureSteps in the referenced pattern_wrapper_name.

For an example of how to use TestProgramGroup, see "Example 4" on page 4605 in the PatternsSpecification/Patterns/TestStep/LogicBist wrapper description.

- TestProgramGroup/pattern : *pattern_wrapper_name*

  The pattern:*pattern_wrapper_name* property specifies the existing pattern wrapper name for the controller to apply. Properties such as test_period, tck_ratio, constant port settings, and clock periods for the test program group are sourced from the specified pattern.

Examples

Example 1

The following example shows a dofile snippet for the In-System Test patterns specification after inserting three IST controllers.

```
  ...
  PatternsSpecification(top, gate, ist) {
    Patterns(m8051_A_ist) {
       # test steps here ...
       }
    Patterns(m8051_B_ist) {
       # test steps here ...
       }
    Patterns(piccpu_ist) {
       # test steps here ...
       }
    InSystemTest {
      Controller(top_gate_tessent_in_system_test_m8051_A_inst) {
        TestProgram(0) {
          pattern: m8051_A_ist ;
        }
      }


Controller(top_gate_tessent_in_system_test_m8051_B_inst) {
        TestProgram(0) {
          pattern: m8051_B_ist ;
        }
      }
      Controller(top_gate_tessent_in_system_test_piccpu_inst) {
        TestProgram(0) {
          pattern: piccpu_ist ;
        }
      }
    }
  }
}
process_patterns_specification
...
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

This example begins with a snippet from a memory file for the CpuInterface protocol that does not have annotations. The example shows an IST controller with data_width = 8. The first two bits of each pattern data line is an opcode for the IST controller that does not correspond to any ICL register. The remaining 6-bit values correspond to the ICL registers.

```
//memory_file_icl_register_annotation : off
//two opcode bits followed by six value bits
01000000
00111111
00100010
```

When the memory_file_icl_register_annotation property is on, the result is the addition of mapping annotations into the memory file as comments. The annotation formats write data as "W{icl_register_list}" and expect data as "E{icl_register_list}".

```
//memory_file_icl_register_annotation : on
//two opcode bits followed by six value bits
01000000 //ToPauseDR(write=000000, read=100010)   W{icl_register_list}
00111111
00100010 //                                        E{icl_register_list}
```

The line "00111111" was not annotated in this example because it refers to the mask for IJTAG scans when the read in the line "00100010" has Xs.

Further, the annotation fills the icl_register_list with register names down to the bit of the write or read for multi-bit registers. The write of zeroes to misr[2], misr[1], misr[0], misr.sib, lbist.sib, lbist.ijtag_so_ff, and the read of ones from misr[6], misr[2] and zeros from misr[5], misr[4], misr[3], and misr[1] results in an annotation similar to the following:

```
//memory_file_icl_register_annotation : on
//two opcode bits followed by six value bits
01000000 //ToPauseDR(write=000000, read=100010)   W{misr[2],misr[1],
         //misr[0],misr.sib,lbist.sib,lbist.ijtag_so_ff}
00111111
00100010 //                                        E{misr[6],misr[5],
         //misr[4],misr[3],misr[2],misr[1]}
```

Padding bits show in an annotation as "@" and you can ignore these. The tool adds padding bits to the IJTAG pattern to fit the IST controller data width.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DefaultsSpecification Configuration Syntax

This section describes the configuration data syntax used inside Tessent Shell to specify default values used by the create_dft_specification command. Tessent Shell comes with built-in default values for all configuration properties. You can use the DefaultsSpecification wrapper to change many of those default values to match your preferences and requirements.

For complete information, see the "DftSpecification" on page 3848

DefaultsSpecification 4708
DefaultsSpecification/DftSpecification. 4709
DynamicDftSignals. 4712
StaticDftSignals 4713
IjtagNetwork 4715
BoundaryScan. 4720
MemoryBisr 4727
MemoryBist 4732
DefaultsSpecification/PatternsSpecification 4751
SignOffOptions. 4753
ManufacturingOptions 4755

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DefaultsSpecification

Specifies changes to the built-in default values of all configuration properties.

Usage

```
DefaultsSpecification(policy) { //Legal: company | group | user
  DftSpecification {
  }
  PatternsSpecification  {
  }
}
```

Description

Specifies changes to the built-in default values of all configuration properties.

You can use the DefaultsSpecification wrapper to change many of the built-in default values to match your preferences and requirements. The defaults setting mechanism also comes with a hierarchy of DefaultsSpecification wrappers that allow you to have company, group, and user-level defaults. You can load three configuration files in your .tessent_startup file: one containing a DefaultsSpecification(company) wrapper, one containing a DefaultsSpecification(group) wrapper, and one containing a DefaultsSpecification(user) wrapper. The final default value of a given property is taken from the DefaultsSpecification(user) wrapper if it is specified in that wrapper; it is taken from the DefaultsSpecification(group) wrapper if it is defined in that wrapper but not in the DefaultsSpecification(user) wrapper; and it is taken from the DefaultsSpecification(company) wrapper when it is defined there but not in the DefaultsSpecification(group) and DefaultsSpecification(user) wrappers. The final default value is the built-in default value if it is not specified in any of the three DefaultsSpecification wrappers.

You can refer to the read_config_data command to learn about loading in configuration data. You can refer to the get_defaults_value command to learn about accessing the specified defaults values from your dofile. If you prefer, you can also use the "set_defaults_value -policy" command to define your different policies instead of using configuration files.

Once created with the set_defaults_value command, the properties defined in the DefaultsSpecification wrapper are copied to the DftSpecification wrapper by the create_dft_specification command or the PatternsSpecification wrapper by the create_patterns_specification command.

Arguments

None.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DefaultsSpecification/DftSpecification

A wrapper that specifies options that define the default values of the properties in the DftSpecification wrapper.

Usage

```
DefaultsSpecification(policy) { //Legal: company | group | user
  DftSpecification {
    mini_occ_capture_trigger     :  shift_en | capture_en; // default:
                                                              shift_en
    rtl_extension                : string ;  // default: v
    gate_extension               : string ;  // default: vg
    reuse_modules_when_possible  : boolean ; // default: auto
    use_rtl_cells                : boolean ; // default: auto
    use_rtl_clock_dff_cell       : on | off | auto ;
    use_rtl_clock_shaper_cell    : on | off | auto ;
    dft_cell_selection_name      : string ;
    persistent_clock_cell_prefix : string ;
                                   //default: tessent_persistent_cell_
    persistent_cell_prefix       : string ;
                                   //default: tessent_persistent_cell_
    use_synchronizer_cell_with_reset : on | off ;
    use_rtl_synchronizer_cell    : on | off | auto ;
    independent_insertion_flow   : on | off | auto ;
    IjtagNetwork {
    }
    BoundaryScan {
    }
    MemoryBist {
    }
    MemoryBisr {
    }
  }
}
```

Description

A wrapper that specifies options that define the default values of the properties in the DftSpecification wrapper. These wrappers are used to define how the pattens are to be created when running the create_dft_specification command to create different pattern types.

Arguments

- mini_occ_capture_trigger : *string* ;

  A string that specifies the default value for the corresponding property found inside the DftSpecification wrapper when the create_dft_specification command is used.

- rtl_extension : *string* ;

  A string that specifies the default value for the corresponding property found inside the DftSpecification wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- gate_extension : *string* ;

  A string that specifies the default value for the corresponding property found inside the DftSpecification wrapper.

- reuse_modules_when_possible : *boolean* ;

  A string that specifies the default value of the corresponding property found inside the DftSpecification wrapper.

- use_rtl_cells : *boolean* ; // default: auto

  A property that sets the default value of the use_rtl_cells property found inside the DftSpecification wrapper when the create_dft_specification command is used.

- use_rtl_clock_dff_cell : on | off | auto ;

  A property that sets the default value of the use_rtl_clock_dff_cell property found inside the DftSpecification wrapper when the create_dft_specification command is used.

- use_rtl_clock_shaper_cell : on | off | auto ;

  A property that sets the default value of the use_rtl_clock_shaper_cell property found inside the DftSpecification wrapper when the create_dft_specification command is used. This property is used only by the ScanHost wrappers.

- dft_cell_selection_name : *string* ;

  A property that sets the default value of the dft_cell_selection_name property found inside the DftSpecification wrapper when the create_dft_specification command is used.

- persistent_clock_cell_prefix : *string* ; // default: tessent_persistent_cell_

  A property that sets the default value of the persistent_clock_cell_prefix property found inside the DftSpecification wrapper when the create_dft_specification command is used.

- persistent_cell_prefix : *string* ; // default: tessent_persistent_cell_

  A property that sets the default value of the persistent_cell_prefix property found inside the DftSpecification wrapper when the create_dft_specification command is used.

- use_rtl_synchronizer_cell : on | off | auto ;

  A property that sets the default value of the use_rtl_synchronizer_cell property found inside the DftSpecification wrapper when the create_dft_specification command is used. This property is used only by the OCC and the MemoryBist wrappers.

- use_synchronizer_cell_with_reset : on | off ;

  A property that sets the default value of the use_synchronizer_cell_with_reset property found inside the DftSpecification wrapper when the create_dft_specification command is used.

- independent_insertion_flow : on | off | auto ;

  A property that sets the default value of the independent_insertion_flow property found inside the DftSpecification wrapper when the create_dft_specification command is used.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example shows a GlobalOptions wrapper that specifies the rtl_extension as “vb” and the gate_extension as ”v” inside the group defaults policy. Because it is not overridden by a user defaults policy wrapper, those default values are used by the create_dft_specification command.

```
read_config_data -from_string {
DefaultsSpecification(user) {
   DftSpecification {
     rtl_extension  : vb ;
     gate_extension : v ;
   }
  }
}
get_defaults_value DftSpecification/rtl_extension
```

```
vb
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DynamicDftSignals

A wrapper that is used to specify the addition of an additional scan cell to control certain dynamic DFT signals.

Usage

```
DefaultsSpecification(policy) { //Legal: company | group | user
  DftSpecification {
    DynamicDftSignals {
      AsyncSetResetDynamicDisable {
        add_holding_scan_cell : on | off ;  // default: off
      }
    }
  }
}
```

Description

Tessent can insert static and dynamic DFT signals to control various aspects of the DFT logic.

You can also use the -add_hold_scan_cell {on | off | auto} argument of the add_dft_signals command to specify whether or not additional control logic (scan cell) is added to the supported dynamic DFT signals to improve the test coverage rates.

Arguments

- AsyncSetResetDynamicDisable/add_holding_scan_cell : on | off;

  A property that defines whether additional logic is added to control the async_set_reset_dynamic_disable dynamic DFT signal.

Examples

The following wrappers define custom values of -add_hold_scan_cell option that serve as the default for the entire company:

```
DefaultsSpecification(company) {
  DftSpecification {
    DynamicDftSignals  {
      AsyncSetResetDynamicDisable {
        add_holding_scan_cell : on
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

StaticDftSignals

A wrapper that is used to specify the addition of an additional scan cell to control certain static DFT signals.

Usage

```
DefaultsSpecification(policy) { //Legal: company | group | user
  DftSpecification {
    StaticDftSignals {
      XBoundingEn {
        add_holding_scan_cell : on | off ;  // default: off
      }
      McpBoundingEn {
        add_holding_scan_cell : on | off ;  // default: off
      }
      ControlTestPointEn {
        add_holding_scan_cell : on | off ;  // default: off
      }
      ObserveTestPointEn {
        add_holding_scan_cell : on | off ;  // default: off
      }
    }
  }
}
```

Description

Tessent can insert static and dynamic DFT signals to control various aspects of the DFT logic. You can create static DFT signals for the following purposes:

- To control global resources such as clocking and power management circuitry.
- To configure various aspects of the circuit during logic test modes.
- To configure the scan chains into the various scan configurations during logic test modes.

You can also use the -add_hold_scan_cell {on | off | auto} argument of the add_dft_signals command to specify whether or not additional control logic (scan cell) is added to the supported static DFT signals to improve the test coverage rates.

Arguments

- XBoundingEn/add_holding_scan_cell : on | off;

  A property that defines whether additional logic is added to control the x_bounding_en static DFT signal.

- McpBoundingEn/add_holding_scan_cell : on | off;

  A property that defines whether additional logic is added to control the mcp_bounding_en static DFT signal.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ControlTestPointEn/add_holding_scan_cell : on | off;

  A property that defines whether additional logic is added to control the control_test_point_en static DFT signal.

- ObserveTestPointEn/add_holding_scan_cell : on | off;

  A property that defines whether additional logic is added to control the observe_test_point_en static DFT signal.

Examples

The following wrappers define custom values of -add_hold_scan_cell option that serves as the default for the entire company:

```
DefaultsSpecification(company) {
  DftSpecification {
    StaticDftSignals  {
      XBoundingEn {
        add_holding_scan_cell : on
      }
      McpBoundingEn {
        add_holding_scan_cell : on
      }
      ControlTestPointEn {
        add_holding_scan_cell : on
      }
      ObserveTestPointEn {
        add_holding_scan_cell : on
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

IjtagNetwork

A wrapper that is used to specify the default values found in the DftSpecification wrapper.

Usage

```
DefaultsSpecification(policy) { // Legal: company | group | user
  DftSpecification {
    IjtagNetwork {
      connect_data_signals_already_connected_to_ports : on | off ;
      to_scan_in_feedthrough_on_pb_sibs : none | buffer | pipeline ;
      ImplementationOptions {
        scan_path_retiming : latch | flop ;
      }
      HostScanInterface {
        Ijtag {
          tck             : port_name ;  // default: ijtag_tck
          reset           : port_name ;  // default: ijtag_reset
          select          : port_name ;  // default: ijtag_sel
          shift_en        : port_name ;  // default: ijtag_se
          capture_en      : port_name ;  // default: ijtag_ce
          update_en       : port_name ;  // default: ijtag_ue
          scan_in         : port_name ;  // default: ijtag_si
          scan_out        : port_name ;  // default: ijtag_so
        }
        STap {
          tdi             : port_name ;  // default: stap_tdi;
          tdo             : port_name ;  // default: stap_tdo;
          tms             : port_name ;  // default: stap_tms;
          tck             : port_name ;  // default: stap_tck;
          trst            : port_name ;  // default: stap_trst;
          reset           : port_name ;  // default: stap_ijtag_reset;
          capture_en      : port_name ;  // default: stap_ijtag_ce;
          shift_en        : port_name ;  // default: stap_ijtag_se;
          update_en       : port_name ;  // default: stap_ijtag_ue;
        }
        Tap {
          tck             : port_name ;  // default: tck
          tdi             : port_name ;  // default: tdi
          tdo             : port_name ;  // default: tdo
          tms             : port_name ;  // default: tms
          trst            : port_name ;  // default: trst
          tdi_local       : port_name ;  // default: (none)
        }
        Tpsp {
          spio            : port_name ;  // default: spio
          spclk           : port_name ;  // default: spclk
        }
      }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
        SecondaryHostScanInterface {
          Ijtag {
            to_tck          : port_name ;  // default: %s_ijtag_to_tck
            to_scan_in      : port_name ;  // default: %s_ijtag_to_si
            from_scan_out   : port_name ;  // default: %s_ijtag_from_so
            to_select       : port_name ;  // default: %s_ijtag_to_sel
            to_capture_en   : port_name ;  // default: %s_ijtag_to_ce
            to_shift_en     : port_name ;  // default: %s_ijtag_to_se
            to_update_en    : port_name ;  // default: %s_ijtag_to_ue
            to_reset        : port_name ;  // default: %s_ijtag_to_reset
          }
          STap {
            to_tdi          : port_name ;  // default: %s_to_tdi;
            from_tdo        : port_name ;  // default: %s_from_tdo;
            to_tms          : port_name ;  // default: %s_to_tms;
            to_tck          : port_name ;  // default: %s_to_tck;
            to_trst         : port_name ;  // default: %s_to_trst;
            to_reset        : port_name ;  // default: %s_ijtag_to_reset;
            to_capture_en   : port_name ;  // default: %s_ijtag_to_ce;
            to_shift_en     : port_name ;  // default: %s_ijtag_to_se;
            to_update_en    : port_name ;  // default: %s_ijtag_to_ue;
          }
          Tap {
            to_tck          : port_name ;  // default: %s_to_tck
            to_tdi          : port_name ;  // default: %s_to_tdi
            from_tdo        : port_name ;  // default: %s_from_tdo
            to_tms          : port_name ;  // default: %s_to_tms
            to_trst         : port_name ;  // default: %s_to_trst
          }
        }
      }
    }
  }
```

Description

The IjtagNetwork default configuration wrapper specifies default values of properties found in the DftSpecification/IjtagNetwork wrapper. You can control some options to guide the creation of the DftSpecification using the create_dft_specification command. The following wrappers specify the default values of the different host interface types supported in the DftSpecification:

- The HostScanInterface/Ijtag wrapper defines default port names for the IJTAG interface. Use an IJTAG interface when you set the design level to “physical_block,” “sub_block,” or “instrument_block” with the set_design_level command.
- The HostScanInterface/Tap wrapper defines default port names for the TAP interface. Use a TAP interface when you set the design level to “chip” using the set_design_level command, and you can request a TAP interface at another design level using the command:

  set_dft_specification_requirements -host_scan_interface_type tap

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- The HostScanInterface/Tpsp wrapper defines default port names for the TPSP interface. You must request the TPSP interface using the command:

  set_dft_specification_requirements -host_scan_interface_type tpsp

- The SecondaryHostScanInterface/Ijtag wrapper defines default port names for secondary IJTAG interfaces.
- The SecondaryHostScanInterface/STap wrapper defines default port names for a secondary STAP interface. Such interfaces are a SecondaryHostScanInterface instantiated within a HostScanInterface containing at least one STap or STapPipeline node, or a SecondaryHostScanInterface instantiated into the Tap/HostSTap wrapper.
- The SecondaryHostScanInterface/Tap wrapper defines default port names for secondary TAP interfaces. Such interfaces are a SecondaryHostScanInterface instantiated within a HostScanInterface containing at least one Tap wrapper, or a SecondaryHostScanInterface instantiated within an STap node.

Arguments

- connect_data_signals_already_connected_to_ports : on | off ;

  A property that defines how the create_dft_specification command should deal with IJTAG instances that have "set_ijtag_instance_options -connect_data_signals_already_connected_to_ports" option set to auto.

- to_scan_in_feedthrough_on_pb_sibs : none | buffer | pipeline ;

  A property that specifies the to_scan_in_feedthrough of the SIBs added by the create_dft_specification command when it adds host physical_block designs to the network. Refer to the property "to_scan_in_feedthrough" in the Sib configuration wrapper for a description of the values.

- ImplementationOptions/scan_path_retiming : latch | flop ;

  A property that specifies if the scan path retiming element is to be implemented using a latch or a flop.

- HostScanInterface/Ijtag/<*port_function*> : *port_name* ;

  A set of properties that define the default values for the properties found inside the HostScanInterface/Interface wrapper of the DftSpecification/IjtagNetwork wrapper. The create_dft_specification command applies the values to the created specification for an IJTAG host scan interface.

- HostScanInterface/STap/<*port_function*> : *port_name* ;

  A set of properties that define the default values for the properties found inside the HostScanInterface/Interface wrapper of the DftSpecification/IjtagNetwork wrapper. The create_dft_specification command applies the values to the created specification for an STAP host scan interface. An STAP host scan interface is a HostScanInterface wrapper that contains at least one STap or STapPipeline node.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- HostScanInterface/Tap/<*port_function*> : *port_name* ;

  A set of properties that define the default values for the properties found inside the HostScanInterface/Interface wrapper of the DftSpecification/IjtagNetwork wrapper. The create_dft_specification command applies the values to the created specification for a TAP host scan interface.

- HostScanInterface/Tpsp/<*port_function*> : *port_name* ;

  A set of properties that define the default values for the properties found inside the HostScanInterface/Interface wrapper of the DftSpecification/IjtagNetwork wrapper. The create_dft_specification command applies the values to the created specification for a TPSP host scan interface.

- SecondaryHostScanInterface/Ijtag/<*port_function*> : *port_name* ;

  A set of properties that define the default values for the properties found inside the HostScanInterface/SecondaryHostScanInterface wrapper of the DftSpecification/IjtagNetwork wrapper. The create_dft_specification command applies the values to the created specification for a secondary IJTAG host scan interface.

- SecondaryHostScanInterface/STap/<*port_function*> : *port_name* ;

  A set of properties that define the default values for the properties found inside the HostScanInterface/STap/SecondaryHostScanInterface wrapper of the DftSpecification/IjtagNetwork wrapper. The create_dft_specification command applies the values to the created specification for a secondary STAP host scan interface. The "create_dft_specification -stap_host_list" switch creates the STAP SecondaryHostScanInterface within the HostScanInterface if it contains at least one STap or STapPipeline node. Example 1 and Example 2 in "Tap/HostSTap" illustrates the use of this switch.

- SecondaryHostScanInterface/Tap/<*port_function*> : *port_name* ;

  A set of properties that define the default value for the properties found inside the HostScanInterface/SecondaryHostScanInterface wrapper of the DftSpecification/IjtagNetwork wrapper. The create_dft_specification command applies the values to the created specification for a secondary TAP host scan interface. Such interfaces are a SecondaryHostScanInterface instantiated within a HostScanInterface containing at least one Tap wrapper, or a SecondaryHostScanInterface instantiated within an STap node.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following wrappers define a custom naming convention for the IJTAG ports that serve as the default for the entire company:

```
DefaultsSpecification(company) {
  DftSpecification {
    IjtagNetwork {
      HostScanInterface {
        Ijtag {
          tck             : my_tck ;
          reset           : my_reset ;
          select          : my_select ;
          shift_en        : my_shift_en ;
          capture_en      : my_capture_en ;
          update_en       : my_update_en ;
          scan_in         : my_scan_in ;
          scan_out        : my_scan_out ;
        }
        Tap {
          tck             : my_tck ;
          tdi             : my_tdi ;
          tdo             : my_tdo ;
          tms             : my_tms ;
          trst            : my_trst ;
        }
        Tpsp {
          spio            : my_spio ;
          spclk           : my_spclk ;
        }
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BoundaryScan

Specifies certain default values for the boundary scan chain and embedded boundary scan chain to build and optionally insert into the design.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
  DftSpecification {
    BoundaryScan {
      outputs_per_enable_cell          : int ;    // default: 16
      max_segment_length_for_logictest : int | unlimited | auto ;
      tck_period                       : period ; // default: 100ns
      create_ac_control                : auto | on ;
      default_cell_on_bidir_port       : combined | separated ;
      Interface  {
      }
      ImplementationOptions {
      }
    }
  }
}
```

Description

A wrapper that specifies certain default values for the both boundary scan and embedded boundary scan. All options apply to both, so is the ImplementationOptions wrapper. Only the Interface wrapper applies only to embedded boundary scan.

Arguments

- outputs_per_enable_cell : *int* ;

  An optional property that instructs the tool to limit the number of output pins controlled by a given output enable cell. The default value of Tessent BoundaryScan is 16. Setting a value in the DefaultsSpecification implies new default values to be used for the respective property of both, the DftSpecification/BoundaryScan and DftSpecification/EmbeddedBoundaryScan, respectively.

- max_segment_length_for_logictest : *int* | unlimited | auto ;

  An optional property that instructs the tool to segment the boundary scan chains into smaller segments to be reused by scan-based logictest modes. The default value of Tessent BoundaryScan is auto. Setting a value in the DefaultsSpecification implies new default values to be used for the respective property of both, the DftSpecification/BoundaryScan and DftSpecification/EmbeddedBoundaryScan, respectively.

- tck_period : *period* ;

  An optional property that instructs the tool to use a different default period of the test clock. The default value of Tessent BoundaryScan period is 100ns. Setting a value in the DefaultsSpecification implies new default values to be used for the respective property of both, the DftSpecification/BoundaryScan and DftSpecification/EmbeddedBoundaryScan, respectively.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- create_ac_control : auto | on ;

  An optional property that instructs the tool to create an interface block to generate the ac_model_en, ac_signal, ac_init_clock1, and ac_init_clock0 signals. When the value of this property is set to on, the tool creates a TAP controller and a dot6 control block. Both are instantiated in the dft_block instance, and the dot6 control block is connected to the TAP controller.

- default_cell_on_bidir_port : combined | separated;

  An optional property that instructs the tool to use either a single boundary scan register (combined) or two independent unidirectional boundary scan registers (separated) for bidirectional ports.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Interface

Specifies certain default values for the embedded boundary scan interface only.

Usage

```
DefaultsSpecification(policy) { //Legal: company | group | user
  DftSpecification {
    BoundaryScan {
      Interface {
        select                  : port_naming ; //bscan_select
        reset                   : port_naming ; //bscan_reset
        force_disable           : port_naming ; //bscan_force_disable
        select_jtag_input       : port_naming ; //bscan_select_jtag_input
        select_jtag_output      : port_naming ;
                                                //bscan_select_jtag_output
        bscan_clock             : port_naming ; //bscan_clock
        capture_en              : port_naming ; //bscan_capture_en
        shift_en                : port_naming ; //bscan_shift_en
        update_en               : port_naming ; //bscan_update_en
        scan_in                 : port_naming ; //bscan_scan_in
        scan_out                : port_naming ; //bscan_scan_out
        auxiliary_output        : port_naming ; //bscan_%s_aux_out
        auxiliary_output_enable : port_naming ; //bscan_%s_aux_out_en
        auxiliary_input         : port_naming ; //bscan_%s_aux_in
        auxiliary_input_enable  : port_naming ; //bscan_%s_aux_in_en
        ac_init_clock0          : port_naming ; //bscan_ac_init_clk0
        ac_init_clock1          : port_naming ; //bscan_ac_init_clk1
        ac_signal               : port_naming ; //bscan_ac_signal
        ac_mode_en              : port_naming ; //bscan_ac_mode_en
      }
    }
  }
}
```

Description

A wrapper that specifies certain default values for the embedded boundary scan interface to be used. Using this wrapper you can override the default naming convention of the Tessent BoundaryScan tool's embedded boundary scan DFT specification DftSpecification/EmbeddedBoundaryScan/Interface.

Arguments

- select : *port_naming* ;

  An optional property that defines the name of the select signal. The select signal is used to enable or disable the embedded boundary scan interface of the core. When not specified, the port is named "bscan_select". Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- reset : *port_naming* ;

  An optional property used to provide the name for the port carrying the reset function. The reset function is active low. When not specified, the port is named "bscan_reset". Setting

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- force_disable : *port_naming* ;

  An optional property used to provide the name for the port carrying the force_disable function. When not specified, the port is named “bscan_force_disable”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- select_jtag_input : *port_naming* ;

  An optional property used to provide the name for the port carrying the select_jtag_input function. When not specified, the port is named “bscan_select_jtag_input”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- select_jtag_output : *port_naming* ;

  An optional property used to provide the name for the port carrying the select_jtag_output function. When not specified, the port is named “bscan_select_jtag_output”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- bscan_clock : *port_naming* ;

  An optional property used to provide the name for the port carrying the bscan_clock function. When not specified, the port is named “bscan_clock”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- capture_en : *port_naming* ;

  An optional property used to provide the name for the port carrying the capture_en function. When not specified, the port is named “bscan_capture_en”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- shift_en : *port_naming* ;

  An optional property used to provide the name for the port carrying the shift_en function. When not specified, the port is named “bscan_shift_en”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- update_en : *port_naming* ;

  An optional property used to provide the name for the port carrying the update_en function. When not specified, the port is named “bscan_update_en”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- scan_in : *port_naming* ;

  An optional property used to provide the name for the port carrying the scan_in function. When not specified, the port is named “bscan_scan_in”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- scan_out : *port_naming* ;

  An optional property used to provide the name for the port carrying the scan_out function. When not specified, the port is named “bscan_scan_out”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- auxiliary_output : *port_naming* ;

  An optional property used to provide the name for the port carrying the auxiliary_output function. When not specified, the port is named “bscan_%s_aux_out”. The presence of the %s symbol somewhere in the string is required. The symbol %s is replaced by the port_name the auxiliary output is associated with. If the port is an element of a bus, the opening bracket is replaced by an underscore. For example, the default name for the auxiliary output port associated with port GPIO[4] is “bscan_GPIO_4_aux_out”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- auxiliary_output_enable : *port_naming* ;

  An optional property used to provide the name for the port carrying the auxiliary_output_enable function. When not specified, the port is named “bscan_%s_aux_out_en”. The presence of the %s symbol somewhere in the string is required. The symbol %s is replaced by the port_name the auxiliary output enable is associated with. If the port is an element of a bus, the opening bracket is replaced by an underscore. For example, the default name for the auxiliary output enable port associated with port GPIO[4] is “bscan_GPIO_4_aux_out_en”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- auxiliary_input : *port_naming*, ... ;

  An optional property used to provide the name for the port carrying the auxiliary_input function. When not specified, the port is named “bscan_%s_aux_in”. The presence of the %s symbol somewhere in the string is required. The symbol %s is replaced by the port_name the auxiliary input is associated with. If the port is an element of a bus, the opening bracket is replaced by an underscore. For example, the default name for the auxiliary input port associated with port GPIO[4] is “bscan_GPIO_4_aux_in”. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- auxiliary_input_enable : *port_naming*, ... ;

  An optional property used to provide the name for the port carrying the auxiliary_input_enable function. When not specified, the port is named

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

"bscan_%s_aux_in_en". The presence of the %s symbol somewhere in the string is required. The symbol %s is replaced by the port_name the auxiliary output enable is associated with. If the port is an element of a bus, the opening bracket is replaced by an underscore. For example, the default name for the auxiliary input enable port associated with port GPIO[4] is "bscan_GPIO_4_aux_in_en". Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- ac_init_clock0 : *port_naming* ;

  An optional property used to provide the name for the port carrying the ac_init_clock0 function, which has a low "off" state. When not specified, the port is named "bscan_ac_init_clk0". This port is only created when AC pads are present. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- ac_init_clock1 : *port_naming* ;

  An optional property used to provide the name for the port carrying the ac_init_clock1 function, which has a high "off" state. When not specified, the port is named "bscan_ac_init_clk1". This port is only created when AC pads are present. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- ac_signal : *port_naming* ;

  An optional property used to provide the name for the port carrying the ac_signal function. When not specified, the port is named "bscan_ac_signal". This port is only created when AC pads are present. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

- ac_mode_en : *port_naming* ;

  An optional property used to provide the name for the port carrying the ac_mode_en function. When not specified, the port is named "bscan_ac_mode_en". This port is only created when AC pads are present. Setting this value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/EmbeddedBoundaryScan/Interface.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ImplementationOptions

Specifies implementation options when building the boundary-scan cells.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
  DftSpecification(module_name,id) {
    BoundaryScan {
      ImplementationOptions  {
        clocking            : tck | gated_tck | gated_tck_inv ; //*DefSpec
        update_stage        : flop | latch ;                    //*DefSpec
        scan_path_retiming  : flop | latch ;                    //*DefSpec
      }
    }
  }
}
```

Description

A wrapper that specifies implementation options when building the boundary-scan cells. The options consist of the clocking, and the use of latches or flip-flops as an update stage and for scan path retiming.

Arguments

- clocking : tck | gated_tck | gated_tck_inv ;

  An optional property that specifies how the clocking of the boundary-scan cell should be constructed. The default value of Tessent BoundaryScan is gated_tck. Setting a value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/BoundaryScan/ImplementationOptions.

- update_stage : flop | latch ;

  An optional property that specifies if the update stage is to be implemented using a latch or a flop. The default value of Tessent BoundaryScan is latch. Setting a value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/BoundaryScan/ImplementationOptions.

- scan_path_retiming : flop | latch ;

  An optional property that specifies if the scan path retiming element is to be implemented using a latch. The default value of Tessent BoundaryScan is latch. Setting a value in the DefaultsSpecification implies a new default value to be used for the respective property of DftSpecification/BoundaryScan/ImplementationOptions.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

MemoryBisr

Specifies certain default values for the memoryBISR to build and optionally insert into the design.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
  DftSpecification {
    MemoryBisr {
      insert_fuse_box_controller_in_sub_block : boolean ; // default: off
      insert_fuse_box_controller_in_chip      : boolean ; // default: on
      insert_fuse_box_controller_in_physical_block : boolean ;
                                                          // default: off
      max_fuse_box_programming_sessions       : int | unlimited ;
                                                          // default: 1
      fuse_box_location                       : internal | external ;
      ChainInterface {
        scan_in              : port_naming ; // default: %s_bisr_si
        parallel_in          : port_naming ;
                                   // default : %s_bisr_parallel_in
        scan_out             : port_naming ; // default: %s_bisr_so
        serial_repair_enable: port_naming ;
                                 // default : %s_bisr_serial_repair_enable
        capture_shift_clock : port_naming ; // default: %s_bisr_clk
        shift_en             : port_naming ; // default: %s_bisr_shift_en
        reset                : port_naming ; // default: %s_bisr_reset
        memory_disable       : port_naming ;
                                     // default: %s_bisr_mem_disable
        memory_chain_select : port_naming ;
                                     // default: %s_bisr_mem_chain_select
      }
      SecondaryHostChainInterface {
        to_scan_in              : port_pin_name ;//default: %s_bisr_to_si
        from_scan_out           : port_pin_name ;
                                  //default: %s_bisr_from_so
        to_capture_shift_clock : port_pin_name ;//default: %s_bisr_to_clk
        to_shift_en             : port_pin_name ;
                                  //default: %s_bisr_to_shift_en
        to_reset                : port_pin_name ;
                                  //default: %s_bisr_to_reset
        to_memory_disable       : port_pin_name ;
                                  //default: %s_bisr_to_memory_disable
        to_memory_chain_select : port_pin_name ;
                                  //default: %s_bisr_to_mem_chain_select
      }
    }
  }
}
```

Description

A wrapper that specifies certain default values for the memoryBISR to build and optionally insert into the design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- insert_fuse_box_controller_in_chip : on | off | when_fuse_box_present ;
  insert_fuse_box_controller_in_sub_block : on | off | when_fuse_box_present ;
  insert_fuse_box_controller_in_physical_block : on | off | when_fuse_box_present ;

  Three optional properties that instruct the tool when to add a BISR controller based on the design level during create_dft_specification. The default is “on” for the chip level and “off” for both the sub block and physical block. When you specify “on”, the tool inserts a MemoryBisr/Controller wrapper in the DftSpecification, which will insert a BISR controller inside the design during process_dft_specification. By default, the tool adds a BISR controller at the chip level where all BISR chains converge and where the fuse box will be instantiated. You can insert the BISR controller and fuse box interface at the physical block or sub-block level if this is highest design hierarchy where all BISR chains converge or where the BISR controller is required.

  The value of when_fuse_box_present allows the tool to add a BISR controller when there is actually a fuse box instance that matches a .tcd_fbox found inside the elaborated design. If no fuse box instance is found, the tool will not add the controller.

  There is no equivalent to these properties in the DftSpecification. If you want to change the BISR controller location, you must use these properties in the DefaultsSpecification.

- fuse_box_location : internal | external ;

  An optional property that specifies if the fuse box is to be instantiated inside the BISR controller, or if it already exists in the design and connections should be made to and from it. The default value of fuse_box_location is internal. Setting a value in the DefaultsSpecification implies a new default value to be used for the respective property of the DftSpecification/MemoryBisr/Controller wrapper.

- max_fuse_box_programming_sessions : *int* | unlimited ;

  An optional property that specifies how many fuse box programming sessions are to be supported by the controller. The default value of max_fuse_box_programming_sessions is 1. The unlimited value disables the compression of repair information. Setting a value in the DefaultsSpecification implies a new default value to be used for the respective property of the DftSpecification/MemoryBisr/Controller wrapper. Refer to the description of this property in the DftSpecification for a more detailed description.

- ChainInterface

  An optional wrapper that specifies the names of the ports created when connecting the BISR chains to ports of a module of type sub_block or physical_block as specified using the set_design_level command. A detailed description of the individual ports can be found at the DftSpecification/MemoryBisr/Interface wrapper.

  The “%s” symbol is required at the start or end of the port name string. It is replaced by the power domain name followed or preceded by an underscore when the module contains power domain names other than “-”. When the power domain name is “-”, the “%s” is replaced by an empty string and the adjacent leading or trailing underscore character is removed from the port name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ChainInterface/scan_in : *port_naming* ;

  A string that specifies the naming to use for the scan_in port. When unspecified, it defaults to the value specified in the MemoryBisr/Interface/scan_in property inside the DftSpecification wrapper. See the get_defaults_value command description for more details about the company, group, and user defaults specifications.

- ChainInterface/parallel_in : *port_naming* ;

  A string that specifies the naming to use for the BISR chain parallel input port. This property is only effective when you enable fast BISR loading by specifying DftSpecification/MemoryBisr/memory_repair_loading_method: from_read_buffer, otherwise it is ignored. For more information, refer to "Fast BISR Loading" in *Tessent MemoryBIST User's Manual For Use With Tessent Shell*. When parallel_in is unspecified, it defaults to %s_bisr_parallel_in.

- ChainInterface/scan_out : *port_naming* ;

  A string that specifies the naming to use for the scan_out port. When unspecified, it defaults to the value specified in the MemoryBisr/Interface/scan_out property inside the DftSpecification wrapper. See the get_defaults_value command description for more details about the company, group, and user defaults specifications.

- ChainInterface/serial_repair_enable : *port_naming* ;

  A string that specifies the naming to use for the fast BISR chain loading enable port. This property is only effective when DftSpecification/MemoryBisr/ memory_repair_loading_method: from_read_buffer has been specified, otherwise it is ignored. This port is connected to the "SerialRepairEnable" output port of the BISR controller. When unspecified, it defaults to "%s_bisr_serial_repair_enable".

- ChainInterface/capture_shift_clock : *port_naming* ;

  A string that specifies the naming to use for the capture_shift_clock port. When unspecified, it defaults to the value specified in the MemoryBisr/Interface/capture_shift_clock property inside the DftSpecification wrapper. See the get_defaults_value command description for more details about the company, group, and user defaults specifications.

- ChainInterface/shift_en : *port_naming* ;

  A string that specifies the naming to use for the shift_en port. When unspecified, it defaults to the value specified in the MemoryBisr/Interface/shift_en property inside the DftSpecification wrapper. See the get_defaults_value command description for more details about the company, group and user defaults specifications.

- ChainInterface/reset : *port_naming* ;

  A string that specifies the naming to use for the active low reset port. When unspecified, it defaults to the value specified in the MemoryBisr/Interface/reset property inside the DftSpecification wrapper. See the get_defaults_value command description for more details about the company, group and user defaults specifications.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ChainInterface/memory_disable : *port_naming* ;

  A string that specifies the naming to use for the memory_disable port. When unspecified, it defaults to the value specified in the MemoryBisr/Interface/memory_disable property inside the DftSpecification wrapper. See the get_defaults_value command description for more details about the company, group and user defaults specifications.

  This port is only created when at least one memory instance has a Memory wrapper (see "Tessent Core Description" on page 4793) with function "bisrSerialData" and a Memory with function "select" in its Memory library description or a child block with this port is present. Such ports are identified in the ICL module as DataInPorts with the Attribute "tessent_bisr_function" set to "memory_disable".

- ChainInterface/memory_chain_select : *port_naming* ;

  A string that specifies the naming to use for the BISR register chain select MSEL port. This port is connected to the "bisrSeIM" output port of the BISR controller. When unspecified, it defaults to "%s_bisr_mem_chain_select".

- SecondaryHostChainInterface

  An optional wrapper that specifies the default port names of the BISR scan interface at the boundary of the current design when you implement the tiling design flow with BISR. The default port names you specify for the properties are used for the corresponding properties in the DftSpecification SecondaryHostChainInterface wrapper.

  The "%s" symbol, specified anywhere within the port name string, is a required symbol within the port name. When you specify the SecondaryHostChainInterface[*interface_name*] declaration in the BISR Segment Order File, the tool creates a SecondaryHostChainInterface wrapper in the DftSpecification that applies to the BISR scan interface for the current design. By default, the tool replaces the "%s" symbol in the port name with the "*interface_name*" string and follows or precedes it with an underscore.

- SecondaryHostChainInterface/to_scan_in : *port_pin_name* ;

  An optional property that specifies the default naming to use for the to_scan_in port in the MemoryBisr/SecondaryHostChainInterface wrapper of the DftSpecification. When unspecified, it defaults to "%s_bisr_to_si".

- SecondaryHostChainInterface/from_scan_out : *port_pin_name* ;

  An optional property that specifies the default naming to use for the from_scan_out port in the MemoryBisr/SecondaryHostChainInterface wrapper of the DftSpecification. When unspecified, it defaults to "%s_bisr_from_so".

- SecondaryHostChainInterface/to_capture_shift_clock : *port_pin_name* ;

  An optional property that specifies the default naming to use for the to_capture_shift_clock port in the MemoryBisr/SecondaryHostChainInterface wrapper of the DftSpecification. When unspecified, it defaults to "%s_bisr_to_clk".

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- SecondaryHostChainInterface/to_shift_en ; *port_pin_name* ;

  An optional property that specifies the default naming to use for the to_shift_en port in the MemoryBisr/SecondaryHostChainInterface wrapper of the DftSpecification. When unspecified, it defaults to “%s_bisr_to_shift_en”.

- SecondaryHostChainInterface/to_reset : *port_pin_name* ;

  An optional property that specifies the default naming to use for the to_reset port in the MemoryBisr/SecondaryHostChainInterface wrapper of the DftSpecification. When unspecified, it defaults to “%s_bisr_to_reset”.

- SecondaryHostChainInterface/to_memory_disable : *port_pin_name* ;

  An optional property that specifies the default naming to use for the to_memory_disable port in the MemoryBisr/SecondaryHostChainInterface wrapper of the DftSpecification. When unspecified, it defaults to “%s_bisr_to_mem_disable”.

- SecondaryHostChainInterface/to_memory_chain_select : *port_pin_name* ;

  An optional property that specifies the default naming to use for the to_memory_chain_select port in the MemoryBisr/SecondaryHostChainInterface wrapper of the DftSpecification. When unspecified, it defaults to “%s_bisr_to_mem_chain_select”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

MemoryBist

Specifies the default values of properties found in the DftSpecification/MemoryBist wrapper.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
  DftSpecification {
    MemoryBist {
      clock_partitioning      : per_clock_domain | per_sync_clock_group ;
      max_steps_per_controller     : int | unlimited ;
      max_test_time_per_controller : time ;            // default is 500ms
      max_memories_per_step        : int | unlimited ;
      max_power_per_step           : int | unlimited ; // default is 500mW
      single_memory_dimension_per_step : on | off ;
      memory_interface_reuse_method : non_unique_instances_only | never ;
      ControllerOptions {
      }
      RepairOptions {
      }
      DiagnosisOptions {
      }
      DirectAccessOptions {
      }
      BistAccessPortOptions {
      }
      AlgorithmResourceOptions {
      }
      MemoryInterfaceOptions {
      }
      MemoryClusterOptions {
      }
    }
  }
}
```

Description

A wrapper that specifies the default values of properties found in the DftSpecification/MemoryBist wrapper.

You can also provide test scheduling constraints to guide the creation of the DftSpecification using the create_dft_specification command.

Arguments

- clock_partitioning : per_clock_domain | per_sync_clock_group ;

  A property that specifies how memories are grouped by the create_dft_specification command according to their clock sources. When set to per_clock_domain, the memories on the same clock domain are allowed to be assigned to a controller. When set to per_sync_clock_group, controller sharing is allowed for memories belonging to synchronous clock sources and having the same clock frequency. Synchronous clock groupings are specified with the add_synchronous_clock_group command and current groupings can be reported with the report_synchronous_clock_groups command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- max_steps_per_controller : int | unlimited ;

  A property that constrains the number of test steps that are allocated per controller. A controller step designates the group of memories that are tested in parallel. The create_dft_specification command assigns test steps to controllers such that the requirement is met.

  When running the process_dft_specification command, the validation of the DftSpecification issues a warning if the controller step count requirement is exceeded.

- max_test_time_per_controller : time | unlimited ;

  A property that constrains the test time that a controller can use to complete the test. The create_dft_specification command assigns memories to controllers such that the requirement is met. The default value is 500ms.

  The estimated test time of a controller is the sum of the execution time of its controller steps. The test time of a controller step depends on factors such as the chosen algorithm and the memory sizes.

  When running the process_dft_specification command, the validation of the DftSpecification issues a warning if the test time requirement is exceeded.

- max_memories_per_step : int | unlimited ;

  A property that constrains the number of memories that is tested in parallel within a controller step. The create_dft_specification command assigns memories to controller steps such that the requirement is met.

  When running the process_dft_specification command, the validation of the DftSpecification issues a warning if the memory count requirement is exceeded.

- max_power_per_step : int | unlimited ;

  A property that constrains the power consumption per controller step. The create_dft_specification command assigns memories to controller steps such that the requirement is met. The specified value is assumed to be in milliwatts. The default value is 500mW.

  The estimated power per step is the sum of the power consumed by memories tested in parallel. The memory power is calculated by multiplying the MilliWattsPerMegaHertz property in the memory library description by the frequency of the clock domain. It does not include the power consumed by the BIST circuitry.

  When running the process_dft_specification command, the validation of the DftSpecification issues a warning if the power consumption requirement is exceeded.

- single_memory_dimension_per_step : on | off ;

  A property that specifies how memories of different dimensions are grouped by the create_dft_specification command.

  When set to on, only memories having the same number of columns, rows, and banks are allowed to be tested in parallel within a controller step. When set to off, memories of different sizes are allowed to be tested in parallel.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- memory_interface_reuse_method : non_unique_instances_only | never

  A property that specifies how ReusedMemoryInterface wrappers are generated by the create_dft_specification command. The default value of non_unique_instances_only infers a ReusedMemoryInterface wrapper for a memory instance located inside a non-unique design instance.

  When set to never, all memory instances are unique and no ReusedMemoryInterface wrappers are generated. If memory interfaces are located in non-unique design instances, they are uniquified during DFT insertion.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ControllerOptions

Specifies the default values of properties found in the Controller/AdvancedOptions and Controller/Step wrappers of the DftSpecification/MemoryBist wrapper.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
  DftSpecification  {
    MemoryBist {
      ControllerOptions {
        algorithm                          : algo_name ; // default:
                                                         // from library
        operation_set                      : opset_name ;// default:
                                                         // from library
        extra_algorithms                   : algo_name, ... ;
        extra_operation_sets               : opset_name, ... ;
        incremental_test_mode              : on | off ;
        comparator_location                : shared_in_controller |
                                             per_interface ;
        pipeline_controller_outputs        : on | off ;
        selective_parallel_memory_test     : on | off ;
        shared_comparators_per_go_id       : int | all ; // default: 1
        use_multicycle_paths               : on | off | auto;
        use_multicycle_paths_period_threshold : period | none ;
        min_misr_segment_bits              : 24 | 256 ;
        rom_signature_location             : external_straps |
                                 min_address | max_address | off ;
      }
    }
  }
}
```

Description

A wrapper that specifies the default values of properties found in the AdvancedOptions and Step wrappers of the DftSpecification/MemoryBist wrapper.

Arguments

- algorithm : *algo_name* ;

  A property that sets the default value of the algorithm property found inside the AdvancedOptions wrapper of the DftSpecification/MemoryBist wrapper.

- operation_set : *opset_name* ;

  A property that sets the default value of the operation_set property found inside the AdvancedOptions wrapper of the DftSpecification/MemoryBist wrapper.

- extra_algorithms : *algo_name*, ... ;

  A property that sets the default value of the extra_algorithms property found inside the AdvancedOptions wrapper of the DftSpecification/MemoryBist wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- extra_operation_sets : *opset_name*, ... ;

  A property that sets the default value of the extra_operation_sets property found inside the AdvancedOptions wrapper of the DftSpecification/MemoryBist wrapper.

- incremental_test_mode : on | off ;

  A property that sets the default value of the incremental_test_mode property found inside the AdvancedOptions wrapper of the DftSpecification/MemoryBist wrapper

- comparator_location : shared_in_controller | per_interface ;

  A property that sets the default value of the comparator_location property found inside the Step wrapper of the DftSpecification/MemoryBist wrapper.

- pipeline_controller_outputs : on | off ;

  A property that sets the default value of the pipeline_controller_outputs property found inside the AdvancedOptions wrapper of the DftSpecification/MemoryBist wrapper.

- selective_parallel_memory_test : on | off ;

  A property that sets the default value of the selective_parallel_memory_test property found inside the AdvancedOptions wrapper of the DftSpecification/MemoryBist wrapper.

- shared_comparators_per_go_id : *int* | all ;

  A property that sets the default value of the shared_comparators_per_go_id property found inside the AdvancedOptions wrapper of the DftSpecification/MemoryBist wrapper. The default is 1.

- use_multicycle_paths : on | off | auto ;

  A property that sets the default value of the use_multicycle_paths property found inside the AdvancedOptions wrapper of the DftSpecification/MemoryBist wrapper. The default setting of auto sets the DftSpecification AdvancedOptions/use_multicycle_paths on/off value based on the threshold specified with the use_multicycle_paths_period_threshold property. If a threshold is specified as none, the auto setting for use_multicycle_paths resolves to on.

  It is recommended to set this property to on for high-speed controllers only. When set to on, the maximum TCK clock frequency used during memory BIST controller access through the IJTAG network is limited to 50% of the functional clock frequency. This might be disadvantageous for controllers running at relatively low functional frequency with respect to TCK. However, when the property is set to off, any TCK clock frequency can be used for controller access. An additional advantage of setting the property to off is that the number of timing exceptions is significantly reduced. Therefore, the processing of SDC and STA files by the synthesis and layout tools is accelerated for large designs.

- use_multicycle_paths_period_threshold : *period* | none ;

  A property that sets the controller clock period threshold for inferring multicycle paths in the generated SDC and STA scripts. The *period* value is the controller's functional clock period specified in nanoseconds. This property is only valid when the use_multicycle_paths property is set to auto. Multicycle paths are disabled for the controller when its functional

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

clock period is larger than the threshold specified by the *period* value. The default setting of none results in the auto setting for the use_multicycle_paths property resolving to on. It is recommended to set a threshold period corresponding to a relatively high frequency. A reasonable value could be a threshold of 25% of the maximum frequency achievable for the technology used in the design. Refer to the use_multicycle_paths property description to learn the advantages of setting this property to an optimal value.

- min_misr_segment_bits : 24 | 256 ;

  A property that sets the default value of the min_misr_segment_bits property found inside the AdvancedOptions wrapper of the DftSpecification/MemoryBist wrapper.

- rom_signature_location : external_straps | min_address | max_address | off ;

  A property that sets the default value of the rom_signature_location property found inside the AdvancedOptions wrapper of the DftSpecification/MemoryBist wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

RepairOptions

Specifies the default values of properties found in the DftSpecification/MemoryBist/Controller/RepairOptions wrapper.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
  DftSpecification  {
    MemoryBist {
      RepairOptions {
        fuse_set_extraction_sequence      : address_before_fuse_map |
                                            fuse_map_before_address ;
        row_bira_location                 : controller |
                                            follow_comparators ;
        max_repair_group_size             : unlimited |
                                            {int[kilobits | megabits]} ;
        repair_sharing                    : on | off ;
        enable_multicycle_operation       : on | off | auto ;
      }
    }
  }
}
```

Description

A wrapper that specifies the default values of properties found in the DftSpecification/MemoryBist/RepairOptions wrapper.

Arguments

- fuse_set_extraction_sequence : address_before_fuse_map | fuse_map_before_address ;

  A property that sets the default value of the fuse_set_extraction_sequence property found inside the RepairOptions wrapper of the DftSpecification/MemoryBist wrapper.

- row_bira_location : controller | follow_comparators ;

  A property that sets the default value of the row_bira_location property found inside the RepairOptions wrapper of the DftSpecification/MemoryBist wrapper.

- max_repair_group_size : unlimited | {*int*[kilobits | megabits]} ;

  A property that specifies the total memory size limit on a memory group. The default value of “unlimited” indicates that any number of compatible memories can be assigned to repair groups. The *int*[units] pair specifies the maximum size allowed for a repair group. Any memory that would cause this limit to be exceeded is assigned to a different repair group. The unit property of kilobits equates to a scaling multiplier of 2^{10} bits (1024 bits) and megabits equates to 2^{20} bits (1048576 bits).

- repair_sharing : on | off ;

  A property that globally enables or disables repair sharing for memories that have “-repair_sharing auto” specified, which is the default. The “-repair_sharing” value can be

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

changed from “auto” for a memory instance with the set_memory_instance_options command.

- enable_multicycle_operation : on | off | auto ;

  A property that sets the default value of the enable_multicycle_operation property found inside the RepairOptions wrapper of the DftSpecification/MemoryBist wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DiagnosisOptions

Specifies the default values of properties found in the DftSpecification/MemoryBist/Controller/DiagnosisOptions wrapper.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
  DftSpecification {
   MemoryBist {
      DiagnosisOptions {
        comparator_selection_mux       : on | off | auto ;
        go_status                      : auto | per_memory ;
        StopOnErrorOptions {
          failure_limit                : int | off ; // default: 4096
        }
      }
    }
  }
}
```

Description

A wrapper that specifies the default values of properties found in the DftSpecification/MemoryBist/DiagnosisOptions wrapper.

Arguments

- comparator_selection_mux : on | off | auto ;

  A property that sets the default value of the comparator_selection_mux property found inside the DiagnosisOptions wrapper of the DftSpecification/MemoryBist wrapper.

- go_status : auto | per_memory ;

  A property that sets the default value of the go_status property found inside the DiagnosisOptions wrapper of the DftSpecification/MemoryBist wrapper.

- StopOnErrorOptions/failure_limit : *int* | off ;

  A property that sets the default value of the StopOnErrorOptions/failure_limit property found inside the DiagnosisOptions wrapper of the DftSpecification/MemoryBist wrapper. The default value becomes 4096 when the controller tests at least one memory of type SRAM or DRAM. It becomes “off” when the controller tests only memories of type ROM. The error counter is no longer used for ROM diagnosis as it was in the LV flow. Instead the functional debug mode is used to perform ROM diagnosis.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DirectAccessOptions

Specifies the default values of properties for wrappers found within the DftSpecification MemoryBist/Controller/DirectAccessOptions wrapper.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
  DftSpecification {
    MemoryBist {
      DirectAccessOptions {
        ExecutionSelections {
          step                    : on | off ;
          memory                  : on | off ;
          algorithm               : on | off ;
          operation_set           : on | off ;
          configuration_data      : on | off ;
          test_port               : on | off ;
          data_inversion_with_address_bit : on | off ;
        }
      }
    }
  }
}
```

Description

A wrapper that specifies the default values of properties within the DftSpecification MemoryBist/Controller/DirectAccessOptions wrapper. The enclosed ExecutionSelections wrapper is used to configure the optional direct access interface signals on the memoryBIST controller.

Arguments

- ExecutionSelections/step : on | off ;

  A property that sets the default value for the step property found in the Controller/DirectAccessOptions/ExecutionSelections wrapper.

- ExecutionSelections/memory : on | off ;

  A property that sets the default value for the memory property found in the Controller/DirectAccessOptions/ExecutionSelections wrapper.

- ExecutionSelections/algorithm : on | off ;

  A property that sets the default value for the algorithm property found in the Controller/DirectAccessOptions/ExecutionSelections wrapper.

- ExecutionSelections/operation_set : on | off ;

  A property that sets the default value for the operation_set property found in the Controller/DirectAccessOptions/ExecutionSelections wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ExecutionSelections/configuration_data : on | off ;

  A property that sets the default value for the configuration_data property found in the Controller/DirectAccessOptions/ExecutionSelections wrapper.

- ExecutionSelections/test_port : on | off;

  A property that sets the default value for the test_port property found in the Controller/DirectAccessOptions/ExecutionSelections wrapper.

- data_inversion_with_address_bit : on | off ;

  A property that sets the default value for the data_inversion_with_address_bit property found in the Controller/DirectAccessOptions/ExecutionSelections wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

BistAccessPortOptions

Specifies the default values of properties for wrappers found within the DftSpecification MemoryBist BistAccessPort wrapper.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
  DftSpecification  {
    MemoryBist {
      BistAccessPortOptions {
        use_multicycle_paths : on | off ;
        DirectAccessOptions {
          direct_access              : on | off ;
          direct_access_clock_source : common | per_bist_clock_domain ;

          ExecutionSelections {
            controller                    : off | auto ;
            step                          : off | auto ;
            memory                        : off | auto ;
            algorithm                     : off | auto ;
            operation_set                 : off | auto ;
            retention_test_phase          : off | auto ;
            configuration_data            : off | auto ;
            test_port                     : off | auto ;
            data_inversion_with_address_bit : off | auto ;
            bira_enable                   : off | auto ;
            check_repair_needed           : off | auto ;
            preserve_fuse_register_values : off | auto ;


          }
        }
      }
    }
  }
}
```

Description

A wrapper that specifies the default values of properties within the DftSpecification MemoryBist/BistAccessPort wrapper.

Arguments

- use_multicycle_paths : on | off ;

  A property that sets the default value for the use_multicycle_paths property found in the MemoryBist/BistAccessPort/AdvancedOptions wrapper

- DirectAccessOptions/direct_access : on | off ;

  A property that sets the default value for the direct_access property found in the MemoryBist/BistAccessPort/DirectAccessOptions wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- DirectAccessOptions/direct_access_clock_source : common | per_bist_clock_domain ;

  A property that sets the default value for the direct_access_clock_source property found in the MemoryBist/BistAccessPort/DirectAccessOptions wrapper.

- ExecutionSelections/controller : off | auto ;

  A property that sets the default value for the controller property found in the ExecutionSelections wrapper within the MemoryBist/BistAccessPort/DirectAccessOptions wrapper.

- ExecutionSelections/step : off | auto ;

  A property that sets the default value for the step property found in the ExecutionSelections wrapper within the MemoryBist/BistAccessPort/DirectAccessOptions wrapper.

- ExecutionSelections/memory : off | auto ;

  A property that sets the default value for the memory property found in the ExecutionSelections wrapper within the MemoryBist/BistAccessPort/DirectAccessOptions wrapper.

- ExecutionSelections/algorithm : off | auto ;

  A property that sets the default value for the algorithm property found in the ExecutionSelections wrapper within the MemoryBist/BistAccessPort/DirectAccessOptions wrapper.

- ExecutionSelections/operation_set : off | auto ;

  A property that sets the default value for the operation_set property found in the ExecutionSelections wrapper within the MemoryBist/BistAccessPort/DirectAccessOptions wrapper.

- retention_test_phase : off | auto ;

  A property that sets the default value for the retention_test_phase property found in the ExecutionSelections wrapper within the MemoryBist/BistAccessPort/DirectAccessOptions wrapper.

- ExecutionSelections/configuration_data : off | auto ;

  A property that sets the default value for the configuration_data property found in the ExecutionSelections wrapper within the MemoryBist/BistAccessPort/DirectAccessOptions wrapper.

- ExecutionSelections/test_port : off | auto ;

  A property that sets the default value for the test_port property found in the ExecutionSelections wrapper within the MemoryBist/BistAccessPort/DirectAccessOptions wrapper.

- ExecutionSelections/data_inversion_with_address_bit : off | auto ;

  A property that sets the default value for the data_inversion_with_address_bit property found in the ExecutionSelections wrapper within the MemoryBist/BistAccessPort/DirectAccessOptions wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ExecutionSelections/bira_enable : off | auto ;

  A property that sets the default value for the bira_enable property found in the ExecutionSelections wrapper within the MemoryBist/BistAccessPort/DirectAccessOptions wrapper.

- ExecutionSelections/check_repair_needed : off | auto ;

  A property that sets the default value for the check_repair_needed property found in the ExecutionSelections wrapper within the MemoryBist/BistAccessPort/DirectAccessOptions wrapper.

- ExecutionSelections/preserve_fuse_register_values : off | auto ;

  A property that sets the default value for the preserve_fuse_register_values property found in the ExecutionSelections wrapper within the MemoryBist/BistAccessPort/DirectAccessOptions wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

AlgorithmResourceOptions

Specifies the default values of properties found in the DftSpecification/MemoryBist/Controller/AlgorithmResourceOptions wrapper.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
  DftSpecification {
    MemoryBist {
      AlgorithmResourceOptions {
        soft_instruction_count                : int ; // default: 0
        data_register_bits                    : int | auto ;
        counter_a_bits                        : int | auto ;
        delay_counter_bits                    : int | auto ;
        max_data_inversion_address_bit_index  : int | max_index ;
                                                // default: 0
        a_equals_b_command_allowed            : on | off ;
        address_segment_x0_y0_allowed         : on | off ;
        max_x0_segment_bits                   : 1 | auto ;
        max_y0_segment_bits                   : 1 | auto ;
      }
    }
  }
}
```

Description

A wrapper that specifies the default values of properties found in the DftSpecificaion/MemoryBist/AlgorithmResourceOptions wrapper.

Arguments

- soft_instruction_count : *int* ;

  A property that sets the default value of the soft_instruction_count property found inside the AlgorithmResourceOptions wrapper of the DftSpecification/MemoryBist wrapper.

- data_register_bits : int | auto ;

  A property that sets the default value of the data_register_bits property found inside the AlgorithmResourceOptions wrapper of the DftSpecification/MemoryBist wrapper.

- counter_a_bits : int | auto ;

  A property that sets the default value of the counter_a_bits property found inside the AlgorithmResourceOptions wrapper of the DftSpecification/MemoryBist wrapper.

- delay_counter_bits : int | auto ;

  A property that sets the default value of the delay_counter_bits property found inside the AlgorithmResourceOptions wrapper of the DftSpecification/MemoryBist wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- max_data_inversion_address_bit_index : int | max_index ;

  A property that sets the default value of the max_data_inversion_address_bit_index property found inside the AlgorithmResourceOptions wrapper of the DftSpecification/MemoryBist wrapper. The default is 0.

- a_equals_b_command_allowed : on | off ;

  A property that sets the default value of the a_equals_b_command_allowed property found inside the AlgorithmResourceOptions wrapper of the DftSpecification/MemoryBist wrapper.

- address_segment_x0_y0_allowed : on | off ;

  A property that sets the default value of the address_segment_x0_y0_allowed property found inside the AlgorithmResourceOptions wrapper of the DftSpecification/MemoryBist wrapper.

- max_x0_segment_bits : 1 | auto ;

  A property that sets the default value of the max_x0_segment_bits property found inside the AlgorithmResourceOptions wrapper of the DftSpecification/MemoryBist wrapper.

- max_y0_segment_bits : 1 | auto ;

  A property that sets the default value of the max_y0_segment_bits property found inside the AlgorithmResourceOptions wrapper of the DftSpecification/MemoryBist wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

MemoryInterfaceOptions

Specifies the default values of properties found in the Controller/AdvancedOptions Controller/Step, and Controller/Step/MemoryInterface wrappers of the DftSpecification/MemoryBist wrapper.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
  DftSpecification {
    MemoryBist {
      MemoryInterfaceOptions {
        bist_data_in_pipelining        : on | off | int ;
        bist_data_out_pipelining       : on | off | per_port ;
        local_comparators_per_go_id    : int | all ;  // default: 1
        observation_xor_size           : off | 1..MaxPosInt |
                                         all ; // default: 3
        repair_analysis_present        : off  | auto ;
        scan_bypass_logic              : async_mux | none | sync_mux |
                                         from_library ;
        data_bits_per_bypass_signal    : 1..MaxPosInt | all ; default: 1
        repair_group_scope             : physical_memory | controller ;
      }
    }
  }
}
```

Description

A wrapper that specifies the default values of properties found in the AdvancedOptions, Step and Controller/Step/MemoryInterface wrappers of the DftSpecification/MemoryBist wrapper.

Arguments

- bist_data_in_pipelining : on | off | *int* ;

  A property that sets the default value of the of the bist_data_in_pipelining property found inside the Step wrapper of the DftSpecification/MemoryBist wrapper.

- bist_data_out_pipelining : on | off | per_port ;

  A property that sets the default value of the of the bist_data_out_pipelining property found inside the Step wrapper of the DftSpecification/MemoryBist wrapper.

- local_comparators_per_go_id : *int* | all ;

  A property that sets the default value of the MemoryInterface/local_comparators_per_go_id property found inside the Step wrapper of the DftSpecification/MemoryBist wrapper.

- observation_xor_size : off | 1..MaxPosInt | all;

  A property that sets the default value of the observation_xor_size property found inside the AdvancedOptions wrapper of the DftSpecification/MemoryBist/Controller wrapper. The default value is 3.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- repair_analysis_present : off | auto ;

  A property that sets the default value of the MemoryInterface/repair_analysis_present property found inside the Step wrapper of the DftSpecification/MemoryBist wrapper.

- scan_bypass_logic : async_mux | none | sync_mux | from_library ;

  A property that sets the default value of the MemoryInterface/scan_bypass_logic property found inside the Step wrapper of the DftSpecification/MemoryBist wrapper.

- data_bits_per_bypass_signal : 1..MaxPosInt | all ;

  A property that sets the default value of the MemoryInterface/data_bits_per_bypass_signal property found inside the Step wrapper of the DftSpecification/MemoryBist wrapper. The default value is 1.

- repair_group_scope : physical_memory | controller ;

  Memory instances that have repair sharing enabled, the selected value impacts repair grouping as follows:

  - physical_memory — Specifies the repair group is restricted within a physical memory. Memories with multiple RowSegment and/or ColumnSegment wrappers are considered as having a single RowSegment and/or ColumnSegment wrapper for redundancy analysis as long as the memory size does not exceed the size specified by the max_repair_group_size property in the RepairOptions wrapper.

  - controller — Specifies the repair group is restricted to all memories tested by the controller. The repair group cannot contain memories tested by other controllers. Memories are automatically assigned to repair groups as long as the memory size does not exceed the size specified by the max_repair_group_size property in the RepairOptions wrapper.

  Repair sharing can be enabled for memory instances with the set_memory_instance_options command if the DefaultsSpecification for repair_sharing is “off”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

MemoryClusterOptions

A wrapper that is used to specify default values of properties found in the DftSpecification/MemoryBist/Controller/MemoryCluster wrapper

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
  DftSpecification  {
   MemoryBist {
      MemoryClusterOptions {
        pipeline_cluster_inputs       : on | off ;
        pipeline_cluster_outputs      : on | off ;
        memory_access_level           : physical | logical | auto ;
        repair_analysis_present       : auto | off ;
        repair_group_scope            : physical_memory | logical_memory |
                                        controller ;
      }
    }
  }
}
```

Description

A wrapper that specifies the default values of properties found in the DftSpecification/MemoryBist/MemoryCluster wrapper.

Arguments

- pipeline_cluster_inputs : on | off ;

  A property that sets the default value of the pipeline_cluster_inputs property found inside the MemoryCluster wrapper of the DftSpecification/MemoryBist wrapper.

- pipeline_cluster_outputs : on | off ;

  A property that sets the default value of the pipeline_cluster_outputs property found inside the MemoryCluster wrapper of the DftSpecification/MemoryBist wrapper.

- memory_access_level : physical | logical | auto ;

  A property that sets the default value of the memory_access_level property found inside the MemoryCluster wrapper of the DftSpecification/MemoryBist wrapper.

- repair_analysis_present : auto | off ;

  A property that sets the default value of the repair_analysis_present property found inside the MemoryCluster wrapper of the DftSpecification/MemoryBist wrapper.

- repair_group_scope : physical_memory | logical_memory | controller ;

  A property that sets the default value of the repair_group_scope property found inside the MemoryCluster wrapper of the DftSpecification/MemoryBist wrapper.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DefaultsSpecification/PatternsSpecification

Specifies certain default values for the pattern generation process through the PatternsSpecification.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
  PatternsSpecification {
    timeplate     : timeplate_name ;
    tester_period : 100ns ;
    AdvancedOptions {
      procfile_name    : file_path ;
      parameter_file   : file_path ;
    }
    SignOffOptions {
    }
    ManufacturingOptions {
    }
  }
}
```

Description

The tool uses these wrappers to define how it creates the patterns when running the create_patterns_specification command to create different pattern types.

Arguments

- timeplate : *timeplate_name* ;

  A property that specifies the timeplate the tool uses to override the default timeplate created when generating the pattern. The timeplate is loaded from a procfile as specified from the PatternsSpecification/AdvancedOptions/procfile_name property. The timeplate and procfile_name properties are both inherited downwards.

- tester_period : *period* ;

  An optional property that defines the period of the tester clock. The default is 100ns. Setting a value in the DefaultsSpecification implies a new default value to be used for the respective property of PatternsSpecification/Patterns.

- AdvancedOptions/procfile_name : *file_path* ;

  A property that specifies a procfile name that contains a test_setup and/or a test_end procedure.

  The file_path must have one of the following formats:

  - A simple filename with no directory path. In this case, the file must exist beside the ICL file.
  - A null string. In this case, no procfile is used.
  - An absolute path starting with a “/”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

  - A relative path starting with "./" or "../". In this case, the file must exist relative to the current working directory.

- AdvancedOptions/parameter_file : *file_path* ;

  An optional that specifies the name of a file containing parameters used to configure the write_patterns command. Setting a value in the DefaultsSpecification implies a new default value to be used for the respective property of PatternsSpecification.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SignOffOptions

Defines how simulation signoff pattern wrappers are to be created when running the create_patterns_specification command.

Usage

```
DefaultsSpecification(policy) { // legal: company | group | user
  PatternsSpecification {
    SignOffOptions {
      simulate_instruments_in_lower_physical_instances : on | off ;
      testbench_module_name                            : format_string ;
      current_design_view : <enum>   ; // legal : auto full ijtag_graybox
      MemoryBist {
        include_setup_chain_test : on | off ;
      }
      MemoryBisr {
        include_repair_chain_test : on | off ;
      }

    }
  }
}
```

Description

This wrapper is used to define how the simulation signoff patterns wrapper are to be created when running the create_patterns_specification command to create the signoff simulation patterns.

Arguments

- simulate_instruments_in_lower_physical_instances : on | off ;

  A property that specifies to re-execute the lower physical instance simulations at higher levels when you are using a hierarchical flow. This property has no effect if your current design does not include any lower physical instances.

  This property only affects the pattern specification creation when create_patterns_specification is called with usage “signoff”. It has no effect for usage “manufacturing”. When specified to “on”, the pattern specification includes Patterns wrapper to simulate all tests situated in the top-level module and below the child physical blocks. The Patterns wrapper keeps the simulation of the test in each physical block instance separated from the other such that the interface model of the other physical block instances can be used when simulating a given physical block instance in order to manage the simulation performance. If you have declared some modules as “sub_block” modules using the set_design_level command, their tests are always simulated as part of their parent physical module, whether it is the top module or a child physical block. When the property is set to off, only the tests situated in the top module and its child sub_block modules are simulated. The tests situated below the child physical block are omitted.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- testbench_module_name : *format_string* ;

  A property that specifies the top-level module name of the testbench in the *patternName.v* file. You can use this property to give a different testbench module name for each testbench so that more than one testbench can be compiled into a common work library for a Verilog simulator.

  The default is TB. You can specify %d to mean the module name of the current design or %p to mean the pattern name. The advantage of using TB for all testbenches is that you can use the same waveform file to analyze failing simulations in any testbench.

- current_design_view : full | ijtag_graybox | auto ;

  A property that specifies the view for the current design.

  auto — The tool uses the IJTAG graybox view when available for the current design for ICLNetworkVerify patterns. Otherwise, the full view is used.

  You can specify a default value with the "set_defaults_value PatternsSpecification/SignOffOptions/current_design_view <*view*>" command prior to calling create_patterns_specification. Use this to specify "full" to override the "auto" behavior for ICLNetworkVerify patterns.

  full — Force the tool to use the full view for the current design for ICLNetworkVerify patterns.

  ijtag_graybox — Force the tool to use the IJTAG graybox view for the current design for ICLNetworkVerify patterns. If the IJTAG graybox is not available, the tool reports that the full view is used instead.

- MemoryBisr/include_repair_chain_test : on | off ;

  A property that specifies the inclusion of test patterns for the testing of BISR chains and connections to and from the BIRA registers. The default value of off creates patterns to test connections that are necessary to repair memories. When set to on, patterns are generated to test all connections, which is useful to debug data connection issues between BISR and BIRA registers, as well as control signal connection issues.

- MemoryBist/include_setup_chain_test : on | off ;

  A property that specifies the create_patterns_specification command to include test patterns that exercise the setup chains inside the memory BIST controllers and interfaces.

Related Topics

DefaultsSpecification/PatternsSpecification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ManufacturingOptions

Defines how simulation manufacturing pattern wrappers are to be created when running the create_patterns_specification command.

Usage

```
DefaultsSpecification(policy) { // legal: company group user
  PatternsSpecification {
    ManufacturingOptions {
      max_async_clock_sources        : integer | unlimited ;
      manufacturing_patterns_formats : format ;
      compress_pattern_files         : on | off ;
      MemoryBist {
        max_controllers_per_test_step : integer | unlimited ;
        include_setup_chain_test : on | off ;
      }
      MemoryBisr {
        include_repair_chain_test : on | off ;
      }
    }
  }
}
```

Description

The ManufacturingOptions wrapper is used in the Defaults/PatternsSpecification wrapper.

Arguments

- max_async_clock_sources : *integer* | unlimited ;

  A property that specifies how many asynchronous clocks can be active within one Patterns wrapper. Modern testers can generate multiple asynchronous clocks. Use this property to control how many instruments, belonging to different asynchronous frequency groups, can be executed in parallel.

  If you specify an integer, Tessent Shell creates memory BIST patterns that use no more than the specified number of asynchronous clocks. Valid integer values are 0 or greater. When you specify a value of 0, all clocks used in the pattern are synchronous. Use the value of 0 when the tester does not have any asynchronous clock generators.

  If you specify unlimited, Tessent Shell groups all memory BIST controllers inside of a single pattern. Use this value when the tester is capable of driving an unlimited number of asynchronous clock channels.

- manufacturing_patterns_formats : *format* ;

  A property that specifies a format for the saved test patterns. Valid formats are WGL, STIL, STIL2005, TITDL, FJTDL, MITDL, TSTL2, and SVF. The default format is STIL. For a description of these formats, see the description of the *format_switch* argument to the write_patterns command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- compress_pattern_files : on | off ;

  A property that enables pattern compression.

- MemoryBist/max_controllers_per_test_step : *integer* | unlimited ;

  A property that partitions the memory BIST controllers into many test steps. When an integer is specified, the memory BIST controller inserts no more than the specified number memory BIST controllers per test step during create_patterns_specification. This property only affects the memory BIST pre-repair (if present) and Go/NoGo patterns.

  When generating the ParallelRetentionTest patterns, all memory BIST controllers are inserted inside the same test step, but the number of controllers with the same parallel_retention_group does not exceed the specified value.

  If BIRA is present, the CheckRepairNeeded patterns are not affected by this property.

- MemoryBist/include_setup_chain_test : on | off ;

  A property that specifies the create_patterns_specification command to include test patterns that exercise the setup chains inside the memory BIST controllers and interfaces.

- MemoryBisr/include_repair_chain_test : on | off ;

  A property that specifies the inclusion of test patterns for the testing of BISR chains and connections to and from the BIRA registers. The default value of off creates patterns to test connections that are necessary to repair memories. When set to on, patterns are generated to test all BISR/BIRA connections during manufacturing test, which is desirable in high-reliability systems to maximize the probability of being able to perform an incremental repair in the field.

Related Topics

DefaultsSpecification/PatternsSpecification

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Metadata Configuration Syntax

The metadata syntax is used to describe configuration data trees in the “default” and “tcd” partitions.

This section describes the configuration metadata syntax used inside Tessent Shell to define all valid configuration syntax. The metadata syntax is also used to create the configuration tree widgets in the Config Data Browser that are used to edit and view the configuration data in the Config Data window. For information about loading configuration data in the Config Data Browser, see the add_config_tab and display_specification commands. Finally, the metadata is used by the report_config_syntax command to report the available syntax in a human-readable format that is consistent with how the syntax is documented in this reference guide.

You typically do not need to define metadata because all configuration data is fully defined in the tool. You can, however, define your own metadata to define any configuration syntax needed to automate portions of your flows. Both the Config Data Browser and the report_config_syntax command, which are driven by the metadata, completely support your syntax.

You can also augment the metadata of the DftSpecification for your custom needs. The added elements are ignored by the process_dft_specification command. You can then add your own code to access the added properties and use it for your custom needs. For example, you may want to package a fully-specified DftSpecification file for an RTL core such that the user of the core can simply add the DFT hardware by running process_dft_specification with your packaged DftSpecification file. In this file, you may want to define extra information such as clock sources with their periods. After you have done this, you can use configuration data commands, such as the get_config_value command, to access the information and issue the add_clocks commands in your flow. You can also add extra information that is accessed by your process_dft_specification.post_insertion Procedure and have it automatically perform custom actions based on the captured information in the DftSpecification file.

If you create custom metadata and want it to be loaded into the tool, you must store the metadata file in the <plugin>/metadata directory, where <plugin> is a directory pointed to by the TESSENT_PLUGIN_PATH environment variable, and you must name the metadata file with the extension “.tessent_meta”. If you name the <plugin> directory “tessent_plugin” and place it one level above the bin directory that contains the Tessent Shell executable “tessent”, the tool automatically loads the metadata at invocation unless the TESSENT_PLUGIN_IGNORE_DEFAULT_PATH environment variable is defined.

Wrapper 4759
Property 4768
RepeatableProperty 4772
DataWrapper 4774
CsvWrapper 4777
Id 4779

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Value 4782
RowIdElement 4787

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Wrapper

A wrapper used in metadata to define the syntax of a wrapper that is usable in configuration data.

Usage

```
Partitions : partition_list ; // Only on first line of file
Wrapper(wrapper_name) {
  Repeatable        : On | Off | Single ;
  IgnoreUnknowns    : On | Off ;
  Usage             : normal | hidden | Env(EnvVar) ;
  ReferenceableOnly : On | Off ;
  InsertFirst;
  InsertLast; // default behavior
  InsertBefore      : meta_leaf_name ;
  InsertAfter       : meta_leaf_name ;
  MergeableContent  : On | Off ;
  Options(Visualizer) {
    cdv_tab_prefix  : string ;
    DisplayString   : string ;
  }
  //Everything below is repeatable and optional
  Id(name_id) {
  }
  Property(property_name) {
  }
  RepeatableProperty(property_name) {
  }
  DataWrapper(wrapper_name) {
  }
  CsvWrapper(wrapper_name) {
  }
  Reference = relative_or_absolute_meta_path ;
  ReferenceWithInheritedDefault = relative_or_absolute_meta_path ;
  ReferenceContents = relative_or_absolute_meta_path ;
}
```

Description

A wrapper used in metadata to define the syntax of a wrapper that is usable in configuration data.

The presence of the Partitions property is used to define the partition in which the wrapper, and all of the elements defined within in it, are to exist.

Arguments

- Partitions : *partition_list* ;

  An optional property that can only be specified on the first line of a file, and that controls in which partition the metadata is to be used when defining syntax. When unspecified, the wrappers and what is defined within them, are defined in the default partition.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The partition name can be specified as “default”, “tcd”, or “default, tcd”. The “default” partition is the one in which the DftSpecification and DefaultsSpecification/DftSpecification wrappers exist. The tcd partition is the one in which the files loaded with the read_core_descriptions command are loaded into. The tcd partition is read-only and configuration data found in it cannot be modified using configuration-editing commands such as the add_config_element command.

All existing syntax defined in the “tcd” partition within the tool is also defined in the default partition. If you absolutely need to edit a tcd file, you need to load it into the default partition using the read_config_data command, perform your edits using the editing commands, save it back to the file using write_config_data, and then load it back into the tcd partition using the read_core_descriptions command. Even if it exists in the default partition, the content of this configuration data is not used by the tool because the tool only looks for tcd data in the tcd partition.

- ***wrapper_name***

  A required string that defines the identifier name (Id) of the wrapper to create. The wrapper name must be unique across all wrapper_names for root-level wrappers inside a partition, and across all wrapper_names and property_names found within a common parent wrapper. The wrapper name should comply with the following requirements:

  - Must start with a letter or number followed by optional letters, numbers, and underscores.
  - Must not contain repeated underscores.
  - Must not end with an underscore.

  When you want to define extra syntax inside an already defined wrapper, create a wrapper that contains your new elements and has a “/” as the first character.

  For example, if you want to add a new wrapper called “MyCustomData” inside DftSpecification/IjtagNetwork, use the following syntax:

```
Wrapper(/DftSpecification/IjtagNetwork) {
  Wrapper(MyCustomData) {
    ...
  }
}
```

- Repeatable : On | Off | Single ;

  A property used to define whether the wrapper is repeatable. Both the Off and Single options mean that the wrapper is not repeatable. For named wrappers, the default value is On. For unnamed wrappers, the Repeatable default value is Off.

  The difference between Off and Single is that a wrapper identified as “Repeatable : Single” always exists whether or not it is specified in the configuration data. You can get the value of properties inside a Single wrapper without having to first check if the wrapper exists or not. A “Repeatable : Off” wrapper does not exist until it is specified. A “Repeatable: single” wrapper is not allowed to have an Id wrapper inside it.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- IgnoreUnknowns : On | Off ;

  A property that specifies that wrappers and properties not declared in the metadata of this wrapper are to be tolerated in this wrapper. You should never set this On in wrappers that are not purely machine generated and consumed by a machine as any typos go unnoticed and are silently ignored. These unknown elements are not viewable and editable in the Config Data Browser and report_config_syntax does not report their syntax. It is much safer to define all allowed syntax within a wrapper even if you choose to keep some hidden using the "Usage : hidden;" property, which is documented immediately following.

- Usage : normal | hidden | Env(EnvVar);

  A property that controls the visibility of a wrapper. When set to normal, the wrapper is reported by report_config_syntax and visible in the Config Data Browser. When set to hidden, the wrapper is not reported by report_config_syntax and is invisible in the Config Data Browser. Its value can still be fetched and edited using the configuration introspection and editing commands such as get_config_value if the name of the hidden element is known to you. The "Usage : Env(EnvVar)" is treated as normal if the EnvVar environment variable is defined and set to 1; otherwise, it is treated as hidden.

- ReferenceableOnly : On | Off ;

  A property that controls the existence of the wrapper. When set to On, this wrapper definition does not exist when it is defined. It and its children only exist when referenced using the Reference property . This option is useful to create a wrapper that defines a library of elements to be referenced from other wrappers.

- *InsertFirst* ;

  An implicit Boolean property that specifies that the wrapper is to be inserted before the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

- *InsertLast* ;

  An implicit Boolean property that specifies that the wrapper is to be inserted after the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

- InsertBefore : *meta_leaf_name* ;

  A property that specifies that the wrapper is to be inserted before an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a wrapper before Wrapper(IjtagNetwork), you would specify "InsertBefore : IjtagNetwork;". This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- InsertAfter : *meta_leaf_name* ;

  A property that specifies that the wrapper is to be inserted after an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a wrapper after Wrapper(IjtagNetwork), you would specify "InsertAfter : IjtagNetwork;". This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

- MergeableContent : On | Off ;

  A property that specifies that the content of the wrapper is merged with existing content. When this property is set to on, the content of the wrapper is merged with the existing content when reading in a new copy of the same parent wrapper. For example, if WprA was defined with MergeableContent set to On and a WprA already exist in memory with SubWrapper1. If you read a second copy of WprA which contains SubWrapper2, the content of the new WprA wrapper is merged with the content of the one that already exists in memory such that both SubWrapper1 and SubWrapper2 exists in WprA.

  For unnamed wrappers, you can only specify MergeableContent On when the Repeatable property is set to Off, which is the default for unnamed wrappers. Additionally, all repeatable unnamed wrappers cannot be in a parent wrapper with MergeableContent set ot On.

- Options(Visualizer)/cdv_tab_prefix : *string* ;

  A property used to control the name of the tab that is used by the Config Data Browser when the add_config_tab command is run on this wrapper. The specified prefix is concatenated with an integer incremented from 1 for all added configuration tabs having a common prefix.

- Options(Visualizer)/DisplayString : *string* ;

  A property used to control how the wrapper is labelled in the Configuration Data window. When unspecified, it default to wrapper_name. This option is useful when you want to create a GUI to display configuration data and you want to name the tree nodes using sentences as oppose to the strict wrapper names.

- Reference = *relative_or_absolute_meta_path* ;

  A property used to define an element inside the wrapper using the same definition that was used inside another wrapper. This allows you not only to keep the metadata DRY, it also enables the element to be moved or copied from one location to the other using the Config Data Browser or the "add_config_element -copy_from" and the move_config_element commands. For an element to be copied or moved from one wrapper to the next, the metadata definition in both places must be created in one place by either having both of them reference a unique definition, or by having one definition reference the other. The path may be absolute or relative such as in these two examples:

```
Reference = ../Interface/scan_in;
Reference = /DftSpecification/IjtagNetwork/HostScanInterface/
Interface/scan_in;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ReferenceWithInheritedDefault = *relative_meta_path* ;

  A property used to define a property inside the wrapper using the same definition that was used inside another wrapper. Not only is the definition inherited, the value of the referenced property becomes the default value of the new property. To use this feature, the relative_meta_path can only come back down in wrappers specified with “Repeatable : Single”; this is because to be able to inherit the default value from it, the referenced property must always exist whenever the referencing property exists. For example, the path *../../Interface/scan_in* is legal because Interface is defined with “Repeatable : Single” inside the IjtagNetwork/HostScanInterface wrapper. The above example could be used to create a property called “scan_in” in a wrapper found inside HostScanInterface that defaults to the value of the “../../Interface/scan_in” property when the “./scan_in” property is not specified.

- ReferenceContents = *relative_or_absolute_meta_path* ;

  A property used to reference the elements inside of the wrapper that it points to and not the wrapper itself. See Example 2 for an example usage of this property.

Examples

Example 1

The following example defines the metadata used by the Sib wrapper inside the DftSpecification/IjtagNetwork wrapper. Following is the constructed syntax reported by report_config_syntax on the same wrapper. Notice the recursive definition of the Sib node inside the Sib node achieved by referencing itself using “Reference = ../Sib;”.

```
> report_config_data -partition meta \
       /dftspecification/ijtagnetwork/hostscaninterface/sib
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Wrapper(Sib) {
  Repeatable : On;
  Id(id) {
    Type : String;
    Repeatable : Off;
    IgnoreCase : On;
  }
  Wrapper(Interface) {
    Repeatable : single;
    Wrapper(Client) {
      Repeatable : single;
      Property(tck) {
        Value(port_name) {
          Type : String;
          DefaultValue : ijtag_tck;
        }
      }
      Property(reset) {
        Value(port_name) {
          Type : String;
          DefaultValue : ijtag_reset;
        }
      }
      Property(select) {
        Value(port_name) {
          Type : String;
          DefaultValue : ijtag_sel;
        }
      }
      Property(capture_en) {
        Value(port_name) {
          Type : String;
          DefaultValue : ijtag_ce;
        }
      }
      Property(shift_en) {
        Value(port_name) {
          Type : String;
          DefaultValue : ijtag_se;
        }
      }
      Property(update_en) {
        Value(port_name) {
          Type : String;
          DefaultValue : ijtag_ue;
        }
      }
      Property(scan_in) {
        Value(port_name) {
          Type : String;
          DefaultValue : ijtag_si;
        }
      }
      Property(scan_out) {
        Value(port_name) {
          Type : String;
          DefaultValue : ijtag_so;
        }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
        }
        Property(reset_polarity) {
          Value(active_polarity) {
            Type : String;
            LegalString : active_high, active_low;
            DefaultValue : active_low;
            IgnoreCase : On;
          }
        }
      }
      Wrapper(Host) {
        Repeatable : single;
        Property(select) {
          Value(port_name) {
            Type : String;
            DefaultValue : ijtag_to_sel;
          }
        }
        Property(scan_in) {
          Value(port_name) {
            Type : String;
            DefaultValue : ijtag_from_so;
          }
        }
      }
    }
    Reference = ../ScanMux/Attributes;
    Reference = ../ScanMux/parent_instance;
    Reference = ../ScanMux/leaf_instance_name;
    Reference = ../ScanMux/Input/keep_active_during_scan_test;
    Property(capture_value) {
      Value(enum) {
        type : string;
        LegalString : 1, 0, self;
        DefaultValue : 0;
      }
    }
    Reference = ../Sib;
    Reference = ../Tdr;
    Reference = ../ScanMux;
    Reference = ../DesignInstance;
  }

  > report_config_syntax \
          /dftspecification/ijtagnetwork/hostscaninterface/sib
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DftSpecification(<module_name>,<id>) {
  IjtagNetwork {
    HostScanInterface(<id>) {
      Sib(<id>) {
        Interface {
          Client {
            tck            : <port_name>       ; // default: ijtag_tck
            reset          : <port_name>       ; // default: ijtag_reset
            select         : <port_name>       ; // default: ijtag_sel
            capture_en     : <port_name>       ; // default: ijtag_ce
            shift_en       : <port_name>       ; // default: ijtag_se
            update_en      : <port_name>       ; // default: ijtag_ue
            scan_in        : <port_name>       ; // default: ijtag_si
            scan_out       : <port_name>       ; // default: ijtag_so

            reset_polarity : <active_polarity> ; // legal  : active_high
(active_low)
          }
          Host {
            select  : <port_name> ; // default: ijtag_to_sel
            scan_in : <port_name> ; // default: ijtag_from_so
          }
        }
        Attributes {
          <attribute> : <string> ; // repeatable
        }
        parent_instance              : <instance_name>      ;
        leaf_instance_name           : <leaf_instance_name> ;
        keep_active_during_scan_test : <boolean> ; // default: auto
        capture_value                : <enum> ; // legal  : 1 (0) self
        Sib(<id>) {
          // Same as /DftSpecification/IjtagNetwork/HostScanInterface/Sib
        }
        Tdr(<id>) {
          // Same as /DftSpecification/IjtagNetwork/HostScanInterface/Tdr
        }
        ScanMux(<id>) {
          // Same as /DftSpecification/IjtagNetwork/HostScanInterface/
ScanMux
        }
        DesignInstance(<instance_name>) {
          // Same as /DftSpecification/IjtagNetwork/HostScanInterface/
DesignInstance
        }
      }
    }
  }
}
```

Example 2

In the following example, the two “Test” wrappers are equivalent. In the first case, the objects P and Q of the RefWrapper are both referenced explicitly. In the second case, the objects P and Q of the RefWrapper are referenced once using the new “ReferenceContents” property.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Wrapper(Test) {
  Reference = /RefWrapper/P;
  Reference = /RefWrapper/W;
}

Wrapper(Test) {
  ReferenceContents = /RefWrapper;
}

Wrapper(RefWrapper) {
  Property(P) {…}
  Wrapper(W) {…}
}
```

Related Topics

Property

RepeatableProperty

DataWrapper

CsvWrapper

report_config_syntax

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Property

A wrapper used in metadata to define a property that is usable in configuration data.

Usage

```
Wrapper(wrapper_name) {
  Property(property_name) {
    Usage        : normal | hidden | Env(EnvVar) ;
    InsertFirst ;
    InsertLast ;       // default
    InsertBefore : meta_leaf_name ;
    InsertAfter  : meta_leaf_name ;
    SpecialUsage(id)  : normal | hidden | Env(EnvVar) ;
    Value(value_id) { // repeatable
    }
  }
}
```

Description

A wrapper used in metadata to define a property that is usable in configuration data.

A property exists as soon as its parent wrapper exists regardless of whether the property is specified or not.

Arguments

- Usage : normal | hidden | Env(*EnvVar*) ;

  A property that controls the visibility of the property. When set to normal, the property is reported by the report_config_syntax command and is visible in the Config Data Browser. When set to hidden, the wrapper is not reported by report_config_syntax and is invisible in the Config Data Browser. Its value can still be fetched and edited using the configuration introspection and editing commands, such as the get_config_value command, if the name of the hidden property is known by you. The “Usage : Env(EnvVar)” is treated as normal if the EnvVar environment variable is defined and set to 1; otherwise, it is treated as hidden.

- InsertFirst ;

  An implicit Boolean property that specifies that the property is to be inserted before the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

- InsertLast ;

  An implicit Boolean property that specifies that the property is to be inserted after the already existing elements in the wrapper. This is useful if you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- InsertBefore : *meta_leaf_name* ;

  A property that specifies that the property is to be inserted before an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a property before Property(scan_in), you would specify “InsertBefore : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

- InsertAfter : *meta_leaf_name* ;

  A property that specifies that the property is to be inserted after an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a property after Property(scan_in), you would specify “InsertAfter : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

- SpecialUsage(*id*) : normal | hidden | Env(*EnvVar*);

  A property that controls the visibility of certain properties. When SpecialUsage is evaluated to visible, then the report_config_syntax command reports the LegalStringSpecial and AllowedSymbolsSpecial values. Otherwise the LegalStringSpecial and AllowedSymbolsSpecial values are not reported by the report_config_syntax command. The default value of SpecialUsage is normal. The specification of (*id*) from LegalStringSpecial, AllowedSymbolsSpecial, and SpecialUsage is optional. The (*id*) is used to group LegalStringSpecial and AllowedSymbolsSpecial values into groups, so that each group of values can be specified with a dedicated SpecialUsage.

Examples

The following example defines properties of all types. It is used in the example of the get_config_value command to illustrate how to introspect them. You can use report_config_data DftSpecification -partition meta to see many more examples.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Wrapper(SingularWrapper) {
  Repeatable : single;
  Wrapper(WrapperWithRepeatableId) {
    id(name*) {
      type : string;
    }
    Reference = /SingularWrapper/WrapperWithRepeatableId;
    Property(time_property) {
      Value(time) {
        type : time;
        DefaultValue : 3.5ns;
      }
    }
    Property(integer_property) {
      Value(integer) {
        type : int;
        DefaultValue : 3;
      }
    }
    Property(binary_property) {
      Value(binary) {
        type : binary;
        DefaultValue : 3'b0;
      }
    }
    Property(string_property) {
      Value(string*) {
        type : string;
        DefaultValue : "";
      }
    }
    CSVWrapper(CsvWrapper) {
      Repeatable : on;
      id(module_name) {
        Type : String;
      }
      id(design_id) {
        Type : String;
        IgnoreCase : On;
      }
    }
    DataWrapper(DataWrapper_with_no_value) {
      RowIdElement(port) {
        Type : String;
        repeatable : on;
      }
    }
    DataWrapper(DataWrapper_value) {
      RowIdElement(port) {
        Type : String;
        repeatable : on;
      }
      Value(period) {
        Type : time;
      }
    }
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

DftSpecification

RepeatableProperty

DataWrapper

CsvWrapper

report_config_syntax

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

RepeatableProperty

A wrapper used in metadata to define a repeatable property that is usable in configuration data.

Usage

```
Wrapper(wrapper_name) {
  RepeatableProperty(property_name) {
    Usage        : normal | hidden | Env(envVar) ;
    InsertFirst ;
    InsertLast ;       // default
    InsertBefore : meta_leaf_name ;
    InsertAfter  : meta_leaf_name ;
    Id(id) {           // repeatable and optional
    }
    Value(value_id) { // repeatable, minimum one
    }
  }
}
```

Description

A wrapper used in metadata to define a repeatable property that is usable in configuration data.

As opposed to normal properties, a repeatable property has no default values and does not exist until it is specified.

Arguments

- Usage : normal | hidden | Env(EnvVar) ;

  A property that controls the visibility of a property. When set to normal, the property is reported by report_config_syntax and visible in the Config Data Browser. When set to hidden, the wrapper is not reported by report_config_syntax and invisible in the Config Data Browser. Its value can still be fetched and edited using the configuration introspection and editing commands such as the get_config_value command if the name of the hidden property is known to you. The “Usage : Env(EnvVar)” is treated as normal if the EnvVar environment variable is defined and set to 1; otherwise it is treated as hidden.

- InsertFirst ;

  An implicit Boolean property that specifies that the property is to be inserted before the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

- InsertLast ;

  An implicit Boolean property that specifies that the property is to be inserted after the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- InsertBefore : *meta_leaf_name* ;

  A property that specifies that the property is to be inserted before an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a property before Property(scan_in), you would specify “InsertBefore : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

- InsertAfter : *meta_leaf_name* ;

  A property that specifies that the property is to be inserted after an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a property after Property(scan_in), you would specify “InsertAfter : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

Examples

The following example shows how the connection property is defined inside the Tdr wrapper.

```
> report_config_data [get_config_value Tdr/DataOutPorts/Connection \
    -in_wrapper /DftSpecification/IjtagNetwork/HostScanInterface \
    -partition meta -object]
```

```
RepeatableProperty(connection) {
  Id(range) {
    Type : string ;
    Repeatable : On ;
  }
  Value(pin_name) {
    Type : string ;
    Repeatable : On ;
  }
}
```

Related Topics

DftSpecification

Property

DataWrapper

CsvWrapper

report_config_syntax

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DataWrapper

A wrapper used in metadata to define a data wrapper that is usable in configuration data.

Usage

```
Wrapper(wrapper_name) {
  DataWrapper(wrapper_name) {
    Usage              : normal | hidden | Env(envVar) ;
    InsertFirst ;
    InsertLast ; //default
    InsertBefore       : meta_leaf_name ;
    InsertAfter        : meta_leaf_name ;
    Options(Visualizer) {
      DisplayString    : string ;
    }
    RowIdElement(element_id) { // not_repeatable, required
    }
    Value(value_id) {          // repeatable, optional
    }
  }
}
```

Description

A wrapper used in metadata to define a data wrapper that is usable in configuration data.

As opposed to a normal wrapper, a data wrapper contains only data rows composed of arbitrary row id element with optional values. This type of wrapper is useful to associate values to arbitrary elements such as ports:

```
ClockPeriods {
  clka : 4ns;
  clkb : 3ns;
}
```

A data wrapper is like a wrapper with “Repeatable : Single”. It is not repeatable and starts to exist as soon as its parent wrapper exists.

Arguments

- Usage : normal | hidden | Env(EnvVar) ;

  A property that controls the visibility of the data wrapper. When set to normal, the data wrapper is reported by report_config_syntax and visible in the Configuration Data Visualizer. When set to hidden, the data wrapper is not reported by report_config_syntax and invisible in the Configuration Data Visualizer. Its content can still be fetched and edited using the configuration introspection and editing commands such as the get_config_value command if the name of the hidden data wrapper is known to you. The “Usage : Env(EnvVar)” is treated as normal if the EnvVar environment variable is defined and set to 1; otherwise it is treated as hidden.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- InsertFirst ;

  An implicit Boolean property that specifies that the data wrapper is to be inserted before the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command; it also controls the order in which folding regions associated with the data wrappers are displayed in the Configuration Options panel inside the Configuration Data Visualizer.

- InsertLast ;

  An implicit Boolean property that specifies that the data wrapper is to be inserted after all already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command; it also controls the order in which folding regions associated with the data wrappers are displayed in the Configuration Options panel inside the Configuration Data Visualizer.

- InsertBefore : *meta_leaf_name* ;

  A property that specifies that the data wrapper is to be inserted before an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a data wrapper before Property(scan_in), you would specify “InsertBefore : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command. Also, it controls the order in which folding regions associated with the data wrappers are displayed in the Configuration Options panel inside the Configuration Data Visualizer.

- InsertAfter : *meta_leaf_name* ;

  A property that specifies that the data wrapper is to be inserted after an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a data wrapper after Property(scan_in), you would specify “InsertAfter : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command. Also, it controls the order in which folding regions associated with the data wrappers are displayed in the Configuration Options panel inside the Configuration Data Visualizer.

- Options(Visualizer)/DisplayString : *string* ;

  A property used to control how the folding region associated with a data wrapper is identified in the Configuration Data window. When unspecified, it defaults to wrapper_name. This option is useful when you want to create a GUI to display configuration data and you want to name the folding regions as opposed to the strict data wrapper name.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example shows how the ClockPeriods data wrapper shown in the description section above is defined in metadata. Notice the value of type time with the extra allowed symbols sync and sync2x.

```
DataWrapper(ClockPeriods) {
  RowIdElement(clock_port) {
    Repeatable : Off ;
    Type       : String ;
    IgnoreCase : Off ;
  }
  Value(period) {
    Type           : Time ;
    AllowedSymbols : sync, sync2x ;
  }
}
```

Related Topics

DftSpecification

Property

RepeatableProperty

CsvWrapper

report_config_syntax

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

CsvWrapper

A wrapper used in metadata to define a CSV wrapper that is usable in configuration data.

Usage

```
Wrapper(wrapper_name) {
  CsvWrapper(wrapper_name) {
    Usage        : normal | hidden | Env(envVar) ;
    InsertFirst ;
    InsertLast ;    // default
    InsertBefore : meta_leaf_name ;
    InsertAfter  : meta_leaf_name ;
    Id(id) {       // repeatable and optional
    }
  }
}
```

Description

A wrapper used in metadata to define a CSV wrapper that is usable in configuration data.

As opposed to a normal wrapper, a CSV wrapper contains only rows composed of arbitrary comma separated values.

```
csv {
  v1,v2,"v3 with spaces or comma";
  v4,v5;
}
```

See the command description of report_config_data and write_config_data to see how the wrapper content can be imported from or exported to Excel. CSV wrappers are neither visible nor editable inside Tessent Visualizer. They are meant to be manipulated graphically inside Excel.

Arguments

- Usage : normal | hidden | Env(EnvVar) ;

  A property that controls the visibility of the CSV wrapper. When set to normal, the wrapper is reported by the report_config_syntax command. When set to hidden, the wrapper is not reported by report_config_syntax. Its value can still be fetched and edited using the configuration introspection and editing commands such as get_config_value if the name of the hidden CSV wrapper is known to you. The "Usage : Env(EnvVar)" is treated as normal if the EnvVar environment variable is defined and set to 1; otherwise it is treated as hidden.

- InsertFirst ;

  An implicit Boolean property that specifies that the CSV wrapper is to be inserted before the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- InsertLast ;

  An implicit Boolean property that specifies that the CSV wrapper is to be inserted after the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

- InsertBefore : *meta_leaf_name* ;

  A property that specifies that the CSV wrapper is to be inserted before the already existing elements in the wrapper. You refer to the element using its leaf_name. For example, to insert a CSV wrapper before Wrapper(IjtagNetwork), you would specify "InsertBefore : IjtagNetwork;". This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

- InsertAfter : *meta_leaf_name* ;

  A property that specifies that the CSV wrapper is to be inserted after an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a CSV wrapper after Wrapper(IjtagNetwork), you would specify "InsertAfter : IjtagNetwork;". This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

Examples

The following example shows how a generic scratch CSV wrapper is defined in the tool. You can use scratch CSV wrappers to import and introspect CSV data.

```
CsvWrapper(csv) {
  Repeatable : On ;
  Usage : hidden ;
  Id(name) {
    Type : string ;
  }
}
```

Related Topics

DftSpecification

Property

RepeatableProperty

DataWrapper

report_config_syntax

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Id

A wrapper used in metadata to define an identifier for named wrappers, repeatable properties, and named CSV wrappers.

Syntax

```
Wrapper(wrapper_name) {
  Id (name_id) {
    Type           : String | Boolean | Real | Int | Time | Binary ;
    LegalString    : string_list ;        // For String
    LegalInt       : integer_range_list ; // For Int
    LegalReal      : real_range_list ;    // For Real
    LegalSize      : int | Unsized ;      // For Binary
    AllowedSymbols : string_list ;        // For all but String
    IgnoreCase     : On | Off ;
    Repeatable     : On | Off ;
    Mandatory      : On | Off ;
  }
  RepeatableProperty(property_name) {
    Id (name_id) {
    }
  }
  CsvWrapper(wrapper_name) {
    Id (name_id) {
    }
  }
}
```

Description

A wrapper used in metadata to define an identifier for named wrappers, repeatable properties, and named CSV wrappers.

Parameters

- *name_id*

  A string used to name the identifier (Id). You can later reference this name can using the "get_config_value -id name_id" command. The Id must begin with a letter, and include only letters, numbers, and underscores.

  You may terminate *name_id* with an asterisk (*). If the *name_id* ends with an *, the Id is repeatable. If there is more than one Id wrapper, only the last one can have an * at the end.

  For example:

  ```
  Wrapper(Step) {
    Id(instance*) {
    }
  }
  ```

  allows Step(u1,u2), where u1 and u2 are repeated instance Ids.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Type : *value* ;

  A property that defines the type of the Id. The allowed types are String, Boolean, Real, Int, Time, and Binary. Ids of type time are allowed to have a unit string: s, ms, us, ns, ps, or fs. Binary numbers can be expressed in binary or hexadecimal format using the Verilog syntax: <size>‘b<binary_digits>, ‘b<binary_digits>, <size>‘h<hexadecimal_digits>, and ‘h<hexadecimal_digits>. The default selection is of type String.

- LegalString : *string_list* ;

  A property that can be used when Type is set to String to limit the string to an allowed list. The validation of the allowed symbols is case insensitive by default or based on the Id’s IgnoreCase property.

- LegalInt : *integer_range_list* ;

  A property that can be used when Type is set to Int to limit the integer to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in “1..9, 20, 21”.

  Instead of using only numerical values, integer_range_list can also use the keywords maxposint and maxnegint. The numerical equivalent of maxposint is max value of a 64-bit integer, which is 9223372036854775807. The numerical equivalent of maxnegint is the largest negative number of a 64-bit integer, which is -9223372036854775808.

- LegalReal : *real_range_list* ;

  A property that can be used when Type is set to Real to limit the real to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in “1.0..9.5, 20.0, 21.0”.

- LegalSize : *int* | <u>Unsized</u> ;

  A property that can be used when Type is set to Binary to limit the binary number to a specific size.

- AllowedSymbols : *string_list* ;

  A property that can be used when Type is set to anything but String to allow extra symbols along with the allowed Boolean, Real, Int, Time, and Binary values. The LegalSize property is an example of a property of type Int with an extra allowed symbol of “Unsized”. The validation of the allowed symbols is case insensitive by default or based on the Id’s IgnoreCase property.

- IgnoreCase : On | <u>Off</u> ;

  A property that can be used to make the LegalString and the AllowedSymbols case sensitive. By default, the LegalString and the AllowedSymbols are case insensitive.

- Repeatable : On | <u>Off</u> ;

  A property that controls if a given name_id can be repeated or not. The validation of the repeatability considers casing or not based on the IgnoreCase property documented above.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Mandatory : On | Off ;

  A property that specifies if the Id is required or not. This option can only be specified Off if you have more than one Id wrapper, the Id wrapper is not the first one and all Id wrappers below it also have this property set to Off.

Examples

The following example defines a wrapper with two Ids, the second one is repeatable and optional.

```
Wrapper(EX1) {
  Id(name) {
    type : string;
  }
  Id(period*) {
    type : time;
    Mandatory : off;
  }
}
```

Related Topics

DftSpecification

RepeatableProperty

CsvWrapper

Value

report_config_data

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Value

A wrapper used in metadata to define an identifier for values of properties, repeatable properties, and data wrappers.

Usage

```
Wrapper(wrapper_name) {
  Property(property_name) {
    Value (value_id) {
      Type            : String | Boolean | Real | Int | Time | Binary ;
      LegalString     : string_list ;         // For String
      LegalStringSpecial(id) : string_list; // For String
      LegalInt        : integer_range_list ; // For Int
      LegalReal       : real_range_list ;     // For Real
      LegalSize       : int | Unsized ;       // For Binary
      DefaultValue    : default_value ;
      AllowedSymbols : string_list ;          // For all but String
      AllowedSymbolsSpecial(id) : string_list;
      IgnoreCase      : On | Off ;
      Mandatory       : On | Off ;
      Repeatable      : On | Off ;
      SizedBinary     : On | Off ;
    }
  }
  RepeatableProperty(property_name) {
    Value (value_id) {
    }
  }
  DataWrapper(wrapper_name) {
    Value (value_id) {
    }
  }
}
```

Description

A wrapper used in metadata to define an identifier for values of properties, repeatable properties, and data wrappers.

Arguments

- *value_id*

  A string used to name the value. You can reference this name later by using the "get_config_value -value_id value_id" command.

  The *value_id* must start with a letter, include only alphanumerics and underscores, and may be terminated by an asterisk ("*"). If the *value_id* ends with an *, you can enter a comma-separated list of values for that value. If there is more than one Value wrapper, only the last one can have an * at the end. For such repeatable values, you can repeat the same value more than once only when the Repeatable property is set to on.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 1:

```
RepeatableProperty(instance) {
  Value(instance_path) {}
}
```

- instance is specified as: instance : chip/module/instance;
- Result: value_id “instance_path” contains “chip/module/instance”.

Example 2:

```
RepeatableProperty(instance) {
  Value(power_domain) {}
  Value(instance_path) {}
}
```

- instance is specified as: instance : domain1, chip/module/instance;
- Result: value_id “power_domain” contains “domain1”value_id “instance_path” contains “chip/module/instance”

Example 3:

```
RepeatableProperty(instance) {
  Value(power_domain) {}
  Value(instance_paths*) {}
}
```

- Where instance is specified as: instance : domain1, instance1, instance2, “chip/module/\escaped_inst ”;
- Result: value_id “power_domain” contains “domain1” value_id “instance_paths” is returned as a Tcl list containing ‘instance1, instance2, {chip/module/\escaped_inst }’ by get_config_value.

- Type : *value* ;

  A property that defines the type of the Value. The allowed types are String, Boolean, Real, Int, Time, and Binary. The default selection is of type String.

  Values of type time must be a non-negative real or integer number. You can optionally include a unit string: s, ms, us, ns, ps, fs, or as.

  Binary numbers can be expressed in binary or hexadecimal format using the Verilog syntax: <size>‘b<binary_digits>, ‘b<binary_digits>, <size>‘h<hexadecimal_digits>, and ‘h<hexadecimal_digits>.

- LegalString : *string_list* ;

  A property that can be used when Type is set to String to limit the string to an allowed list. The validation of the allowed list considers casing or not based on the Value’s IgnoreCase property.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- LegalStringSpecial(*id*) : *string_list*;

  A property that can be used when Type is set to String to limit the string to an allowed list. The allowed specification of LegalStringSpecial is in addition to the LegalString. The usage of type String allows legal values to be values from both LegalString and LegalStringSpecial.

- LegalInt : *integer_range_list* ;

  A property that can be used when Type is set to Int to limit the integer to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in "1..9, 20, 21".

  Instead of using only numerical values, integer_range_list can also use the keywords maxposint and maxnegint. The numerical equivalent of maxposint is max value of a 64-bit integer, which is 9223372036854775807. The numerical equivalent of maxnegint is the largest negative number of a 64-bit integer, which is -9223372036854775808.

- LegalReal : *real_range_list* ;

  A property that can be used when Type is set to Real to limit the real to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in "1.0..9.5, 20.0, 21.0".

- LegalSize : int | Unsized ;

  A property that can be used when Type is set to Binary to limit the binary number to a specific size.

- DefaultValue: *default_value* ;

  A property that is mandatory and only used when the Value wrapper is inside a Property wrapper. The specified *default_value* provides the value of the property when the property is not specified in the wrapper where it exists.

  Note

  The specified default value must be legal for the given value type and allowed values. You can use the null value "" regardless of the value type and allowed values.

- AllowedSymbols : *string_list* ;

  A property that can be used when Type is set to anything but String to allow extra symbols along with the allowed Boolean, Real, Int, Time, and Binary values. The LegalSize property for Value is an example of a property of type Int with an extra allowed symbol of "Unsized". The validation of the allowed symbols considers casing or not based on the Value's IgnoreCase property.

- AllowedSymbolsSpecial(*id*) : *string_list*;

  A property that can be used when Type is set to Int. The allowed specification of AllowedSymbolsSpecial is in addition to the AllowedSymbols. The usage of type Int allows legal values to be values from both AllowedSymbols and AllowedSymbolsSpecial.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- IgnoreCase : On | Off ;

  A property that can be used to make the LegalString and the AllowedSymbols case-insensitive.

- Mandatory : On | Off ;

  A property that specifies if the Value is required or not. For Property and RepeatableProperty, the Mandatory of the first Value must be on. For DataWrapper, the Mandatory of the first Value can be on or off. If you set the Mandatory property of a Value to off, the Mandatory of all subsequent Values must also be set to off.

- Repeatable : On | Off ;

  An optional property that specifies if a *value_id* that has an ID ending with an * can have the same value repeated more than once.

  For example, a Value wrapper defined as follows would prevent the same pin name from being specified more than once:

```
Property (pin_name) {
 Value (name*) {
   Type : string;
 }
}
```

  Setting this property to on enables you to repeat the same pin name.

```
Property (pin_name) {
 Value (name*) {
   Type : string; Repeatable : on;
 }
}
```

- SizedBinary : On | Off ;

  A property that specifies that the binary value from the configuration data must be a sized binary. It must be a sized binary, hexadecimal, or decimal number using the *int*'b, *int*'h, or *int*'d prefix.

Examples

The following example defines a property with two values; the second one is repeatable and optional.

```
Property(EX1) {
  Value(name) {
    type : string ;
  }
  Value(period*) {
    type : time ;
    Mandatory : off ;
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

DftSpecification

RepeatableProperty

CsvWrapper

report_config_syntax

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

RowIdElement

A wrapper used in metadata to define the syntax of the RowIdElements of a data wrapper.

Usage

```
DataWrapper(wrapper_name) {
  RowIdElement (element_id) {
    Type           : String | Boolean | Real | Int | Time | Binary ;
    LegalString    : string_list ;        // For String
    LegalInt       : integer_range_list ; // For Int
    LegalReal      : real_range_list ;    // For Real
    LegalSize      : int | Unsized ;      // For Binary
    AllowedSymbols : string_list ;        // For all but String
    IgnoreCase     : On | Off ;
    Repeatable     : On | Off ;
  }
}
```

Description

A wrapper used in metadata to define the syntax of the RowIdElements of a data wrapper.

Arguments

- *element_id*

  A string used to name the row id element.

  The element_id must start with a letter, include only letters, numbers and underscores.

- Type : String | Boolean | Real | Int | Time | Binary ;

  A property that defines the type of the RowIdElement. The allowed types are String, Boolean, Real, Int, Time, and Binary. RowIdElements of type time are allowed to have a unit string: s, ms, us, ns, ps, or fs. Binary numbers can be expressed in binary or hexadecimal format using the Verilog syntax: <size>‘b<binary_digits>, ‘b<binary_digits>, <size>‘h<hexadecimal_digits>, and ‘h<hexadecimal_digits>.

- LegalString : *string_list* ;

  A property that can be used when Type is set to String to limit the string to an allowed list. The validation of the allowed list considers casing or not based on the IgnoreCase property documented below.

- LegalInt : *integer_range_list* ;

  A property that can be used when Type is set to Int to limit the integer to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in “1..9, 20, 21”.

  Instead of using only numerical values, integer_range_list can also use the keywords maxposint and maxnegint. The numerical equivalent of maxposint is max value of a 64-bit integer, which is 9223372036854775807. The numerical equivalent of maxnegint is the largest negative number of a 64-bit integer, which is -9223372036854775808.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- LegalReal : *real_range_list* ;

  A property that can be used when Type is set to Real to limit the real to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in “1.0..9.5, 20.0, 21.0”.

- LegalSize : *int* | Unsized ;

  A property that can be used when Type is set to Binary to limit the binary number to a specific size.

- AllowedSymbols : *string_list* ;

  A property that can be used when Type is set to anything but String to allow extra symbols along with the allowed Boolean, Real, Int, Time, and Binary values. The LegalSize property above is an example of a property of type Int with an extra allowed symbol of “Unsized”. The validation of the allowed symbols considers casing or not based on the IgnoreCase property documented below.

- IgnoreCase : On | Off ;

  A property that can be used to make the LegalString and the AllowedSymbols case-insensitive. This option also affects whether casing is ignored when performing the repeatability check described in the following Repeatable property.

- Repeatable : On | Off ;

  A property that controls whether a given element_id can be repeated or not within the Data Wrapper. The validation of the repeatability considers casing or not based on the preceding IgnoreCase property.

Examples

The following example defines a data wrapper with RowIdElement which is a arbitrary string meant to represent a clock port name. Each clock port name are only allowed once as specified by the fact that Repeatable is set to Off. Because IgnoreCase is also Off, port names Clk and clk is not considered as repeated port names.

```
DataWrapper(ClockPeriods) {
  RowIdElement(clock_port) {
    Repeatable : Off ;
    Type       : String ;
    IgnoreCase : Off ;
  }
  Value(period) {
    Type           : Time ;
    AllowedSymbols : sync. sync2x ;
  }
}
```

Related Topics

DftSpecification

RepeatableProperty

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

CsvWrapper

Value

report_config_syntax

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DftSpecification and Complex Ports

Because complex signals cannot be edited in Tessent Shell, DftSpecification properties must be specified as scalars or one-dimensional vectors to facilitate connection editing. To use complex port types in DftSpecifications, define scalar and one-dimensional vector pins to connect to those complex ports.

Consider the following DftSpecification:

```
DftSpecification(design1,rtl1) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Interface {
        scan_in : buf_si/Y ;
        scan_out : buf_so/A ;
      }
    }
  }
}
```

The SystemVerilog design corresponding to this DftSpecification is as follows:

```
typedef struct packed {
  logic a;
  logic b;
} struct_t;

module core1(ijtag_in, ijtag_out);
  input struct_t ijtag_in;
  output struct_t ijtag_out;

  buffer_module buf_si (.A(ijtag_in.a), .Y());
  buffer_module buf_s0 (.A(), .Y(ijtag_out.a));
  ...
endmodule
```

Embedded Boundary Scan Flow With Complex Pad IO Ports. 4790

Embedded Boundary Scan Flow With Complex Pad IO Ports

In the chip-level flow, object names at the top level must be legal for the BSDL file format.

Legal port names at the chip level are scalars or one-dimensional vectors. If your design has complex ports, add a wrapper to map them to ports with simple names with the write_design command and the -simulation_wrapper option.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Complex pad ports at the physical_block and sub_block level are supported. For example, consider a structure like the following:

```
typedef struct packed {
        logic a;
        logic [1:0] b;
      } data_t;
```

And a core that instantiates that port:

```
module core1(data_compl);
       input data_t data_compl;
       ...
endmodule
```

Assuming that the data_compl ports are connected to pad cells, specify your BoundaryScan hardware as follows:

```
SETUP> set_dft_specification_requirements -boundary_scan on
SETUP> set_boundary_scan_port_options \
  -pad_io_ports [get_ports {data_compl*}]
```

After running check_design_rules, create the DftSpecification as follows:

```
ANALYSIS> set spec [create_dft_specification]
      //  sub-command: create_dft_specification
      //
      //  Begin creation of DftSpecification(cored,rtl)
      //    Creation of RtlCells wrapper
      //    Creation of IjtagNetwork wrapper
      //    Creation of EmbeddedBoundaryScan wrapper
      //
      //  Done  creation of DftSpecification(cored,rtl)
      //
ANALYSIS> report_config_data $spec
    DftSpecification(cored,rtl) {
      EmbeddedBoundaryScan {
        pad_io_ports : data_compl.a, data_compl.b[1], data_compl.b[0];
      }
    }
```

Tessent Shell inserts the required BoundaryScan hardware while performing DFT insertion.

Note

Map the complex port names to simple names later during the chip-level design flow.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 14
Tessent Core Description

This section provides links to supporting documentation that describes the Tessent Core Description (TCD) configuration data syntax.

Refer to the links below for the primary documentation that describes the following TCD macro module types:

- **BoundaryScan Segments —** Refer to "Tessent Core Description" in the *Tessent BoundaryScan User's Manual*.
- **FuseBoxInterface —** Refer to "FuseBoxInterface" in the *Tessent MemoryBIST User's Manual*.
- **Memories —** Refer to "Memory" in the *Tessent MemoryBIST User's Manual*.
- **Scan —** Refer to "Scan" on page 4794.
- **Shared Bus Memory Clusters —** Refer to "MemoryCluster" in the *Tessent MemoryBIST User's Manual*.

Scan 4794

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Scan

Describes the scan chains for the specific *module_name*.

Usage

```
Core(module_name) {
  Scan {
    module_type                        : normal | occ | memory |
single_chain_mode_logic | edt | logic_bist | in_system_test ;
    allow_scan_out_retiming            : 1 | 0;
    is_hard_module                     : 1 | 0;
    exclude_from_concatenated_netlist  : 1 | 0;
    pre_scan_drc_on_boundary_only      : 1 | 0;
    internal_scan_only                 : 1 | 0;
    TestEn(port_name,...) {
      active_polarity : sized_binary | all_ones | all_zeros;
    }
    Iproc(proc_name) {
      parameter_value_list : parameter_value, ...;
    }


    Mode(mode_name) {
      type :  unwrapped | internal | external;
      traceable : 1 | 0;
      make_active_automatically :  1 | 0;
      enable_dft_signal : dft_signal_name;
      ModeEn(mode_en_port_name,...) {
        active_polarity  : sized_binary | all_ones | all_zeros;
      }
      ScanEn(scan_en_port_or_pin_name,...) {
        active_polarity : sized_binary | all_ones | all_zeros;
        pipeline_count  : 0 | 1;
        clock           : port_name;
      }
      SePipelineEn(SePipelineEn_port_or_pin_name) {
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
}
      Clock(clock_port_or_pin_name,...) {
        off_state       : sized_binary | all_ones | all_zeros;
        used_for_capture : on | off | auto;
      }
      ClockOut(clock_out_port_or_pin_name,...) {
        slow_clock_input : port_name;
        fast_clock_input : port_name;
     }
      SetReset(set_reset_port_or_pin_name,...) {
        active_polarity  : sized_binary | all_ones | all_zeros
        gated_off_during_shift_inside : on | off | auto;
      }
      ScanChain {
        length           : int|auto;
        mux_si_functional_source : 1 | 0;
        no_si_pipelining : 1 | 0;
        scan_in_port     : port_name,...;
        scan_out_port    : port_name,...;
        scan_in_clock    : port_name,...;
        scan_out_clock   : port_name,...;
      }
      ForcedDuringPreScanTracing {
        port_or_pin_name : constrained_value; // repeatable
      }
    }
  }
}
```

Note

The Mode wrapper container is mandatory only for a core with multiple scan modes; it is optional when only one mode is defined. If you do not specify the Mode wrapper, place its relevant content in the Scan wrapper.

Note

The enable_dft_signal property and the ModeEn wrapper are mutually exclusive and only required for a core with multiple scan modes.

Description

Describes the scan chains for the specific *module_name*. The tool automatically reads these descriptions during module matching and connects the scan enable port of the modules to the scan_en signal.

See the "set_design_sources -format tcd_scan" command description for information about where the tool searches for these items. See the read_core_descriptions command description for information about reading the descriptions explicitly. See the set_module_matching_options command description for information about the name matching process.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

To see the content of a read-in Core(*module_name*)/Scan, use the following command syntax:

```
report_config_data Core(module_name)/Scan -partition tcd
```

To see the complete supported syntax, use the following command syntax:

```
report_config_syntax [get_config_value Core/Scan -partition meta:tcd -object]
```

Default Generation of CTL Files

By default, if you are generating IP with scan segments and tcd_scan is available, the tool writes CTL files containing scan-related data to the TSDB. The following Tessent IP includes scan segments:

- EDT block Controller Chain Mode (CCM)
- In-System Test Controller (CCM)
- LogicBIST Controller CCM present, segment_per_instrument == ON
- OCC (various types)
- Sib(STI)
- Single Chain Mode Logic (CCM)

The scan segments for the Tessent IP are also included in the respective tcd_scan files generated using IP creation. You generate the IP using the process_dft_specification command. For more information, see "DFT Insertion Flow With Third-Party Scan Insertion" in the *Tessent Shell User's Manual*.

Arguments

- module_type : normal | occ | memory | single_chain_mode_logic | edt | logic_bist | in_system_test;

  A property that defines the type of the module. Most modules have type normal.

  Identifying a module with type "occ" instructs the tool to infer a clock on the specified ClockOut location when entering the analysis mode in the dft -scan context if no clock exists. This ensures the scan insertion flow does not make clock connections for inserted re-timing elements to a net that is not driven by the OCC. Also, the scan insertion uses this information to not align all the OCC chains, which may result in encoding conflicts when too many OCC care bits align.

- allow_scan_out_retiming : 1 | 0;

  A Boolean property to specify whether the clock associated to a scan chain is suitable for use as the source for retiming elements or not. By default, when a segment needs retiming, the tool adds the retiming element after the scan-out port using the clock port associated to the scan chain. In certain situations, this is not the best approach because the clock input is not balanced with the chain scan-out port. This situation happens in an OCC, for example, as can be seen in the schematic diagram of The Standard OCC in the *Tessent Scan and*

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ATPG User's Manual. When the property is set to 0, the tool adds the retiming element at the scan-in port of the next scan element instead and uses the clock associated to the next scan element.

- type : unwrapped | internal | external;

  A property to define a type of a scan mode. Refer to Unwrapped Cores Versus Wrapped Cores in the *Tessent Scan and ATPG User's Manua*l for more details.

- make_active_automatically : 1 | 0;

  A Boolean property to indicate whether the scan segments should be included in the scan element population by default.

  Setting the make_active_automatically property to 0 ensures that the scan segments of a given node are not included in the scan element population by default when performing scan stitching on the next level of design hierarchy. You must set the active_child_scan_mode property on the design module or instance to make these scan elements usable for scan stitching.

- enable_dft_signal : dft_signal_name ;

  A property to define a port on a core to be driven by a specific DFT signal of Scan Mode usage. The tool uses this port to enable a scan mode within the core. It is mutually exclusive with the "ModeEn" wrapper.

- ModeEn (mode_en_port_name,...)

  A wrapper to define one or more ports as having the function "ModeEn". The tool uses these ports to enable a scan mode within the core. It is mutually exclusive with the "enable_dft_signal" property.

- is_hard_module : 1 | 0;

  A property to specify whether the module is hard or not. When set to 1, the tool assumes flip-flops not traced as part of the scan chains are non-scannable. It does not collect these scan chains by scan insertion and does not issue DRC violations for these non-scannable flops.

  When set to 0, the tool assumes the module is soft and that the flip-flops not traced as part of the scan chains are scannable, unless you explicitly declare them as non-scannable using the add_nonscan_instances command. The extra scannable flops may generate DRC violations, and if DRC is clean, the tool inserts these flops in the newly-created scan chains.

- exclude_from_concatenated_netlist : 1 | 0;

  A property to specify that the module is a hard macro and should not be included in the concatenated netlist. A hard macro is a module that has a predefined layout, but is too large to be a cell. Do not include these modules in the concatenated netlist because the synthesis and layout tool use the Liberty format instead of the netlist view. This property can only be set to 1 when the is_hard_module property is set to 1. Its value gets transferred to the exclude_from_concatenated_netlist module level attributes and affects the write_design command accordingly. See the documentation of the -exclude_modules switch of the write_design command description for more details.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- pre_scan_drc_on_boundary_only : 1 | 0;

  A property to specify that the module is to be treated as a black box. The tool runs DRCs on the scan chains from the external boundary of the module. It does not trace them, and does not report a DRC violation for the internals of the module. This property is useful when the module requires a complex test_setup procedure to configure the module in scan mode.

  For example, you would use this feature to describe the chain going through the Tessent OCC controller. Because the scan chain length is configurable by the max_capture_size static signals, you would need to configure it to the maximum value for the tracing to work and to consider the maximum chain length during scan chain balancing. Instead of doing that, you can describe the maximum length of the chain using the ScanChain/length property. If the scan internals of the scan chain have a design flaw, the tool detects it after scan insertion while performing DRC one scan mode at a time.

- internal_scan_only : 1 | 0;

  A property to specify whether the tool allows the scan chains of the module to be promoted as wrapper chains. See the set_wrapper_analysis_options command description for more details on how scan segments are handled by the analysis. When this property is set to 1, the tool does not allow the scan chains to be promoted to the shared wrapper chains. Also, the tool infers dedicated wrapper cells to isolate them if the chain interacts with the outside, even if "set_wrapper_analysis_options -allow_internal_segments_as_wrapper" is set to on. The tool connects the scan enable port of such modules to the scan_en signal.

- TestEn(*port_name*,...)

  A wrapper to define one or more TestEn ports on the module. See the description of the ltest_en DFT signal in Table 3-12 on page 210 in the add_dft_signals command description.

- TestEn/active_polarity : *sized_binary* | all_ones | all_zeros;

  A property that defines the active polarity of the TestEn ports. The value must be a sized binary number where the width must match the number of ports listed in the TestEn parenthesis. A "1" in the *n*th position means the *n*th TestEn port in the list is active high, and a "0" means it is active low. For example, the port list (A, b[1:0]), requires a 3-bit binary number where the left most bit corresponds to port A and the right most bit corresponds to bit b[0]. You can also use the symbols all_ones and all_zeros when they are all of the same active polarity.

- traceable : 1 | 0;

  A property to specify whether the scan chain is traceable or not. When not traceable, you must specify the length, scan_in_clock, and scan_out_clock properties inside the ScanChain wrapper as the tool cannot extract them.

  Note

  It is not always possible to accurately identify the flops inside the OCC. Therefore, you are not allowed to change the traceable property.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- ScanEn(*scan_en_port_or_pin_name*,...)

  A wrapper to define one or more ports as having the function "ScanEn". The tool asserts these ports active to place the scan chains in their shift configuration. You must have at least one ScanEn port when there are one or more ScanChain wrappers.

- ScanEn/active_polarity : sized_binary | all_ones | all_zeros;

  A property that defines the active polarity of the ScanEn ports. The value must be a sized binary number where the width must match the number of ports listed in the ScanEn parenthesis. A "1" in the *n*th position means the *n*th ScanEn port in the list is active high, and a "0" means it is active low. For example, the port list (A, b[1:0]), requires a 3-bit binary number where the leftmost bit corresponds to port A and the rightmost bit corresponds to bit b[0]. You can also use the symbols all_ones and all_zeros when they are all of the same active polarity.

- ScanEn/pipeline_count : 0 | 1;

  A property to specify whether the scan_en is pipelined or not. The default value of 0 means the scan_en of the core is not pipelined. While implementing scan enable pipelining, the scan insertion tool only connects a scan segment from a TCD to a pipelined scan enable if that scan segment has a ScanEn/pipeline_count of 0 (no pipeline already) and a valid ScanEn/clock. Otherwise, the tool connects the scan segment to the non-pipelined scan enable. See Scan Enable Pipelining in the *Tessent Scan and ATPG User's Manual*.

- ScanEn/clock(*clock_port_or_pin_name*)

  A property to specify the scan enable pipelining clock. If specified, the property indicates that the tool can safely implement external scan enable pipelining using the clock signal driving the specified *clock_port_or_pin_name*. See "Scan Enable Pipelining" in the *Tessent Scan and ATPG User's Manual*.

- SePipelineEn(*SePipelineEn_port_or_pin_name*)

  A wrapper to define a core as having a scan enable pipeline enable port or pin. The scan insertion tool writes out a SePipelineEn wrapper for a scan-inserted core if it is sourced from a top-level port.

- Clock(*clock_port_or_pin_name*,...)

  A wrapper to define one or more ports as having the function "Clock". All scan cells in the scan chain must be clocked by one of those defined Clock ports.

- Clock/off_state : sized_*binary* | all_ones | all_zeros;

  A property that defines the off state of the clock ports when pulsing them to extract the scan chains. The value must be a sized binary number where the width must match the number of ports listed in the ScanEn parenthesis. A "1" in the *n*th position means the *n*th ScanEn port in the list is active high, and a "0" means it is active low. For example, the port list (A, b[1:0]), requires a 3-bit binary number where the leftmost bit corresponds to port A and the rightmost bit corresponds to bit b[0].

  The offstate is only relevant if your chains have a mixture of rising and falling edge flip-flops. Defining the off state incorrectly in that case may cause the scan chain to be

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

extracted with a length that is not completely accurate and may have a minor effect on chain balancing.

- Clock/used_for_capture : on | off | auto;

  A property that defines that the related clock ports are used for capture of the internal scan chain segments. The "on" value adds the related instance pins to the DFT_C6 checks to ensure that the interface on the described core with embedded scan chains connects well with other logic at the levels above. The "auto" value enables the same if the related module has an "is_blackboxed" attribute and has no "mode" wrapper.

- ClockOut(*clock_out_port_or_pin_name,...*)

  This wrapper defines one port or pin inside the module on which to add a clock when going to analysis mode in the dft -scan context. The ClockOut ports are only allowed and used when module_type is set to occ. See the preceding description of the module_type property for more information. The ClockOut port wrapper must include one slow_clock_port and/or fast_clock_port.

- ClockOut/slow_clock_input : *port_name*;

  A string property to provide the name of the input port through which the slow clock enters the module. The tool uses this clock as the shift clock and, optionally, the capture clock when not operating in fast capture mode. If you have an OCC that only gates the clock, and the shift clock is injected earlier in the clock tree, you only have a fast_clock_port. The mini tck OCC present in the Sib(sti) only has a slow_clock_port.

- ClockOut/fast_clock_input : *port_name*;

  A string property to provide the name of the input port through which the fast clock enters the module. The tool uses this clock as the capture clock when operating in fast capture mode. If you have an OCC that only gates the clock, and the shift clock is injected earlier in the clock tree, you only have a fast_clock_port.

- FastCaptureClockEn(*pin_name_on_persistent_cell*)

  Only use this wrapper within the Core/Scan wrapper with a module_type equal to "occ". The wrapper specifies the enable pin of the clock gating cell that gates the fast clock during logic test mode. This pin is forced off when running the DFT_C1, DFT_C2, DFT_C3, and DFT_C6 DRCs in order to check the functional clock source.

- FastCaptureClockEn/clock_input : *port_name*;

  The tool does not yet use this property.

- ShiftRegisterClockEn(*pin_name_on_persistent_cell*)

  Only use this wrapper within the Core/Scan wrapper with module_type equal to "occ". The wrapper specifies the enable pin of the clock gating cell that gates the clock of the shift register inside the OCC. This pin is forced off when running the DFT_C1, DFT_C2, DFT_C3, and DFT_C6 DRCs in order to check the functional clock source.

- ShiftRegisterClockEn/clock_input : *port_name*;

  The property is currently unused.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- SetReset(*set_reset_port_name*,...)

  A wrapper to define one or more ports as having the function "SetReset". The tool uses these ports to control the Set and Reset pins of the scannable flip-flops found inside the module and asserts them inactive when tracing the scan chains.

- SetReset/active_polarity : sized_binary | all_ones | all_zeros;

  A property that defines the active polarity of the SetReset ports. The value must be a sized binary number where the width must match the number of ports listed in the ScanEn parenthesis. A "1" in the *n*th position means that the *n*th SetReset port in the list is active high, and a "0" means that it is active low. For example, the port list (A, b[1:0]), requires a 3-bit binary number where the leftmost bit corresponds to port A and the rightmost bit corresponds to bit b[0]. You can also use the symbols all_ones and all_zeros when they are all of the same active polarity.

- SetReset/gated_off_during_shift_inside : on | off | auto;

  A property that defines that the related SetReset ports are gated off during scan chain shifting. The "off" value adds the related instance pins to the DFT_C9 checks to ensure that the interface on the described core with embedded scan chains connects well with other logic at the levels above. The "auto" value enables the same if the related module has an "is_blackboxed" attribute and has no "mode" wrapper.

- ScanChain/length : int | auto;

  A property to define the length of the scan chain. The tool only allows the value auto when the traceable property is 1. When the value is an integer and traceable is 1, it generates a warning if the traced length differs from the specified length, and the traced length is used. The minimum scan chain length is one scan cell.

- ScanChain/mux_si_functional_source : 1 | 0;

  A Boolean property to specify if the scan chain allows its scan_in_port to be preconnected when used as an existing segment at the next level. When set to 1, the mux_si_functional_source property instructs the scan insertion tool to omit S6 rule checking for this chain segment and to multiplex any functional source on its scan-in during scan chain insertion. The tcd_scan data created by scan insertion automatically sets this property to 1 if the scan chain has its top scan-out connection muxed with a functional source. When set to 0, the tool behaves normally and enforces S6 rule checking on the chain segment.

- ScanChain/no_si_pipelining : 1 | 0;

  A Boolean property to specify if the scan chain allows the insertion of a pipelining flip-flop in front of the scan_in_port. When set to 1, the no_si_pipelining property overrides any pipelining instructions set by the set_scan_insertion_options -si_pipelining command specified with "all_chains" or "wrapper_chains_only".

  Scan chains defined inside tcd_scan wrappers and having module_type equal to occ have no_si_pipelining inferred by the tool to 1. The tcd_scan data created by scan insertion automatically sets this property to 1 if the scan chain starts with a scan element having the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

no_si_pipelining property set or inferred to 1, or when a pipelining flip-flop is already inserted inside the core.

When set to 0, the tool does not prevent the insertion of a pipelining flip-flop on the scan chain in front of the scan_in_port and follows the direction set by the set_scan_insertion_options -si_pipelining command.

- ScanChain/*scan_in_port* : *port_name*, ...;

  A property to define the name of the scan-in port for the chain.

- ScanChain/*scan_out_port* : *port_name*, ...;

  A property to define the name of the scan-out port for the chain.

- ScanChain/*scan_in_clock* : *port_name*, ...;

  A property to define the name of the clock used by the first scan flop of the chain. The tool uses a "~" prefix to indicate an inversion between the clock pin/port and the first scan flop. This property is optional when traceable is 1. The tool generates a warning if it extracts the clock of the first scan element of the chain and finds it to be different than the specified one. The extracted one is used when they differ.

  Additionally, the tool uses a "%" prefix to indicate that the scan_in_clock port or pin is unbalanced with respect to the clock used by the first scan element of the chain.

- ScanChain/*scan_out_clock* : *port_name*, ...;

  A property to define the name of the clock used by the last scan flop of the chain. The tool uses a "~" prefix to indicate an inversion between the clock pin/port and the last scan flop. This property is optional when traceable is 1. The tool generates a warning if it extracts the clock of the last scan element of the chain and finds it to be different than the specified one. The extracted one is used when they differ.

  Additionally, the tool uses a "%" prefix to indicate that the scan_out_clock port or pin is unbalanced with respect to the clock used by the last scan element of the chain.

- Iproc(*proc_name*)

  This wrapper defines the name of an iProc that must be used to activate the OCC. This iProc is automatically iCalled from the parent level when the OCC is still active in the external test modeof the child. If you are using the Tessent OCC, this is automatically populated with the name "setup".

- Iproc(*proc_name*)/parameter_value_list : *parameter_value*, ... ;

  This wrapper defines the parameters to used when calling the iProc mentioned in the parent wrapper. If you are using the Tessent OCC, this is automatically populated with this parameter value pair: "fast_capture_mode, 1".

- ForcedDuringPreScanTracing/*port_or_pin_name* : *constrained_value*;

  A repeatable property and value pair to define the constrained value on a given port or pin during pre-scan tracing. The *constrained_value* can be 0, 1, or X.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The TCD below shows examples of the scan_in_clock/scan_out_clock properties. Sometimes the <clock_port_name> is preceded by a ~ and/or a %, and sometimes it is not.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Core(TOP) {
  Scan {
    allow_internal_pins : 1;
    is_hard_module : 1;
    exclude_from_concatenated_netlist : 1;
    internal_scan_only : 1;
    IntTestEn(int_ltest_en) {
      active_polarity : all_ones;
    }
    ExtTestEn(ext_ltest_en) {
      active_polarity : all_ones;
    }
    Mode(internal) {
      type : internal;
      traceable : 1;
      make_active_automatically : 1;
      ModeEn(ts_stm0s0,ts_stm1s0) {
        active_polarity : 2'b10;
      }
      ScanChain {
        length : 36;
        scan_in_clock : clk2;
        scan_out_clock : ~clk2;
        scan_in_port : ts_internal_si[0];
        scan_out_port : ts_internal_so[0];
      }
      ScanChain {
        length : 20;
        scan_in_clock : clk2;
        scan_out_clock : ~clk2;
        scan_in_port : ts_internal_si[1];
        scan_out_port : ts_internal_so[1];
      }
      ScanChain {
        length : 21;
        no_si_pipelining : 1;
        scan_in_clock : clk2;
        scan_out_clock : ~%clk3;
        scan_in_port : ts_internal_si[2];
        scan_out_port : ts_internal_so[2];
      }
      ScanChain {
        length : 7;
        no_si_pipelining : 1;
        scan_in_clock : %clk3;
        scan_out_clock : ~clk1;
        scan_in_port : ts_internal_si[3];
        scan_out_port : ts_internal_so[3];
      }
      ScanChain {
        length : 3;
        scan_in_clock : clk2;
        scan_out_clock : ~clk1;
        scan_in_port : ts_internal_si[4];
        scan_out_port : ts_internal_so[4];
      }
      ScanEn(scan_en) {
        active_polarity : all_ones;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
        }
        Clock(clk1) {
          off_state : 1'b0;
        }
        Clock(clk2) {
          off_state : 1'b0;
        }
        Clock(clk3) {
          off_state : 1'b0;
        }
        ChildBlockInstance(uA1) {
          child_mode_name : "";
          enable_dft_signal : "";
        }
        ForcedDuringPreScanTracing {
          port_en1 : 0;
          pin_en2  : 1;
        }
      }
    }
  }
```

Related Topics

add_scan_chains

read_core_descriptions

set_design_sources

set_module_matching_options

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 15
Tessent Shell Data Base (TSDB)

This chapter describes the Tessent Shell Database which is a structured directory containing subdirectories and files. The subdirectory naming is designed to group related information for each specific view of each design block. The complete subdirectory is deleted and recreated cleanly when a step is repeated thereby ensuring that no stale files from a previous run can remain in the active TSDB directory.

The root_directory name can be anything and located anywhere, and is specified using the set_tsdb_output_directory command. You can use a single root directory as the output of all of your design blocks but you can use as many as you like. As long as you make the TSDB directories associated to child blocks visible (using the open_tsdb command) to the tool when processing a given block, it does not matter if you have everything consolidated into one directory or distributed into many.

root_directory 4807
instruments 4817
dft_inserted_designs 4820
patterns 4827
logic_test_cores 4829
<design_name>.lbist_mode_<mode_name> 4830
<design_name>.atpg_mode_<mode_name> 4832
<design_name>.atpg_retargeting_mode_<mode_name> 4833

root_directory

A directory that groups up to four subdirectories containing related information.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Summary

Figure 15-1. Root Directory Structure

Description

A directory that groups up to four subdirectories containing related information. The *root_directory* is specified using the set_tsdb_output_directory command. The subdirectories are created by the first command that needs to create a subdirectories inside them.

- instruments — This directory is the first time you issue the process_dft_specification or the "write_edt_files -tsdb" command and switch.
- dft_inserted_designs — This directory is created when you do either of the following:
  - The first time you issue the process_dft_specification command without specifying the -no_insertion option.
  - The first time you issue the "extract_icl -write_in_tsdb" command and switch.
- patterns — This directory is created the first time you issue the process_patterns_specification command.
- logic_test_cores — This directory is created the first time you issue the write_tsdb_data command in the patterns -scan context (see "Contexts and System Modes" in the *Tessent Shell User's Manual*). Currently, only the LogicBIST patterns are supported by the write_tsdb_data command.

How to Relocate the TSDB Directories

A complete chip design has several TSDB directories containing the data of several physical and sub-blocks of the design. Figure 15-1 shows that the root directory of a TSDB can be

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

anything but the naming of the inner directories is specific to a certain *design_name* and *design_id* the data is for.

The set_tsdb_output_directory command defines the name of the root directory with *tsdb_outdir* as the default name. Given that you must expose all TSDB root directories belonging to a chip to the tool when you get to the top level, we recommend you name your TSDB root name consistent with how you generate them.

If you store all TSDB data in a single directory, no matter which block you are working on, you can use *<chip_name>.tsdb* as your TSDB output directory name. This only works if a single person using a single DFT flow implements DFT in all blocks.

If more than one person inserts DFT in your chip, we recommend you use one TSDB per block or even one TSDB per block/per design_id if they are not done at the same time by the same person.

For example, consider a physical block, "b1" with two DFT insertion passes. Harry does the first pass in New York (NY), and Sally does the second in Los Angeles (LA). They each have their own file system, and the TSDB directory created by Harry is exported to the LA file system but with in a location specific to the LA file system. The TSDB created by Harry is likely read-only for Sally. In such cases, it is best for Harry to name the TSDB *b1_rtl1.tsdb* and for Sally to name it *b1_rtl2.tsdb*. If Harry implements both DFT insertion passes back to back, then *b1_rtl.tsdb* would be a logical name for the TSDB root directory.

Several enhancements to the TSDB infrastructure enable you to relocate TSDBs and still use the read_design command to load the design into the tool, the run_testbench_simulations command to simulate your testbenches, and the write_design_import_script command to load your RTL design into your synthesis tool. The tool stores information used by those commands in the *<design_name>.design_source_dictionary* file. (See the dft_inserted_designs section for more details on the *<design_name>.design_source_dictionary* file.) This file includes three types of file paths:

- Absolute paths to unmodified design files from the original repository.
- Relative paths to newly modified design and newly created instrument files stored in the current TSDB directory.
- Logical paths to previously modified design and previously created instrument files stored in different TSDB directories. These only exist when you perform more than one insertion pass on a given block, and each pass uses different TSDB output directories.

Using different paths for different object types is the key to allowing easy TSDB relocations. See the detailed example later in this section to understand how this works and how to use the "set_tsdb_options -directory_path_map" switch to specify a mapping when the absolute path to the design repository is different at different sites. You can avoid using a mapping by locating the design repository at the same place in all sites.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

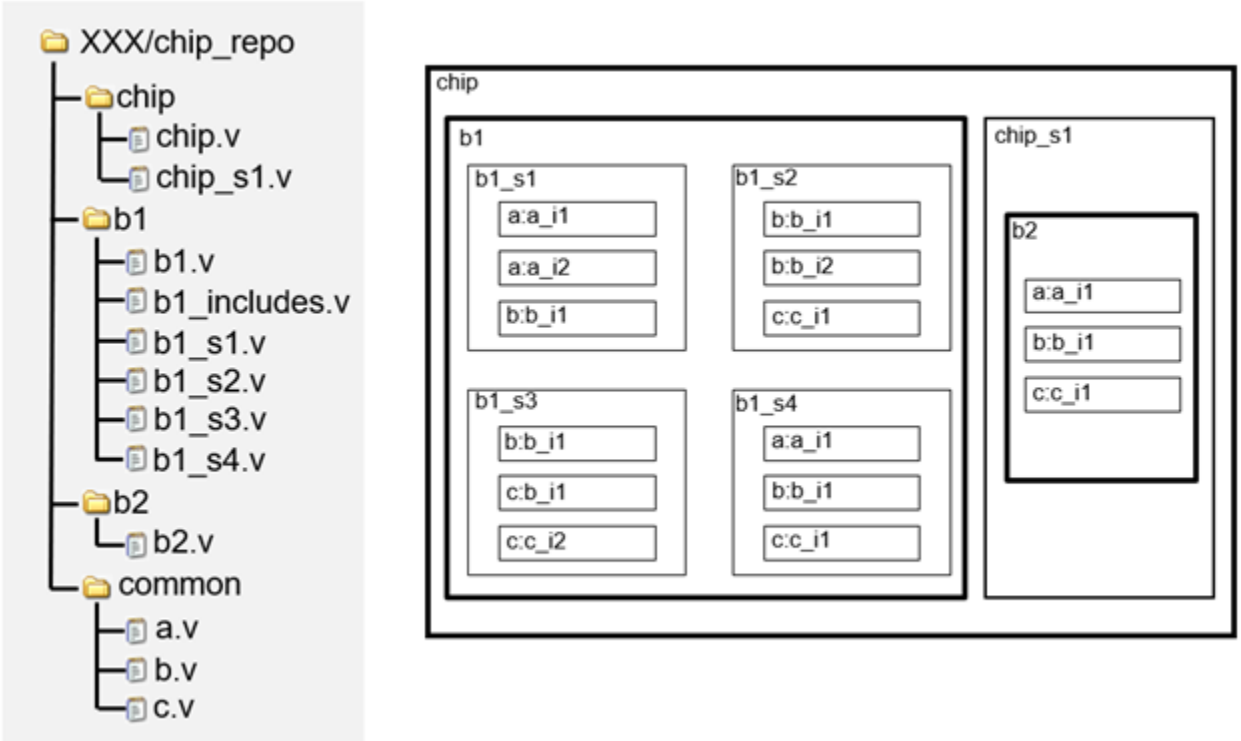
Note

You can use soft links in different sites to point to the design repository. If you read your design files through those soft links, the tool preserves the soft link names in the absolute paths stored in the <*design_name*>.*design_source_dictionary* file, removing the need to use the "set_tsdb_options -directory_path_map" command to map them.

Example

Figure 2 shows a simple design for this example.

Figure 15-2. Example Chip Design Process in Different Sites



The design consists of a chip with two child physical blocks: b1 and b2. The *chip_repo* directory on the left shows the file organization used for this design. A subdirectory called *common* includes RTL files that the two child physical blocks share. This requires using the "set_insertion_options -edited_module_prefix" switch to specify a different prefix for each physical block.

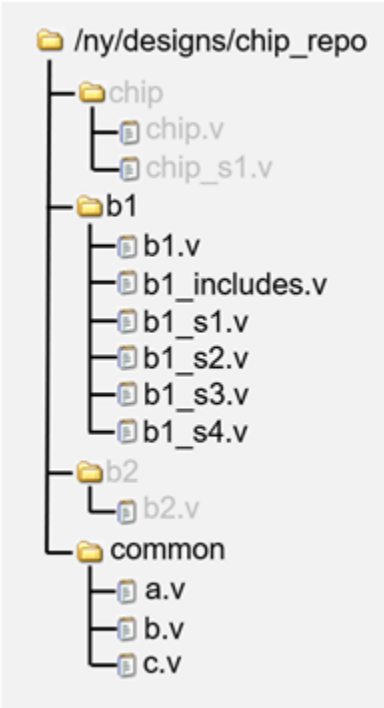
The b1 child physical block undergoes two DFT insertion passes, Harry does the first insertion pass in New York, and Sally does the second insertion pass in Los Angeles. They relocate the TSDBs from where they were created to where they are used in different locations but preserve complete TSDB automation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

First Insertion Pass of b1 Implemented by Harry in New York

Harry is located in New York (NY). The left-hand side of Figure 15-3 shows that Harry has checked out the design repository in the */ny/designs/chip_repo* directory.

Figure 15-3. First Insertion Pass: b1 Loading




```
set design /ny/designs
read_verilog ${designs}/chip_repo/b1/b1.v \
             ${designs}/chip_repo/b1/b1_s1.v \
             ${designs}/chip_repo/b1/b1_s2.v \
             ${designs}/chip_repo/b1/b1_s3.v \
             ${designs}/chip_repo/b1/b1_s4.v

read_verilog ${designs}/chip_repo/common/a.v \
             ${designs}/chip_repo/common/b.v \
             ${designs}/chip_repo/common/c.v

set_current_design b1

set_design_level physical_block
set_insertion_options -edited_module_prefix b1

set_tsdb_output_directory b1_rtl1.tsdb
```

The right-hand side of Figure 15-3 shows the dofile commands Harry uses to read the b1 physical block into the tool. The tool reads all the files from the *b1* directory and some from the *common* directory.

The list of files needed to elaborate the b1 physical block is specified once during the first insertion pass. After that, they will use the read_design command.

Caution

It is critical only to read files used by the b1 physical block and not those exclusively used by other physical blocks. As this example shows, you can read the same files when you use child modules in multiple physical blocks. Do not read in the file defining the top module of other physical blocks or the chip level for b1; otherwise, the tool incorrectly associates them with the b1 block, which breaks the rest of the flow.

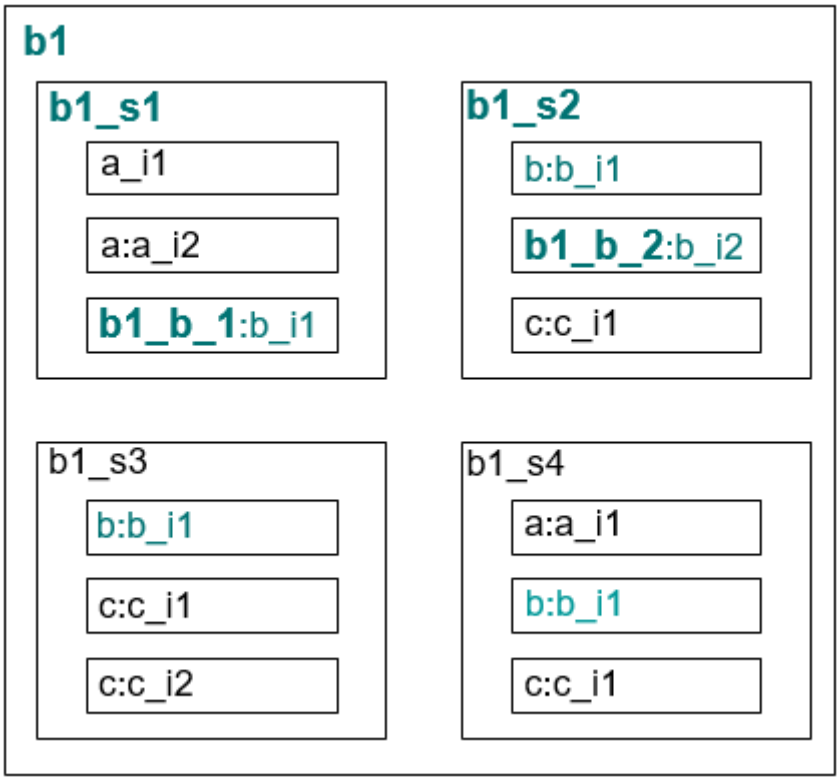
Because the b1 physical block uses RTL modules that other physical blocks also use, Harry uses the set_insertion_options command to specify an edited_module_prefix equal to “b1_”. This ensures that if you edit the a, b, or c modules within their b1 context, they are prefixed with “b1_” to not conflict with similar edits done in the context of other physical blocks.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Also, note the use of the set_tsdb_output_directory command to set the TSDB output directory to *b1_rtl1.tsdb*. Because Harry only intends to perform the first insertion pass for the b1 physical block, he uses *<design_name>_<design_id>.tsdb* to name the TSDB. Had he been doing all insertion passes on the RTL view, he could have used *<design_name>_rtl.tsdb*.

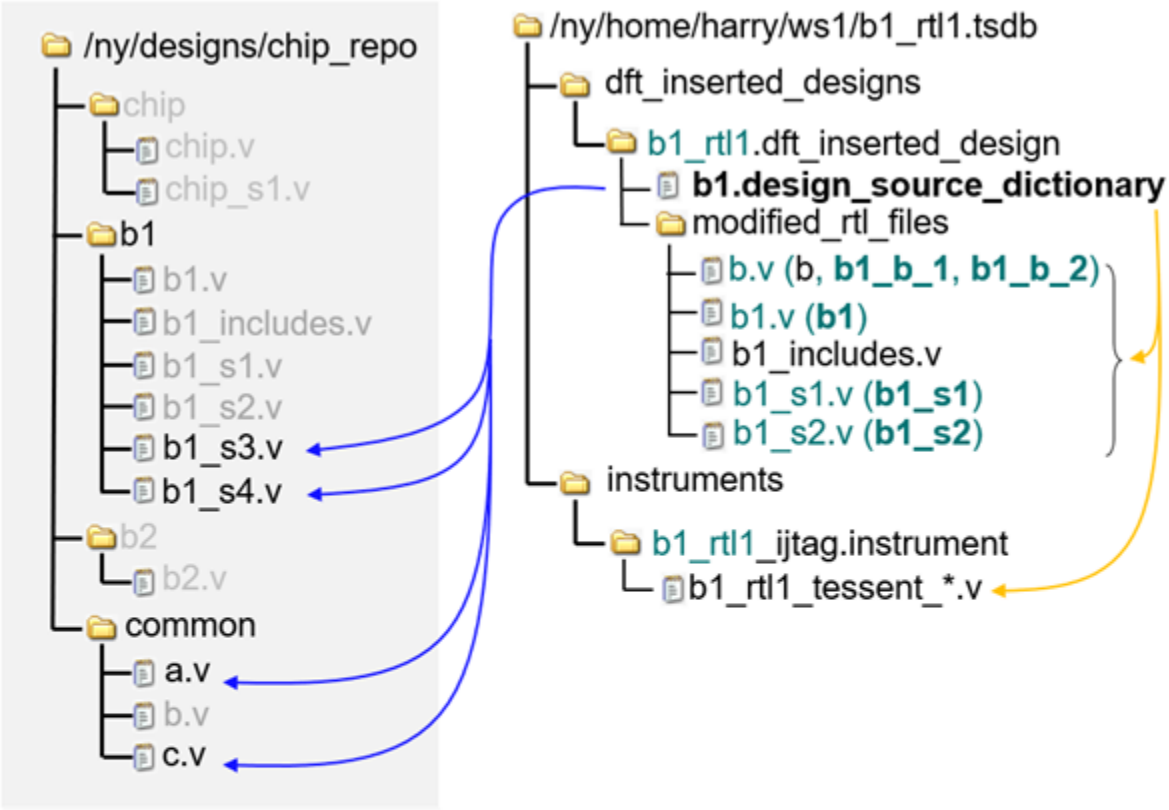
Harry's DftSpecification results in the insertion of DFT logic within the b1_s1/b1_i1 and b1_s2/b1_i2 instances. Connections from those elements to the top module also cause b1, b1_s1, and b1_s2 to be modified. Figure 15-4 shows the resulting design view and a TSDB output directory, shown in Figure 15-5.

Figure 15-4. Modified b1 rtl View After First Insertion Pass



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 15-5. Design Source Dictionary After First Insertion Pass




```
./modified_rtl_files/b1.v
./modified_rtl_files/b1_s1.v
./modified_rtl_files/b1_s2.v
/ny/designs/chip_repo/b1/b1_s3.v
/ny/designs/chip_repo/b1/b1_s4.v
/ny/designs/chip_repo/common/a.v
./modified_rtl_files/b.v
/ny/designs/chip_repo/common/c.v
../../instruments/b1_rtl1_ijtag.instrument/b1_rtl1_tessent_sib_1.v
../../instruments/b1_rtl1_ijtag.instrument/b1_rtl1_tessent_sib_2.v
```

Bold names in Figure 15-4 indicate modified instances. The teal color indicates that the newly created TSDB output directory now stores the definition of those modules. There are several instances of the b module, so the two edited instances were first uniquified using the specified *b1_* prefix and the *_#* suffix. See the set_insertion_options command in the dofile for the switch names used. The *b.v* file in the TSDB output directory contains all three versions of the b module: the original b version and the two uniquified versions.

Figure 15-5 shows the TSDB output directory created by the first insertion pass. The *.dft_inserted_design* directory contains the *b1.design_source_directory* file, which stores the current view of the design. (See the format of this dictionary in the dft_inserted_designs

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

section.) The color-coded arrows show the pointer to the files used, and the file path names represented by the arrows are listed at the bottom with the same color scheme.

The path to files that were not modified and remain in the chip repository uses absolute paths (shown in blue), while the path to files local to the newly created TSDB is relative (shown in orange). The use of absolute and relative paths enables you to relocate the TSDB and the design source dictionary to remain valid, as shown in the next step.

Second Insertion Pass of "b1" Implemented by Sally in Los Angeles

Sally is in Los Angeles, and has checked out the design repository in the */la/design/chip_repo* directory, as shown in Figure 15-6.

Figure 15-6. Second Insertion Pass: b1 Loading

Because this is not the first insertion pass on this block, Sally checked out its TSDB in the */la/tsdbs* directory. The absolute path to the design repository in their company is not the same across sites. Sally uses the "set_tsdb_options -directory_path_map" switch to map the absolute path starting with */ny/designs* to */la/designs*. Having opened the TSDB from where it is currently located, Sally uses the read_design command to load it into the tool.

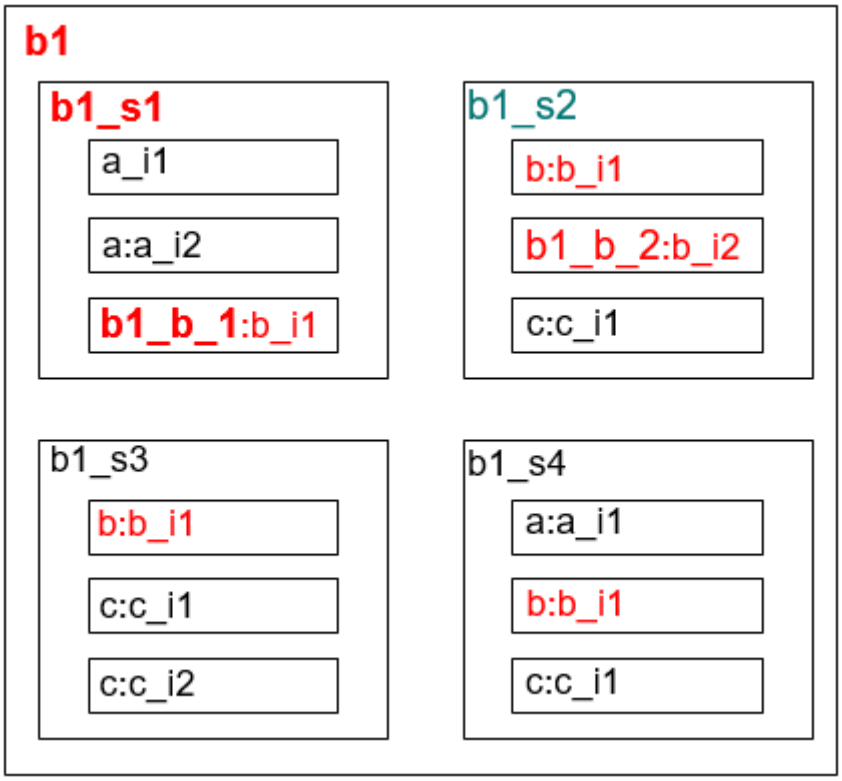
If the chip repository was at the same place in all sites, Sally would not need the "set_tsdb_options -directory_path_map" switch. Even if the chip repository physically resides at different places, you can use a common link in all sites that points to their respective physical local location. Then, the absolute paths stored in the design source dictionary maintain the link name, making them work in all sites.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For example, if you had a link called */designs* pointing to */ny/designs* and */la/designs* in their respective sites, and Harry had read the Verilog files through that link, the absolute path written out in New York would remain valid in Los Angeles and would not need to be mapped.

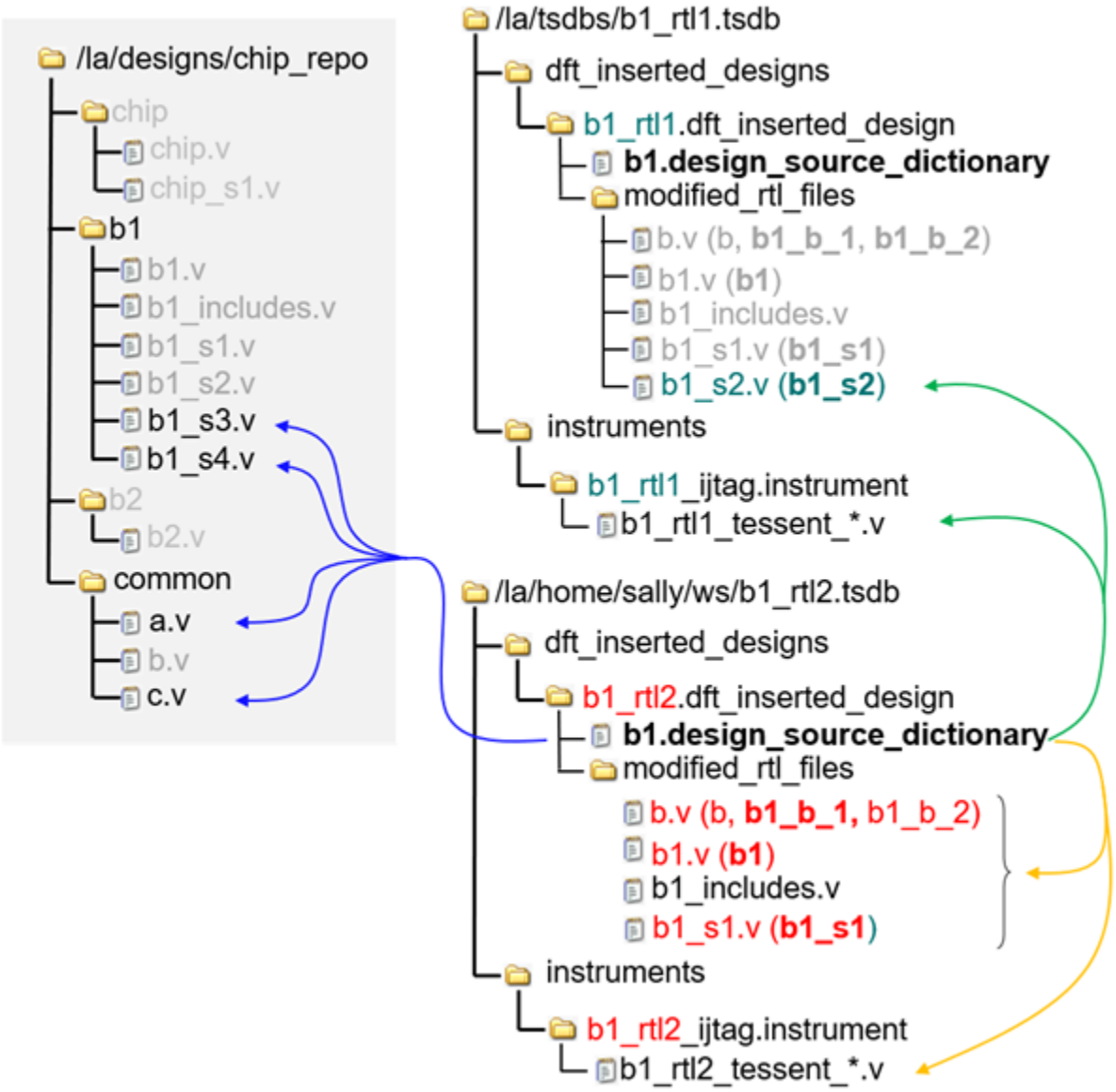
The specific DftSpecification used by Sally results in the insertion of DFT logic within the b1_s1/b1_i1 instance for a second time. Connections from those elements to the top module also cause b1 and b1_s1 to be modified. This results in a design view illustrated in Figure 15-7 and a TSDB output directory shown in Figure 15-8. Because Harry already uniquified the module of the b1_s1/b1_i1 instance in pass one, the tool does not need to uniquify it. Also, because it already had the b1_ prefix, the tool did not need to add it.

Figure 15-7. Modified b1 rtl View After Second Insertion Pass



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 15-8. Design Source Dictionary After Second Insertion Pass



Bold names in Figure 15-7 indicate modified instances. The red color indicates that the newly created TSDB output directory now stores the definition of those modules. The teal color indicates that the module's definition remains in the previous TSDB.

Figure 15-8 shows the TSDB output directory created by the second insertion pass. The *.dft_inserted_design* directory contains the *b1.design_source_directory* file, which stores the current view of the design. (See the format of this dictionary in the dft_inserted_designs section.) The color-coded arrows show the pointer to the files used. The file path names are listed in Figure 15-9.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 15-9. File Paths in Design Source Dictionary After Second Insertion Pass

```
./modified_rtl_files/b1.v
./modified_rtl_files/b1_s1.v
dft_inserted_design(b1_rtl1)/modified_rtl_files/b1_s2.v
/la/designs/chip_repo/b1/b1_s3.v
/la/designs/chip_repo/b1/b1_s4.v
/la/designs/chip_repo/common/a.v
./modified_rtl_files/b.v
/la/designs/chip_repo/common/c.v
instrument(b1_rtl1_ijtag)/b1_rtl1_tessent_sib_1.v
instrument(b1_rtl1_ijtag)/b1_rtl1_tessent_sib_2.v
../../instruments/b1_rtl2_ijtag.instrument/b1_rtl2_tessent_sib_1.v
```

The path to files that were not modified and remain in the chip repository uses absolute paths (shown in blue), while the path to files local to the newly created TSDB is relative (shown in orange).

Compared with the first pass results (Figure 15-5), the design source dictionary now also includes symbolic paths to files that remained in the previous TSDBs (shown in green). You only have symbolic paths in your design source dictionary when the subsequent insertion passes on a given block do not use the same TSDB output directory as the previous one.

You can use the map_tsdb_container_paths command if you need to convert a symbolic path name to an absolute path. The separation of those different files into absolute, relative, and symbolic paths enables you to relocate the TSDB and the design source dictionary to remain valid.

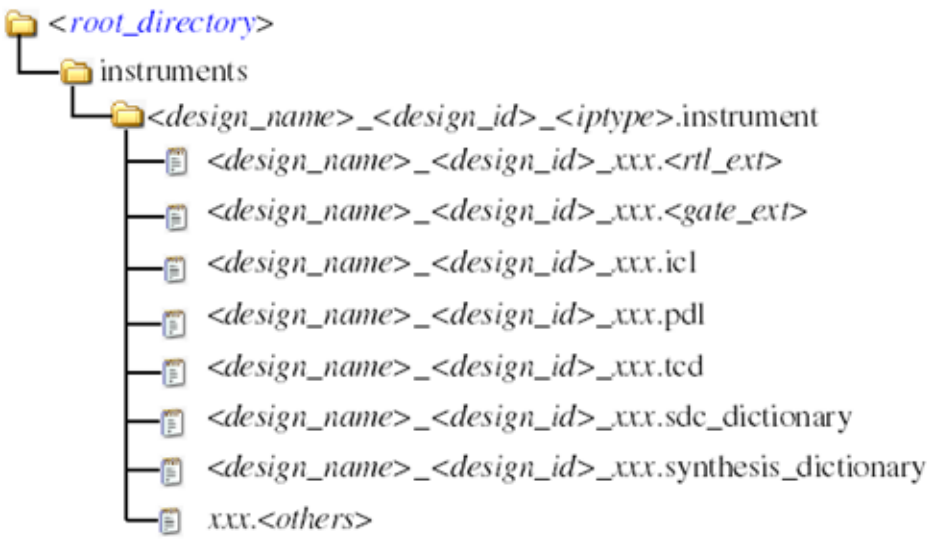
If this is the last insertion pass for this block in RTL, Sally will use the write_design_import_script command to export a design load script for the synthesis tool. The run_testbench_simulations command manages the Verilog simulations, and it uses the design source dictionary to manage the design view needed by the simulation.

instruments

An instruments directory that contains a series of subdirectories that are referred to as “instrument containers”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Summary



Description

The instruments directory contains a series of subdirectories that are referred to as “instrument containers”. The instrument container names are formed by concatenating the name of the design, the design identifier, and a unique string that is specific to the instrument or IP type. Table 15-1 shows the string used for the various instrument types supported by the tool. See the “set_context -design_identifier” command to learn more about the design identifier string.

You can only have one instrument container per design_name/design_id/ ip_type combination visible in the opened TSDBs. If you have multiple TSDB directories containing the output of multiple runs on the same design with the same design id, and you try to open a second TSDB using the open_tsdb command, the tool reports an error if the new TSDB exposes a second copy of the same instrument container. If this occurs, you need to close the first TSDB before opening the second one. Also note that the current TSDB output directory, specified with the set_tsdb_output_directory command, is always open.

Table 15-1. Instrument Container Names

Instrument Type	Instrument container name
Boundary Scan	<design_name>_<design_id>_bscan.instrument
EDT	<design_name>_<design_id>_edt.instrument
Hybrid EDT/LBIST	<design_name>_<design_id>_edt_lbist.instrument
IJTAG Network	<design_name>_<design_id>_ijtag.instrument
Logic Bist	<design_name>_<design_id>_lbist.instrument
LPCT	<design_name>_<design_id>_edt.instrument

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 15-1. Instrument Container Names (cont.)

Instrument Type	Instrument container name
Memory BISR	<design_name>_<design_id>_mbisr.instrument
Memory BIST	<design_name>_<design_id>_mbist.instrument
ncp_index_decoder	<design_name>_<design_id>_lbist_ncp_index_decoder.instrument
OCC	<design_name>_<design_id>_occ.instrument
RTL cells	<design_name>_<design_id>_cells.instrument

Components

- <design_name>_<design_id>_xxx.<rtl_ext>

  One or many files containing the Verilog RTL description of the instrument. The extension defaults to “v” but is user-specified using the rtl_extension property in the DftSpecification.

- <design_name>_<design_id>_xxx.<gate_ext>

  One or many files containing the Verilog netlist description of the instrument. The extension defaults to “vg” but is user-specified using the gate_extension property in the DftSpecification. This view of the instrument is only generated when synthesizing the instrument using the run_synthesis command which is only used when the -no_rtl insertion flow is used. In the -rtl insertion flow, the instruments are synthesized in context along with the rest of the functional logic.

- <design_name>_<design_id>_xxx.icl

  One or many files containing the ICL descriptions of the instruments. The extension is always “icl”. These ICL descriptions are collected into a unified ICL file and stored in a dft_inserted_designs subdirectory when running ICL extraction.

- <design_name>_<design_id>_xxx.pdl

  One or many files containing the PDL iProcs for the instruments. The extension is always “pdl”. The PDL files contain an iProcsForModule statement referring to an ICL module name followed by one or many iProcs. The PDL files are concatenated into a single file and stored beside the unified ICL file when running ICL extraction.

- <design_name>_<design_id>_xxx.tcd

  One or many files containing extra information about the instrument. The “tcd” extension stands for “Tessent Core Description”. This file uses the configuration data format to encode extra information needed while using the instrument. For example, memory BIST uses this file to store the complete information about the memories tested by the controller such that precise diagnosis of failures can be done later. The TCD configuration file consists of a wrapper Core(<module_name>) with a sub-wrapper specific to the instrument type. Again, for example, memory BIST creates a

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

"Core(<ctrl_mod_name>)/MemoryBistController" to store all of the information needed to diagnose failures on memories tested by a given controller.

- <design_name>_<design_id>_xxx.sdc_dictionary

  One or many files containing a Tcl dictionary capturing timing exceptions associated with the instrument modules. The instance-specific information ends up in the extracted ICL file. The true SDC procs ends up being created in the *dft_inserted_designs* subdirectory when running ICL extraction (using the extract_icl or set_system_mode commands in analysis mode).

- <design_name>_<design_id>_xxx.synthesis_dictionary

  One or many files containing a Tcl dictionary capturing tool-independent synthesis instructions. The information is used by the run_synthesis command to understand which modules need to be synthesized, which RTL files to read, and which netlist files to write out. This file is created systematically but is typically only created when using the -no_rtl insertion flow. When using the rtl insertion flow, the run_synthesis is not used and the instruments are synthesized in context along with the rest of the functional logic.

- xxx.<others>

  Optional extra files having custom extensions specific to each instrument type that enable features that are specific to each of them. For example, the boundary scan instrument creates a BSDL file for the current design with the extension ".bsdl" to describe the boundary scan implementation.

Related Topics

DftSpecification

PatternsSpecification

process_dft_specification

write_edt_files

process_patterns_specification

extract_icl

run_testbench_simulations

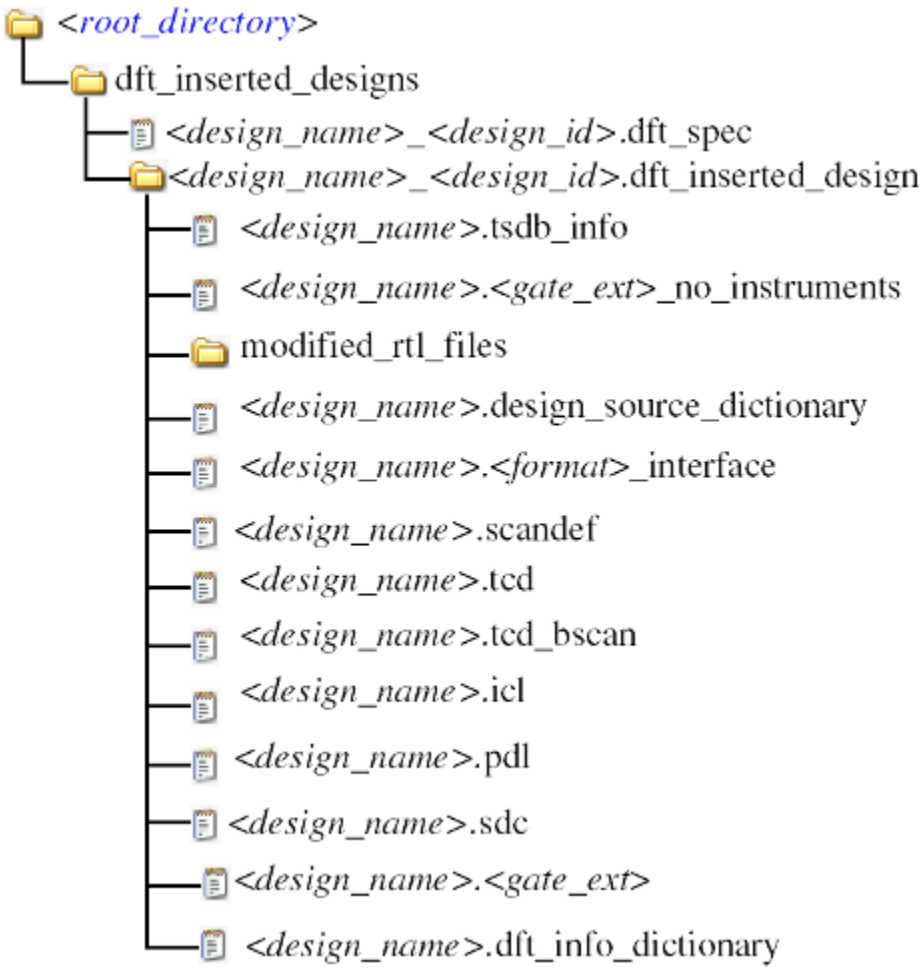
run_synthesis

dft_inserted_designs

A directory that contains one or many subdirectories that capture the information relative to a design in which some DFT has been inserted.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Summary



Description

The dft_inserted_designs directory contains one or many subdirectories capturing the information relative to a design in which some DFT has been inserted. The name of the subdirectory is formed by concatenating the name of the design with the design identifier followed by the ".dft_inserted_design" extension. See the set_context -design_identifier option description to read about the design identifier string value.

The directory is created and populated when the process_dft_specification command runs unless the -no_insertion option is specified. The directory can also be created by the "write_edt_files -tsdb" commands. Finally, if it was not already created by the process_dft_specification or the write_edt_files commands, the directory can be created by the extract_icl command. The creating step is stored in the *.tsdb_info* file such that the tool knows to delete the directory completely before recreating it using the step that originally created it. This ensures there can never be any stale information in the .dft_inserted_design directory as you repeat the process.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You can only have one .dft_inserted_design directory per design_name/design_id combination visible in the opened TSDBs. If you have multiple TSDB directories containing the output of multiple runs on the same design with the same design id, and you try to open a second TSDB using the open_tsdb command, the tool issues an error if the new TSDB exposes a second copy of the same .dft_inserted_design directory. If this occurs, you need to close the first TSDB before opening the second one. Also note that the current TSDB output directory, specified with the set_tsdb_output_directory command, is always open.

Components

- <design_name>.tsdb_info

  A file that is created when the .dft_inserted_design directory is created to store information about the design view and the creation step. The format of this file is shown below.

```
TsdbInfo(module_name,design_id) {
  creation_date         : date;
  creation_user         : user_id;
  creation_step         : process_dft_specification | extract_icl |
                          write_edt_files;
  level                 : chip | physical_block | sub_block;
  icl_extraction_needed : on | off;
  library_name          : logical_library_name;
  OpenedTsdbDirectories {
    path_to_tsdb_directory ; // repeatable
  }
  ChildBlockModules {
    module_name : design_id;
  }
}
```

  You can read this file into memory using the read_config_data and introspect its content using the get_config_value command.

  This file may grow over time as more information may be needed to automate new flows.

  This file is created when the process_dft_specification runs unless the -no_insertion option is specified. It can also be created by the “write_edt_files -tsdb” commands. Finally, the file can be created by the extract_icl command, if it was not already created by the process_dft_specification or the write_edt_files commands.

- <*design_name*>.<gate_ext>_no_instruments

  A Verilog netlist file, created when you use the “set_context -no_rtl” flow, that contains all of the design files forming the current physical block. The netlist of sub-blocks are included in this netlist but the netlist from child physical blocks is not included. See the set_design_level command for more information about the block types.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note, this concatenated netlist does not include the netlist of the instruments inserted in the current insertion pass because they need to be synthesized prior to being included in the netlist.

This file is created by process_dft_specification when the -no_insertion option is not used, or by the "write_edt_files -tsdb" command.

- modified_rtl_files

  A directory, created when using the "set_context -rtl" flow, that contains the RTL files that were edited during the current insertion pass. In the RTL flow, the file structure of the original RTL must be preserved. Because RTL supports five Verilog formats and four VHDL formats, they cannot be mixed into a common file because each file needs to be read with a unique read_verilog -format or read_vhdl -format command. Also, because of compiler directives that persist from one file to another, the file compilation order must be preserved to ensure that the proper design view is loaded. The list of modified files is taken and their common ancestor directory is computed. The relative path between every file and the common ancestor directory is preserved in the modified_rtl_files directory. For example, assume the following three files were modified: /a/b/c/f1, /a/b/c/f2, and /a/b/d/f3. The common ancestor is "/a/b". The subdirectory "c" is created in the modified_rtl_files directory to store f1 and f2, and the subdirectory "d" is created in the modified_rtl_files directory to store f3. If included files are edited, their filenames are uniquely modified and stored in the subdirectory called "INCLUDE".

  This directory is created by the process_dft_specification command when the -no_insertion option is not used.

- <*design_name*>.design_source_dictionary

  A file that contains a Tcl dictionary describing precisely how to load the current design. The directory consists of a series of ordered Read# entries containing the complete information to reproduce it. The information is general enough to be translated into read commands for any tool. The run_testbench_simulations command uses that information to create the compilation script for the supported simulators.

  This file is created by the process_dft_specification command when the -no_insertion option is not used, or by extract_icl if not already created by the process_dft_specification command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The format of the Read# entries is shown below:

```
read# { // read1 for the first read, read2 for the second, etc.
  format { // See read_verilog/read_vhdl -format option
    "verilog_1995" | "verilog_2001" | "verilog_sv31a" |
    "verilog_sv2005" | "verilog_sv2009" | "vhdl_1987" |
    "vhdl_1993" | "vhdl_2002" | "vhdl_2008"
  }
  work_library { //See read_verilog/read_vhdl -in_library option
    "<library_name>"
  }
  files {
    "<relative_filename>"
    ...
  }
  libraries { // See set_design_sources command
    {directory | file <relative_directory_or_filename>}
  }
  extensions { // See set_design_sources command
    <extension list used for searching in directory libraries>
  }
  defines { // See set_design_macros command
    "<macro_name>[=<value>]
    ...
  }
  incdirs { // See set_design_include_directories command
    <relative_directory_name>
    ...
  }
  options { // See allowed options in read_verilog and read_vhdl
            // commands
    <option_used_in_read_command>
    ...
  }
}
```

- <*design_name*>.<format>_interface

  A file that contains only the port and parameter declarations of the current design. The format of the file is encoded in the extension such that the right format can be used to load it back. The table below shows the mapping between the extension and the format.

  This file is created by the process_dft_specification command when the -no_insertion option is not used, or by extract_icl if it is not already created by process_dft_specification command.

  When you use the set_current_design command, the tool automatically reads in the most recent *design_name.format*_interface file using the "read_verilog -interface_only" or "read_vhdl -interface_only" command if the corresponding *design_name* does not already exist in memory. This enables you to only load the modules associated with the current design and have the interface view of all child physical or sub-blocks loaded automatically.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The set_current_design command does not automatically load a child block that has the interface_has_external_dependencies property set to 1 in the *.tsdb_info* file of the child block. The interface_has_external_dependencies property is set to "1" if the port list of the interface has external dependencies such as typedef packages or System Verilog interfaces. If you want to load that child block, use the "read_design *child_block* -view interface" command.

Table 15-2. Format to Extension Mapping for the Interface View

Format	extension
verilog 1995	v95_interface
verilog 2001	v_interface
verilog sv31a	sv31a_interface
verilog sv2005	sv05_interface
verilog sv2009	sv09_interface
vhdl 1987	vhd87_interface
vhdl 1993	vhd_interface
vhdl 2002	vhd02_interface
vhdl 2008	vhd08_interface

- <*design_nane*>.scandef

  A file that is created by insert_test_logic -write_in_tsdb on. It contains the scandef description of the scan chains so that you can perform scan chain reordering and re-partitioning during layout.

- <*design_name*>.tcd

  A file that is created by the process_dft_specification, insert_test_logic -write_in_tsdb on, and write_design -tsdb commands. It contains the DesignInfo and optionally the Scan wrappers within the Core(*design_name*) wrapper. The DesignInfo includes lots of data specified or created in an insertion pass so that it can be restored at the beginning of a future insertion pass. The file is always read-in when using the read_design command no matter which view is loaded ensuring that the DesignInfo is imported. The Scan wrapper describes the scan chains that were created by the insert_test_logic command and is reused in a hierarchical scan insertion process.

  The information stored in the Scan wrapper is also used by the import_scan_mode command to import the clock, chain, and load_unload definitions of the scan mode such that it can be used to create ATPG modes.

- <*design_name*>.tcd_bscan

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

A file that is created by the process_dft_specification command when inserting a Boundary Scan chain into a sub or physical block and the block contains a Core(<module_name>)/BoundaryScan wrapper. This file is reused at the next level up to stitch the boundary scan chain in the parent module and eventually generate the BSDL file once the chip level is reached. See the set_design_level command for a description of the design levels.

- <*design_name*>.icl

  A file that contains the concatenated ICL representation of the current design. It is generated when running the extract_icl command.

- <*design_name*>.pdl

  A file that contains the content of all PDL files that were seen beside the input ICL and tcd_memory_lib files when running the extract_icl command. For every read-in ICL or tcd_memory_lib file, a PDL file having the same name and location but with the .pdl extension are loaded and concatenated into the merged PDL file. These PDL files must only contain a series of iProcsForModules and iProcs statements. The automatic collection of PDL files makes it easy to place a PDL file beside any ICL file and have it automatically collected during ICL extraction and automatically loaded after ICL elaboration so that it can be used in a ProcedureStep wrapper or with the set_test_setup_icall command.

  The source of each imported file is marked with a comment specifying where it was taken from. Because ICL extraction is done hierarchically, a merged PDL file may be merged into a higher level PDL file. You notice that the markers get extra # as they get included into a parent file as shown here:

  ```
  # Imported from ../corea_rtl.dft_inserted_design/corea.pdl
  ## Imported from ../../../my_instrument/counter.pdl

  The path "../../../my_instrument/counter.pdl" is relative to its
  original file location "../corea_rtl.dft_inserted_design".
  ```

- <design_name>.sdc

  A file that is created during ICL extraction or explicitly created using the extract_sdc command. The file combines the module-level timing information found in the *.sdc _dictionary* files with the instance-level information found in the ICL file. Refer to the "Timing Constraints SDC" in the *Tessent Shell User's Manual* for complete information.

- <design_name>.<gate_ext>

  A file that is typically created by the run_synthesis command after you have run the process_dft_specification command in the -no_rtl context. The run_synthesis command concatenates the <design_name>.<gate_ext>_no_instruments netlist with the newly-synthesized instruments found in the files at:

  ```
  instruments/<design_name>_<design_id>_xxx.<gate_ext>
  ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When using the write_design -tsdb command in the -no_rtl context, this file is directly created with the content of the current design found in memory. When using the write_design -tsdb -softlink_netlist command, the <design_name>.<gate_ext> entry is created as softlink to point to the location of the file containing the current design module in memory.

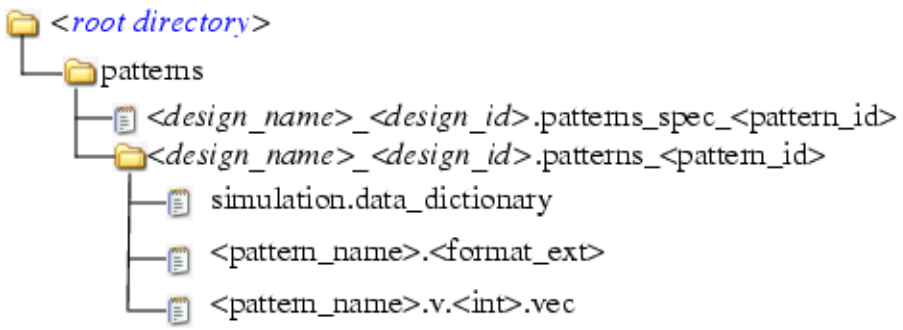
- <design_name>.dft_info_dictionary

  A file that is created during ICL extraction when using the extract_icl command with the -write_in_tsdb switch not set to off. The file is a Tcl dictionary containing the information about the DFT inserted in the design that must be considered during scan insertion when using a third-party tool. See the get_dft_info_dictionary command for more information.

patterns

A directory that stores the patterns and pattern specifications associated with each design, design id, and pattern id.

Summary



Description

A directory used to store the patterns and patterns specifications associated with each design, design id, and pattern id. The patterns specification file is always updated upon each successful validation of the specification using the process_patterns_specification where is the TSDB output directory as specified using the set_tsdb_output_directory command.

Components

- *<design_name>_<design_id>*.patterns_spec_<pattern_id>

  A file that contains the patterns specification that was just created using the create_patterns_specification, and successfully validated using the process_patterns_specification command. The <design_name>, <design_id>, and

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

<pattern_id> values correspond exactly to the matching elements in the PatternsSpecification(<design_name>, <design_id>, <pattern_id>) wrapper.

- *<design_name>_<design_id>*.patterns_<pattern_id>

  A directory that is created the first time a given PatternsSpecification(<design_name>, <design_id>, <pattern_id>) wrapper is processed using the process_patterns_specification command. The <design_name>, <design_id> and <pattern_id> values correspond exactly to the matching elements in the PatternsSpecification(<design_name>, <design_id>, <pattern_id>) wrapper.

- simulation.data_dictionary

  A file that contains a Tcl dictionary that is used by the run_testbench_simulations command to know the patterns list with their associated information as well as the design source dictionary explaining how to load the complete design downward. The format of the dictionary is illustrated below. Notice that the list of entries found in the Patterns entry is always the full list of patterns found in the PatternsSpecification wrapper independent of the -select or the -exclude option used in the last invocation of the process_patterns_specification command.

```
set simulation_dict {
  top_design_logical_library <library_name>
  patterns {
    <pattern_name> {
      testbench_module_name <module_name>
      top_design_instance_name <instance_name>
      extra_simulation_file_list <file_list>
      top_module_name_list <module_name_list>
      simulation_macro_list {<macro_definition_list>}
      lower_instances {
        <instance_name> {
          view full | interface
          logical_library <library_name>
        }
      }
      file_list {<testbench_and_sidefiles_list>}
      icl_time "<datetime>"
      sim_data_files <status>
    }
  }
  design_source_dictionary {
    read# {
    }
  }
}
```

- <pattern_name>.<format_ext>

  A file that contains the top-level testbench or the patterns file expressed in the requested format. The extension of the file is “v” for Verilog testbenches, or the exact format specified using the “manufacturing_patterns_formats” property in the PatternsSpecification wrapper. If you specify the “compress_pattern_files” option, also

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

found in the PatternsSpecification wrapper, to “on”, the patterns file is compressed and the *.gz* string is appended to the <format_ext> string.

Not all patterns found in the PatternsSpecification wrapper are updated or created when running the process_patterns_specification command with the -select or the -exclude options. Stale pattern files left over in the <design_name>_<design_id>.patterns_<pattern_id> directories from a previous run of the process_patterns_specification command is automatically deleted from the directory if they are no longer found in the current PatternsSpecification wrapper.

- <pattern_name>.v.<int>.vec

  A set of files that is only created for Verilog testbenches that contain the vector data for the associated pattern.

- <pattern_name>_<others>.v

  A set of files that is only created for Verilog testbenches that contain side modules to be compiled in parallel with the Verilog testbench. Such modules may include clock monitoring behavioral code. They may also contain BISR emulation code for sub-physical instances using the interface view. See the “emulate_bisr_chains_in_lower_physical_blocks” property inside the SimulationOptions wrapper for more information about those files.

logic_test_cores

A directory that stores the information generated when a test mode on a logic core was successfully design rule checked, and for which ATPG or LogicBIST patterns may have to be created.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Summary

Description

A directory used to store the information generated when a test mode on a logic core was successfully design rule checked, and for which ATPG or LogicBIST patterns may have been created.

Components

- *<design_name>*[_*<design_id>*].logic_test_core

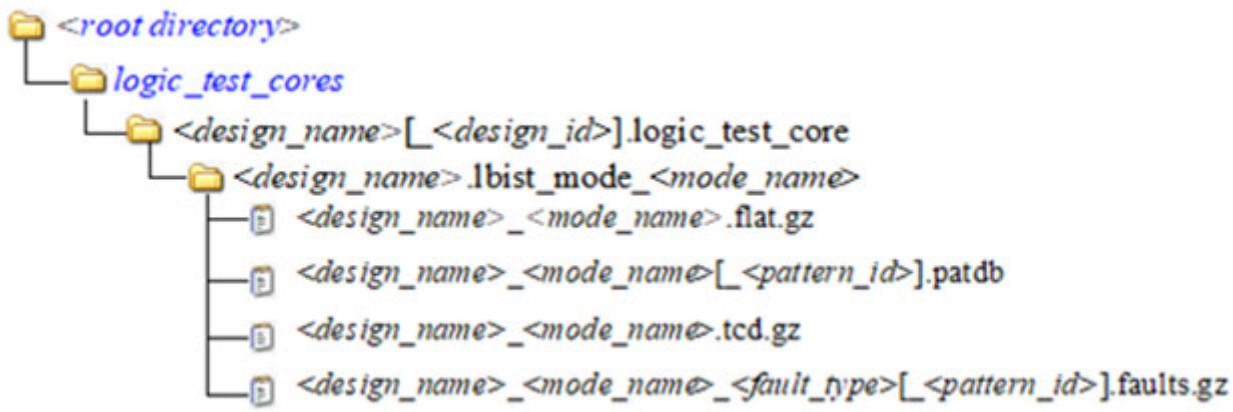
  A directory used to hold all logic_bist, atpg, and atpg_retargeting containers. The directory uses the design name as its name with an optional design_id suffix that you can use to differentiate various views of the same design such as the pre-layout and post-layout versions.

<design_name>.lbist_mode_<mode_name>

A directory used to store the information generated when a LogicBIST test mode on a logic core was successfully design rule checked and for which LogicBIST patterns were created.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Summary



Description

A directory used to store the information generated when a LogicBIST test mode on a logic core was successfully design rule checked and for which LogicBIST patterns were created.

Components

- *<design_name>*.lbist_mode_*<mode_name>*

  A directory used to store the information for a given LogicBIST test mode associated to a given design. The design is uniquely identified using its name and an optional design id as specified using the set_context command. The mode name is specified using the set_current_mode command.

  The directory is created clean each time the write_tsdb_data command is issued in analysis mode and patterns -scan context. The mode is identified as a LogicBIST mode when there was at least one add_lfsrs command specified in setup mode prior to going to analysis mode. You can use the "get_context -logic_bist" command to verify that it is a LogicBIST mode.

- *<design_name>*_*<mode_name>*.flat.gz

  A file containing the flat model representing the DRCed LogicBIST test mode. This file is used to fault simulate other fault models at a later time without having to redo DRC. Its main utility is, however, to perform diagnosis when the design is in production. See the *Tessent Diagnosis User's Manual* for further information about how to run diagnosis and how it uses the flat model.

- *<design_name>*_*<mode_name>*[_*<pattern_id>*].patdb

  A file containing the LogicBIST pattern information. It contains all PRPG seeds, all MISR signatures, and the flop expect data for a specified amount of patterns. The pattern

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ID is specified using the -pattern_id option when invoking the write_tsdb_data command.

- *<design_name>_<mode_name>*.tcd.gz

  A file containing a Core/LbistMode wrapper describing different aspect of the logictest test mode. The stored information is mainly used to map diagnostic data back to flip-flop names.

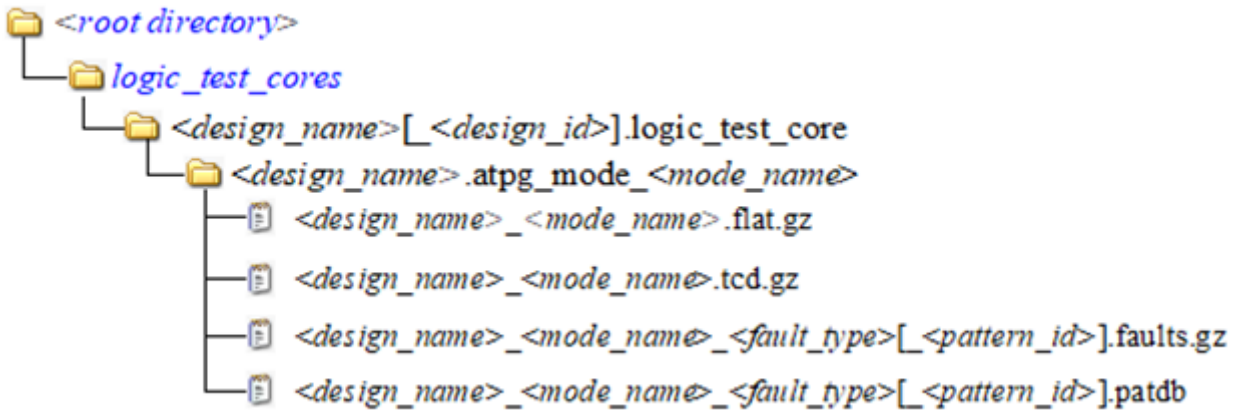
- *<design_name>_<mode_name>_<fault_type>*[_*<pattern_id>*].faults.gz

  A file containing the fault classification for a given fault type and a specific pattern id that is later used to create a consolidated fault coverage report per fault model. The pattern ID is specified using the -pattern_id option when invoking the write_tsdb_data command.

<design_name>.atpg_mode_<mode_name>

A directory used to store the information related to a logic core that has been configured for ATPG, passed design rule checks, and for which ATPG patterns were created.

Summary



Description

A directory used to store the information related to a logic core that has been configured for ATPG, passed design rule checks, and for which ATPG patterns were created. One typical usage is to store the ATPG results after generating retargetable patterns for a wrapped core. See also the write_tsdb_data command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Components

- *<design_name>*.atpg_mode_*<mode_name>*

  A directory used to store the information for a given ATPG test mode associated to a given design. The design is uniquely identified using its name and an optional design id as specified using the set_context command. The mode name is specified using the set_current_mode command.

- *<design_name>*_*<mode_name>*.flat.gz

  A file containing the flat model representing the DRCed ATPG test mode. The primary purpose of this file is for diagnosis. This file is also used to fault simulate other fault models at a later time without having to redo DRC.

- *<design_name>*_*<mode_name>*.tcd.gz

  A file containing a Core wrapper describing different aspect of the ATPG test mode. The TCD file for the ATPG mode is used for pattern retargeting. Similarly, for top-level ATPG, it can be used for the TCD flow. See "Scan Pattern Retargeting" and "Core Mapping for ATPG Process Overview" in the *Tessent Scan and ATPG User's Manual*.

- *<design_name>*_*<mode_name>*_*<fault_type>*[_*<pattern_id>*].faults.gz

  A file containing the fault classification for a given fault type and a specific pattern id that is later used to create a consolidated fault coverage report per fault model. The *pattern_id* is specified using the -pattern_id option when invoking the write_tsdb_data command.

- *<design_name>*_*<mode_name>*_*<fault_type>*[_*<pattern_id>*].patdb

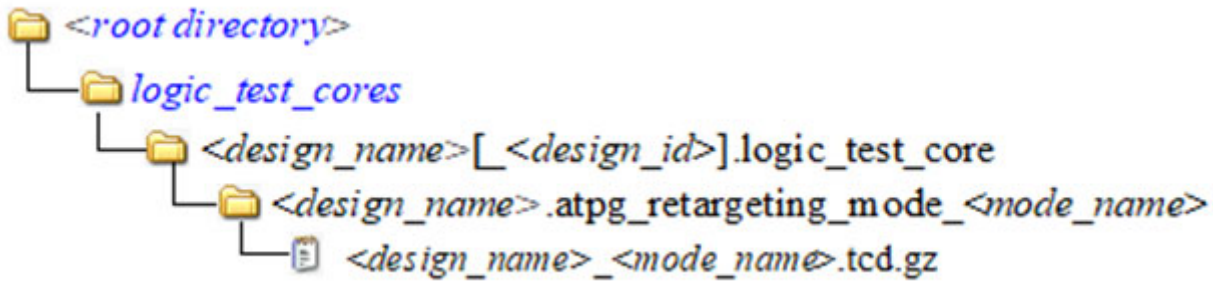
  A file containing the ATPG pattern information.

<design_name>.atpg_retargeting_mode_<mode_name>

A directory used to store the information related to a logic core that has been configured for retargeted ATPG, passed design rule checks, and for which retargeted ATPG patterns were created.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Summary



Description

A directory used to store the information related to a logic core that has been configured for retargeted ATPG, passed design rule checks, and for which retargeted ATPG patterns were created. See also the write_tsdb_data command and "Scan Pattern Retargeting" in the *Tessent Scan and ATPG User's Manual*.

Components

- *<design_name>*.atpg_retargeting_mode_*<mode_name>*

  A directory used to store the information for a given ATPG retargeting test mode associated to a given design. The design is uniquely identified using its name and an optional design id as specified using the set_context command. The mode name is specified using the set_current_mode command. The ATPG retargeting mode does not require that you have done pattern retargeting first. You need to have done at least done core extraction for pattern retargeting; see "Core-in-Core Pattern Retargeting" in the *Tessent Scan and ATPG User's Manual* for more information.

- *<design_name>*_*<mode_name>*.tcd.gz

  A file containing a Core wrapper describing different aspect of the ATPG retargeting test mode. The TCD for the retargeting mode can be used for the following: pattern retargeting in a subsequent step to core extraction without requiring extraction or even a design; and/or for reverse mapping of tester failures for core-level diagnosis.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 16
Instrument Connectivity Language (ICL)

This chapter specifies the syntax of the Instrument Connectivity Language (ICL) of the IEEE 1687-2014 (IJTAG) standard as implemented by Tessent IJTAG.

For more information about Tessent IJTAG, refer to the *Tessent IJTAG User's Manual*.

ICL Overview 4837
Unsupported ICL Language Elements 4837
Common Features of the ICL Language. 4838

ICL Statement Descriptions 4842
Syntax Conventions 4843
AccessLink 4844
AddressPort. 4853
Alias 4855
Attribute 4858
CaptureEnPort. 4860
Chain. 4862
ClockMux 4865
ClockPort 4868
DataInPort. 4870
DataInOutPort. 4874
DataMux 4876
DataOutPort 4879
DataRegister 4882
Enum. 4887
Instance 4890
IRSelectPort 4896
LocalParameter 4897
LogicSignal. 4898
Module 4900
NameSpace 4902
OneHotDataGroup 4903
OneHotScanGroup 4907
Parameter 4909
ReadEnPort. 4912
ResetPort. 4914
ScanInPort. 4916
ScanInterface 4918
ScanMux 4924
ScanOutPort 4927

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ScanRegister 4930
SelectPort 4935
ShiftEnPort 4937
TCKPort 4939
TMSPort 4941
ToCaptureEnPort 4944
ToClockPort 4948
ToIRSelectPort 4953
ToResetPort 4955
ToSelectPort 4958
ToShiftEnPort 4961
ToTCKPort 4965
ToTMSPort 4967
ToTRSTPort 4969
ToUpdateEnPort 4971
TRSTPort 4975
UpdateEnPort 4976
UseNameSpace 4978
WriteEnPort 4979

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL Overview

The syntax described in this chapter is not based on the formal grammar used in the IJTAG standard, but rather through a hierarchical description of an ICL Module and its elements.

Examples and suggestions for usage are provided for each language element. These elements are divided into language statements; that is, all the elements in the direct body of the ICL Module definition (plus a few more), and parameters. Parameters to an ICL language statement further define the properties of the statement. For example, the ICL language statement ScanRegister has the parameter ResetValue, which defines the value the scan register assumes if the reset signal turns active.

Unsupported ICL Language Elements 4837
Common Features of the ICL Language 4838
Numbers 4838
Strings 4839
Identifiers 4839
Limited Identifier Mathematics 4840
Concatenations 4840

Unsupported ICL Language Elements

Most of the language elements of the IEEE 1687 standard are supported. Currently unsupported ICL language elements or language elements with limited support are clearly marked.

For example:

Note

The DefaultLoadValue parameter is currently not supported within the ScanInterface.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Common Features of the ICL Language

The following sections introduce common features of the language.

Numbers 4838
Strings 4839
Identifiers 4839
Limited Identifier Mathematics 4840
Concatenations 4840

Numbers

ICL recognizes unsigned integer numbers in any of the following three number base formats: decimal, hexadecimal, and binary.

Unsized decimal numbers may be composed of a sequence of the digits 0 to 9. If needed, their size is determined based on context.

Sized numbers are composed of a size constant, followed by an optional white space and a base token (d or D for decimal, h or H for hexadecimal, and b or B for binary), and the digits of the number, expressed in the respective number base. The size constant always expresses the number of binary digits of the sized number, independent of the used base format. For example, the sized numbers 4'b1011, 4'hb and 4'd11 are identical and exactly 4 binary digits wide.

Binary, decimal, and hexadecimal numbers may use their respective common digit symbols (0 and 1 for binary, 0 to 9 for decimal, 0 to 9 and a to f for hexadecimal). These digit symbols are case insensitive. In addition, binary and hexadecimal numbers may use x or X to denote the unknown digit value.

Padding is applied automatically as necessary if the number of digits of the sized number is less than the size constant requires. For example, assume this comparison:

```
LogicSignal test { dataIn[7:0] == 8'b10 ; }
```

The “==” operator compares two 8-bit (that is, 8 binary digit wide sized numbers). However, only 2 digits “10” are given. Padding is executed based on the following rules: If the left most digit is non-x, then the padding adds as many 0 to the left as necessary. If however the leftmost digit is x, then padding adds as many x to the left as necessary. The above example is therefore padded to the following:

```
LogicSignal test { dataIn[7:0] == 8'b00000010 ; }
```

The opposite of padding is truncation. Truncation happens if the number of digits of the sized number is larger than the size constant requires. Consequently, excessive digits are removed

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

from the left until the size requirement is matched. However, only occurrences of 0 or x may be removed. Removing any other digit symbol results in an error.

Strings

Escaping in ICL strings is not compliant with the IEEE 1687-2014 standard. This has been done to maintain backward compatibility with existing ICL files generated by Tessent tools.

Backslashes in ICL strings are interpreted as escaping indicators only if the next character is a backslash or double quotation marks. If the next character is something else, the backslash is interpreted as an ordinary character of the string.

Identifiers

IJTAG differentiates between four identifiers: The scalar identifier, the parameter identifier, the vector identifier, and the pin identifier.

The scalar identifier is used to denote the elements of the ICL netlist such as the ports of a module, a module name or an instance name. The scalar identifier is case sensitive and must start with a letter. Thereafter, the scalar identifier may use letters, numbers, and the underscore symbol. It may not contain any white spaces nor the "[" or "]" symbols (these are used for vector identifiers).

The parameter identifier is used to reference a Parameter or LocalParameter definition. The parameter reference is composed of the "$" symbol and the scalar identifier name of the referenced (local) parameter. For example, if the parameter scalar identifier is userValue, the the parameter reference is $userValue.

There are two types of vector identifiers: The single index and the range vector identifier. There is no length limit for these identifiers.

A single index vector identifier is composed of a scalar identifier, followed by the "["symbol, an unsigned decimal number, followed by the "]" symbol. Its width is considered to be 1. An example is bus[4], denoting the 4th bit of the scalar identifier bus.

A ranged vector identifier is composed of a scalar identifier, followed by the "[" symbol, a first unsigned decimal number, the ":" symbol, a second unsigned decimal number, and followed by the "]" symbol. An example is:

```
bus[0:4]
```

denoting 5 bits of the scalar identifier bus. Note that the range can either be increasing, for example [0:4], or decreasing, for example [4:0].

Pin identifiers refer to a port name, optionally including an instance path name. The port name is either a scalar, a single index or range index vector identifier. The instance names are all

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

scalar identifiers, and the instance path is composed of these scalar identifiers, separated by the "." symbol. An example is:

```
top.myInstA.myInstB.dataIn[7:0]
```

In this example the identifiers "top", "myInstA" and "myInstB" are all denoting names of instances, whereas dataIn[7:0] is a ranged vector identifier of a port scalar named dataIn.

Limited Identifier Mathematics

Parameter identifiers, single indexes and ranged indexes may use limited mathematical operations: + (addition), - (subtraction), * (multiplication), / (division), and % (modulo). For parameter identifiers, a parameter value can be computed using these operations; for vector identifiers any used unsigned decimal number can be expressed using these operations. In addition, parameter reference may be used.

Examples are:

```
Parameter ten = 10 ;
Parameter twenty = 2*$ten ;
bus[$twenty:$ten]
bus[$twenty%2]
bus[$ten-1:0]
```

Concatenations

Under certain conditions, identifiers might be concatenated with other identifiers and/or sized and unsized numbers. The concatenation operator is the "," symbol.

A first example is the following:

```
Parameter topBit 1'b1 ;
LogicSignal test { dataInA[3:0], dataInB[3:0] == $topBit, 0, 4'b1111 ; }
```

Observe that the left hand side of the comparator (==) is 8 bits wide. The right hand side has 2 sized numbers of a total width of 5 bits. Therefore the unsized number 0 in the middle is auto-expanded into 3'b000. It is obvious that such a comparison construct may contain at most one unsigned number. This example shows also the usage of parameter references in lieu of a number.

A concatenation may also include the "~" symbol, denoting a sized, bit-wise, binary inversion. For example: ~4' ha is equivalent to 4'h5.

Interestingly, the "~"symbol can also be used within the concatenation:

```
LogicSignal test { dataInA[3:0], dataInB[3:0] == $topBit, ~0, 4'b1111 ; }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Here, the unsized, but inverted 0 is auto-expanded into 4'b1111.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ICL Statement Descriptions

The following describes each ICL statement available in Tessent IJTAG.

Syntax Conventions 4843
AccessLink 4844
AddressPort 4853
Alias 4855
Attribute 4858
CaptureEnPort 4860
Chain 4862
ClockMux 4865
ClockPort 4868
DataInPort 4870
DataInOutPort 4874
DataMux 4876
DataOutPort 4879
DataRegister 4882
Enum 4887
Instance 4890
IRSelectPort 4896
LocalParameter 4897
LogicSignal 4898
Module 4900
NameSpace 4902
OneHotDataGroup 4903
OneHotScanGroup 4907
Parameter 4909
ReadEnPort 4912
ResetPort 4914
ScanInPort 4916
ScanInterface 4918
ScanMux 4924
ScanOutPort 4927
ScanRegister 4930
SelectPort 4935

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ShiftEnPort 4937
TCKPort 4939
TMSPort 4941
ToCaptureEnPort 4944
ToClockPort 4948
ToIRSelectPort 4953
ToResetPort 4955
ToSelectPort 4958
ToShiftEnPort 4961
ToTCKPort 4965
ToTMSPort 4967
ToTRSTPort 4969
ToUpdateEnPort 4971
TRSTPort 4975
UpdateEnPort 4976
UseNameSpace 4978
WriteEnPort 4979

Syntax Conventions

This chapter uses these syntax conventions when documenting ICL syntax properties.

Table 16-1. SyntaxConventions for ICL Files

Convention	Example	Usage
Upright	ScanInPort *port_name*	An upright font indicates an ICL language element, either a statement or a parameter.
Italic	Attribute *att_name* = *att_value*;	An italic font indicates a user-supplied value.
Underline	ActivePolarity 0 | <u>1</u>;	An underlined item indicates the default value.
|	ActivePolarity 0 | <u>1</u>;	The vertical bar separates a list of values from which you must choose one. Do not include the bar in the configuration file.
...	ActiveSignals { *signal_name*; … }	Ellipses indicate a repeatable value.
//	// optional	The double slash indicates the text immediately following is a comment and tells the tool to ignore the text.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

AccessLink

Links an ICL network to an 1149.1 TAP controller that is described in a BSDL file.

Usage

```
Module module_name {
  AccessLink instance_name Of STD_1149_1_2001 {
    BSDLEntity entity_name;
    instruction_name {
      ScanInterface { scan_interface_name; ...} // required
      ActiveSignals { signal_name; ...} // optional
    }
  }
}
```

Description

A wrapper that can be used inside the top level ICL Module to link an ICL network to an 1149.1 TAP controller that is described in a BSDL file. The tool currently only supports the BSDL format that is part of IEEE1149.1-2001.

When reading an ICL file which contains an ICL module with an AccessLink Of STD_1149_1_2001, the read_icl command must be used with the -bsdl_file option to provide the path to the referenced BSDLEntity. An error is generated if the specified BSDLEntity is not found in the specified BSDL file.

Arguments

- *instance_name*

  A string used to define the instance name of the AccessLink. The string must start with a letter and only contain letters, numbers and underscores. As shown below, the instance_name is used when referring to the defined signal_name in an ICL connection. When the AccessLink is read in the tool, a TAP ICL module is created from the BSDL and instantiated in the top level ICL module in place of the AccessLink using the same instance_name. This is also illustrated below.

- BSDLEntity *entity_name*;

  A required keyword-string pair which defines the name of the BSDL entity that is referenced by the AccessLink wrapper. The file pointed to by the "read_icl -bsdl_file" option must contain an "Entity entity_name is" clause.

- **instruction_name**

  A required string that references one of the instruction names defined in the INSTRUCTION_OPCODE attribute of the BSDL file. When the specified instruction_name is loaded in the INSTRUCTION register, the scan_interface_names listed in the ScanInterface wrapper are placed along the active DR scan path between the TAP_SCAN_IN and TAP_SCAN_OUT ports.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- *scan_interface_name*

  A repeatable string that references the ScanInterface of ICL module instances. The format of the string is a leaf_instance_name followed by an optional ".scan_interface_name" string. The ".scan_interface_name" string is optional when the ICL module associated with the leaf_instance_name only has one ScanInterface. If the ICL module associated to the leaf_instance_name has more than one ScanInterface wrapper, the ".scan_interface_name" string is required to identify one of them.

  The scan_interface_name are serially daisy chained with the first one closer to TDI and the last one closer to TDO.

- *signal_name*

  A repeatable string that defines signal_names that are high when the instruction_name is loaded into the INSTRUCTION register. When a signal_name is not specified inside a given instruction_name wrapper, the signal is low when the instruction_name is loaded into the INSTRUCTION register. The signal_names are identified as "instance_name.signal_name" when referenced to make connections in the ICL. This is illustrated below.

Examples

Below is an example of a top level ICL module containing an AccessLink. It references a BSDL file which contains the element shown below. After the elaboration which is triggered by the set_current_design command, you can see that a TAP controller was extracted from the BSDL file and instantiated into the top level ICL module. The TAP ports were also extracted and added to the ICL module. The original AccessLink wrapper is commented out. The ClockPorts and their original connections are left unaffected.

Note

If an ICL module with an AccessLink statement has been loaded into memory by means of read_icl, then the commands write_icl and report_icl_modules do not result in usable ICL of this module before it has been subject to set_current_design.

Following is the content of mydesign.icl with an AccessLink wrapper. Note how the ActiveSignal long_setup is referenced as "bsdl_tap.long_setup" when used to connect the "selectWIR" input port of the "core" instance.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module mydesign {
  ClockPort CLK;
  AccessLink bsdl_tap Of STD_1149_1_2001 {
    BSDLEntity acme4523;
    BP0_LONG_SETUP {
      ScanInterface {core.ijtag0;}
      ActiveSignals {long_setup;}
    }
    BP0_SHORT_SETUP {
      ScanInterface {core.ijtag0;}
    }
    BP1_LONG_SETUP {
      ScanInterface {core.ijtag1;}
      ActiveSignals {long_setup;}
    }
    BP1_SHORT_SETUP {
      ScanInterface {core.ijtag1;}
    }
  }

  Instance core Of core {
    InputPort selectWIR = bsdl_tap.long_setup;
    InputPort clk = CLK;
  }

}
```

Following is the section from the BSDL file “acme4523.bsdl” that is used to extract the TAP controller.

```
entity acme4523 is
   attribute TAP_SCAN_RESET of TRST : signal is true;
   attribute TAP_SCAN_IN    of TDI  : signal is true;
   attribute TAP_SCAN_MODE  of TMS  : signal is true;
   attribute TAP_SCAN_OUT   of TDO  : signal is true;
   attribute TAP_SCAN_CLOCK of TCK  : signal is (1.0e+07, BOTH);

attribute INSTRUCTION_OPCODE of acme4523: entity is
      "IDCODE           (111111111111111110)," &
      "BYPASS           (111111111111111111)," &
      "EXTEST           (111111111111101000)," &
      "SAMPLE           (111111111111111000)," &
      "PRELOAD          (111111111111111000)," &
      "CLAMP            (111111111111101111) " &
      "BP0_LONG_SETUP   (XXXXXX000111XXX010)," &
      "BP1_LONG_SETUP   (XXXXXX001101XXX010)," &
      "BP2_LONG_SETUP   (XXXXXX001001XXX010)," &
      "BP3_LONG_SETUP   (XXXXXX000101XXX010)," &
      "BP4_LONG_SETUP   (XXXXXX000001XXX010)," &
      "BP0_SHORT_SETUP  (XXXXXX110110XXX010)," &
      "BP1_SHORT_SETUP  (XXXXXX111100XXX010)," &
      "BP2_SHORT_SETUP  (XXXXXX111000XXX010)," &
      "BP3_SHORT_SETUP  (XXXXXX110100XXX010)," &
      "BP4_SHORT_SETUP  (XXXXXX110000XXX010) " ;

end acme4523;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Below is a dofile that loads the ICL modules in memory and elaborates them. The modified top level ICL module as well as the created TAP ICL modules are then reported. Note how the TAP ICL module is called <bsdl_entity_name>_bsdl_tap. Its leaf instance name reused the one that was used by the AccessLink.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
> set_context patterns -ijtag
> read_icl mydesign.icl -bsdl_file acme4523.bsdl
> read_icl core.icl
> set_current_design
> report_icl_module -module mydesign
Module mydesign {
  ClockPort CLK;
  ScanInPort TDI;
  TMSPort TMS;
  TCKPort TCK;
  TRSTPort TRST;
  ScanOutPort TDO {
    Source bsdl_tap.tdo;
  }
/*
  AccessLink bsdl_tap Of STD_1149_1_2001 {
    BSDLEntity acme4523;
    BP0_LONG_SETUP {
      ScanInterface {core.ijtag0;}
      ActiveSignals {long_setup;}
    }
    BP0_SHORT_SETUP {
      ScanInterface {core.ijtag0;}
    }
    BP1_LONG_SETUP {
      ScanInterface {core.ijtag1;}
      ActiveSignals {long_setup;}
    }
    BP1_SHORT_SETUP {
      ScanInterface {core.ijtag1;}
    }
  }
  */

  ScanMux tessent_access_link_mux0 SelectedBy
                bsdl_tap.BP0_LONG_SETUP_select,
                bsdl_tap.BP0_SHORT_SETUP_select,
                bsdl_tap.BP1_LONG_SETUP_select,
                bsdl_tap.BP1_SHORT_SETUP_select {
    4'b1xxx | 4'bx1xx | 4'bxx1x | 4'bxxx1 : TDI;
   }
  Instance core Of core {
    InputPort selectWIR = bsdl_tap.long_setup;
    InputPort clk = CLK;
    InputPort si = tessent_access_link_mux0;
  }
  Instance bsdl_tap Of acme4523_bsdl_tap {
    InputPort tdi = TDI;
    InputPort tms = TMS;
    InputPort tck = TCK;
    InputPort trst = TRST;
    InputPort BP0_LONG_SETUP_from_scan_out = core.fso0;
    InputPort BP0_SHORT_SETUP_from_scan_out = core.fso0;
    InputPort BP1_LONG_SETUP_from_scan_out = core.fso1;
    InputPort BP1_SHORT_SETUP_from_scan_out = core.fso1;
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
> report_icl_module -module acme4523_bsdl_tap
Module acme4523_bsdl_tap {
  TCKPort tck;
  TMSPort tms;
  TRSTPort trst;
  ScanInPort tdi;
  ScanOutPort tdo {
    Source IRMux;
  }
  ToResetPort testLogicReset {
    ActivePolarity 1;
    Source fsm.tlr;
  }
  ToResetPort testLogicResetInv {
    ActivePolarity 0;
    Source ~fsm.tlr;
  }
  ToCaptureEnPort BP0_LONG_SETUP_capture_en;
  ToShiftEnPort BP0_LONG_SETUP_shift_en;
  ToUpdateEnPort BP0_LONG_SETUP_update_en;
  ToSelectPort BP0_LONG_SETUP_select {
    Source BP0_LONG_SETUP_logic_signal;
  }
  ToResetPort BP0_LONG_SETUP_reset_inv {
    ActivePolarity 0;
  }
  ScanInPort BP0_LONG_SETUP_from_scan_out;
  ScanOutPort BP0_LONG_SETUP_to_scan_in {
    Source tdi;
  }
  ToCaptureEnPort BP0_SHORT_SETUP_capture_en;
  ToShiftEnPort BP0_SHORT_SETUP_shift_en;
  ToUpdateEnPort BP0_SHORT_SETUP_update_en;
  ToSelectPort BP0_SHORT_SETUP_select {
    Source BP0_SHORT_SETUP_logic_signal;
  }
  ToResetPort BP0_SHORT_SETUP_reset_inv {
    ActivePolarity 0;
  }
  ScanInPort BP0_SHORT_SETUP_from_scan_out;
  ScanOutPort BP0_SHORT_SETUP_to_scan_in {
    Source tdi;
  }
  ToCaptureEnPort BP1_LONG_SETUP_capture_en;
  ToShiftEnPort BP1_LONG_SETUP_shift_en;
  ToUpdateEnPort BP1_LONG_SETUP_update_en;
  ToSelectPort BP1_LONG_SETUP_select {
    Source BP1_LONG_SETUP_logic_signal;
  }
  ToResetPort BP1_LONG_SETUP_reset_inv {
    ActivePolarity 0;
  }
  ScanInPort BP1_LONG_SETUP_from_scan_out;
  ScanOutPort BP1_LONG_SETUP_to_scan_in {
    Source tdi;
  }
  ToCaptureEnPort BP1_SHORT_SETUP_capture_en;
  ToShiftEnPort BP1_SHORT_SETUP_shift_en;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
ToUpdateEnPort BP1_SHORT_SETUP_update_en;
ToSelectPort BP1_SHORT_SETUP_select {
  Source BP1_SHORT_SETUP_logic_signal;
}
ToResetPort BP1_SHORT_SETUP_reset_inv {
  ActivePolarity 0;
}
ScanInPort BP1_SHORT_SETUP_from_scan_out;
ScanOutPort BP1_SHORT_SETUP_to_scan_in {
  Source tdi;
}
DataOutPort long_setup {
  Source long_setup_LogicSignal;
}
ScanInterface Client {
  Port tdi;
  Port tdo;
  Port trst;
  Port tms;
  Port tck;
}
ScanInterface BP0_LONG_SETUP {
  Port BP0_LONG_SETUP_capture_en;
  Port BP0_LONG_SETUP_shift_en;
  Port BP0_LONG_SETUP_update_en;
  Port BP0_LONG_SETUP_select;
  Port BP0_LONG_SETUP_from_scan_out;
  Port BP0_LONG_SETUP_to_scan_in;
  Port BP0_LONG_SETUP_reset_inv;
}
ScanInterface BP0_SHORT_SETUP {
  Port BP0_SHORT_SETUP_capture_en;
  Port BP0_SHORT_SETUP_shift_en;
  Port BP0_SHORT_SETUP_update_en;
  Port BP0_SHORT_SETUP_select;
  Port BP0_SHORT_SETUP_from_scan_out;
  Port BP0_SHORT_SETUP_to_scan_in;
  Port BP0_SHORT_SETUP_reset_inv;
}
ScanInterface BP1_LONG_SETUP {
  Port BP1_LONG_SETUP_capture_en;
  Port BP1_LONG_SETUP_shift_en;
  Port BP1_LONG_SETUP_update_en;
  Port BP1_LONG_SETUP_select;
  Port BP1_LONG_SETUP_from_scan_out;
  Port BP1_LONG_SETUP_to_scan_in;
  Port BP1_LONG_SETUP_reset_inv;
}
ScanInterface BP1_SHORT_SETUP {
  Port BP1_SHORT_SETUP_capture_en;
  Port BP1_SHORT_SETUP_shift_en;
  Port BP1_SHORT_SETUP_update_en;
  Port BP1_SHORT_SETUP_select;
  Port BP1_SHORT_SETUP_from_scan_out;
  Port BP1_SHORT_SETUP_to_scan_in;
  Port BP1_SHORT_SETUP_reset_inv;
}
Attribute tessent_instruction_reg = "INSTRUCTION";
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Attributetessent_bypass_reg="BYPASS";
Attributetessent_boundary_scan_reg="BOUNDARY";
Attributetessent_device_id_reg="DEVICE_ID";
Attributetessent_instrument_type="mentor::jtag_bscan";
Attributetessent_instrument_subtype="tap";
Attributetessent_signature="a693927b83d713ce7521168994205e56";
Enum INSTRUCTION_OPCODES{
  BP2_SHORT_SETUP=18'bxxxxxx111000xxx010;
  BP3_SHORT_SETUP=18'bxxxxxx110100xxx010;
  BP0_SHORT_SETUP=18'bxxxxxx110110xxx010;
  BP1_SHORT_SETUP=18'bxxxxxx111100xxx010;
  BP4_SHORT_SETUP=18'bxxxxxx110000xxx010;
  PRELOAD=18'b111111111111111000;
  BYPASS=18'b111111111111111111;
  IDCODE=18'b111111111111111110;
  SAMPLE=18'b111111111111111000;
  CLAMP=18'b111111111111101111;
  EXTEST=18'b111111111111101000;
  BP4_LONG_SETUP=18'bxxxxxx000001xxx010;
  BP2_LONG_SETUP=18'bxxxxxx001001xxx010;
  BP3_LONG_SETUP=18'bxxxxxx000101xxx010;
  BP0_LONG_SETUP=18'bxxxxxx000111xxx010;
  BP1_LONG_SETUP=18'bxxxxxx001101xxx010;
}
ScanRegister INSTRUCTION[17:0] {
  ScanInSource tdi;
  CaptureSource 18'bxxxxxxxxxxxxxxxx01;
  ResetValue 18'b111111111111111110;
  RefEnum INSTRUCTION_OPCODES;
}
ScanRegister BOUNDARY[41:0] {
  ScanInSource tdi;
}
ScanRegister BYPASS {
  ScanInSource tdi;
  CaptureSource 1'b0;
}
ScanRegister DEVICE_ID[31:0] {
  ScanInSource tdi;
  CaptureSource 32'b00010001001000110100001001000111;
}
ScanMux IRMux SelectedBy fsm.irSel {
  1'b1 : INSTRUCTION[0];
  1'b0 : DRMux;
}
ScanMux DRMux SelectedBy INSTRUCTION[17:0] {
  18'b111111111111111000 : BOUNDARY[0];
  18'b111111111111101000 : BOUNDARY[0];
  18'b111111111111001111 : BYPASS;
  18'b111111111111111111 : BYPASS;
  18'b111111111111101111 : BYPASS;
  18'b111111111111111110 : DEVICE_ID[0];
  18'bxxxxxx110110xxx010 : BP0_SHORT_SETUP_from_scan_out;
  18'bxxxxxx111100xxx010 : BP1_SHORT_SETUP_from_scan_out;
  18'bxxxxxx000111xxx010 : BP0_LONG_SETUP_from_scan_out;
  18'bxxxxxx001101xxx010 : BP1_LONG_SETUP_from_scan_out;
}
LogicSignal long_setup_LogicSignal {
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
      (INSTRUCTION == BP0_LONG_SETUP) || (INSTRUCTION == BP1_LONG_SETUP);
    }
    LogicSignal BP0_LONG_SETUP_logic_signal {
      INSTRUCTION == BP0_LONG_SETUP;
    }
    LogicSignal BP0_SHORT_SETUP_logic_signal {
      INSTRUCTION == BP0_SHORT_SETUP;
    }
    LogicSignal BP1_LONG_SETUP_logic_signal {
      INSTRUCTION == BP1_LONG_SETUP;
    }
    LogicSignal BP1_SHORT_SETUP_logic_signal {
      INSTRUCTION == BP1_SHORT_SETUP;
    }
    Instance fsm Of TOP_bsdl_tap_fsm {
      InputPort tck = tck;
      InputPort tms = tms;
      InputPort trst = trst;
    }
  }
```

Related Topics

read_icl

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

AddressPort

Declares the name of an address input port of a Module.

Usage

```
Module module_name {
  AddressPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of an address input port of a Module. It is used if the module contains addressable instances or data registers.

This port serves as the address of the addressable instances and data registers.

Arguments

- *port_name*

  A required string that identifies the name of an address port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL AddressPort of the module that is also used for the respective ports of the module in the design.

  The source of the address port name signal may only be an address port of the parent module, or scan register bits of a ScanRegister statement. In both cases, this connection must be direct, and may not include any logic signals.

  If a module has more than one AddressPort, the address comparison with logic signals of the module implicitly concatenates all address port signals in order of declaration (top to bottom).

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

WriteEnPort

Instance

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ReadEnPort

DataRegister

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Alias

Collects signals from different origins to form a named, new, virtual signal.

Usage

Syntax 1

```
Module module_name {
  Alias alias_name = concat_signal ;
}
```

Syntax 2

```
Module module_name {
  Alias alias_name = concat_signal {
    RefEnum enum_name ;
    AccessTogether ;
    iApplyEndState value ;
  }
}
```

Description

The Alias statement collects signals from different origins to form a named, new, virtual signal.

The name of the alias can be used in any operation or construct where the original signals would have been allowed. For example, PDL commands like iWrite or iRead can use the alias name. Choosing meaningful alias names can make both ICL and PDL, much more human readable.

A second usage of the Alias statement is to instruct Tessent to automatically return to a specified value for the selected signals. Typically this feature is used to return a write-enable signal to the inactive value assignment automatically, once the write operation has been completed.

If none of the parameters RefEnum, AccessTogether, or iApplyEndState are used, you must use Syntax 1.

The name of the Alias must be unique among all port names, and among all the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal.

Arguments

- alias_name = *concat_signal*

  A required string-string pair, connected with the '=' symbol. The *alias_name* defines the name of the alias, whereas *concat_signal* refers to one signal, or a concatenation of signals. Such a concatenation of signals is created by a signal name followed by a comma (','), followed by another signal name and so on. The concatenated signals can only be from DataInPort, ScanInPort, DataOutPort, or ScanOutPort. The width of the alias name should match the width of the concatenated signal.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

An alias name can become part of another Alias statement, however circular definitions are not allowed. Further, if any of the signals in the concatenation has more than one level of hierarchy; that is, the hierarchical instance-port path name contains the dot (“.”), the Alias may not be used in any other ICL statements, but only in PDL.

- RefEnum *enum_name*

  An optional keyword-string pair that references the name of a Related Topics table. Using the enumeration table, the writing to the aliased signal can use synonyms listed in the table, instead of bit strings. Understand that the enumeration table is not necessarily a complete list. You are still allowed to read or write any value, including those not listed in the enumeration table. An Alias statement may have at most one RefEnum parameter.

- AccessTogether

  An optional keyword that requests Tessent to operate all signals listed in *concat_signal* at the same time. If this turns out to be not possible, for example, based on mutually exclusive hardware requirements, an error is returned. Without this keyword, Tessent is allowed to find a solution where some of the signals are operated first, followed by operating another set of signals in the *concat_signal* Alias.

  Note

  Currently Tessent does not support the AccessTogether parameter.

- iApplyEndState *value*

  An optional keyword-string pair that identifies the value to which the signals listed *concat_signal* returns to automatically, if ever changed by an iWrite command. The value returns with the next iApply PDL command. Typically this feature is used to return the write-enable signal to the inactive value assignment automatically, once the write operation has been completed.

  There are additional conditions to the signals and value used in the Alias and iApplyEndState statement and parameter: All resolved signals must be legal targets of the iWrite command; *value* may not contain unknown values; if a signal is part of multiple iApplyEndState parameters, all specified values must be the same; if the signals resolve to register bits, either ScanRegister or DataRegister, the register must have a ResetValue parameter defined and the reset value should match the iApplyEndState-value for these signals; if a signal does not resolve to register bits, it must resolve to primary inputs, no other resolved signal source is allowed for the *concat_signal*, if the iApplyEndState parameter is used; further, there may not be a DataOutPort with an Enable parameter between the resolved signal source and the signal specified in *concat_signal* of the Alias statement.

Examples

The following example shows several usages of the Alias statement.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module instrument {
   DataInPort  in[8:0];
   DataOutPort out[7:0];
   ClockPort   clk;

   Alias write_en    = in[8]{ RefEnum YesNo; iApplyEndState 1'b0; }
   Alias enable      = in[7]            { RefEnum YesNo;    }
   Alias mode[3:0]   = in[6:5],in[3:2] { RefEnum Modes;    }
   Alias p1[2:0]     = in[4],in[1:0];
   Alias go          = out[0]           { RefEnum PassFail; }
   Alias done        = out[1]           { RefEnum YesNo;    }
   Alias count[5:0]  = out[7:2];
   Enum PassFail {
      Pass = 1'b1;
      Fail = 1'b0;
   }
   Enum YesNo {
      Yes  = 1'b1;
      No   = 1'b0;
   }
   Enum Modes {
      none      = 4'b0 ;
      red       = 4'b0011 ;
      blue      = 4'haa ;
      green     = 4'd9 ;
   }
}
```

Related Topics

iRead

LogicSignal

ScanRegister

iWrite

DataInPort

DataOutPort

DataRegister

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Attribute

Specifies a user attribute name and value.

Usage

```
Module module_name {
  Attribute att_name = att_value ;
}
```

Description

The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool.

Tessent understands several port attributes and pin attributes as defined in this manual.

Note that most Tessent attributes are read-only and are automatically generated by the Tessent tool, which also assigns a value to the attribute.

The Attribute statement is not only allowed in Module, but in most ICL objects documented in this chapter.

Note

In its usage within ICL objects, like modules, ports, scan or data mux, and so on, the Attribute statement is always optional.

Arguments

- *att_name* = *att_value*

  An required string and optional "="symbol and value combination, where *att_name* defines the name of the Attribute, and *att_value*, if present, defines the value of the attribute.

  Within an ICL object where multiple Attribute statements are used, each *att_name* must be unique.

Examples

Example 1

The first example shows a very simple usage of assigning a string "true" to an attribute name "is_top":

```
Module top {
  Attribute is_top = "true" ;
  [...]
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The second example shows a Tessent attribute connection_rule_option that influences ICL Network Extraction. Please see "Performing ICL Extraction" in the *Tessent IJTAG User's Manual* for the details of the attribute.

```
DataInPort din {Attribute connection_rule_option = "allowed_tied_high";}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

CaptureEnPort

Declares the name of the capture enable port of a Module's client ScanInterface.

Usage

```
Module module_name {
  CaptureEnPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the capture enable port of a Module's client ScanInterface.

This port is used to control the capture operation of a module's ScanRegister elements. A module may have zero or more capture enable ports. For an internal module; that is, not a hand-off module, a single capture enable port is implied if there are scan in/out ports in the module. In this case, a CaptureEnPort need not be declared. The CaputureEnPort must be declared if:

- The module is a hand-off module.
- There is more than one capture enable port in the ICL module definition. This may happen if there are multiple scan in/out ports defined that are operated independently.

The capture enable signal is an active high signal. A CaptureEnPort of an instance is sourced by a CaptureEnPort of the parent module, a ToCaptureEnPort of another instance or a DataMux which has a capture signal at one of its inputs and the constant value 1'b0 at the other input. If the DataMux is configured such that it selects the constant value 1'b0, the capture enable pulse is assumed to be suppressed.

Note

The suppressing of the enable signal described above is currently not supported in Tessent.

Arguments

- *port_name*

  A required string that identifies the name of a capture enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL CaptureEnPort of the module that is also used for the respective ports of the module in the design.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following example shows an ICL module definition with a single scan-in / scan-out chain, therefore, no client ScanInterface is required. The capture enable port is named “ce”. Observe that the body of the Module does not further use the capture enable port name, nor is its active-high value used or stated anywhere. Its connection to the scan chain is done implicitly. Please see the examples of the ScanInterface to learn how you can use multiple scan chains, potentially with multiple capture enable signals.

```
Module tdr2 {
   ScanInPort       si;
   ScanOutPort      so  { Source R[0]; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;
   ScanRegister R[7:0] {
      ScanInSource     si;
      ResetValue       8'b0 ;
      DefaultLoadValue 8'b0 ;
   }
}
```

Related Topics

ShiftEnPort

UpdateEnPort

ScanInPort

ScanOutPort

ScanInterface

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chain

Defines the name of a scan chain within a ScanInterface.

Usage

```
ScanInterface interface_name {
  Chain chain_name {
    Attribute att_name = att_value ;
    Port port_name ;
    DefaultLoadValue load_value ;
  }
}
```

Description

The Chain statement defines the name of a scan chain within a ScanInterface.

This statement does not create a scan register in the sense of the ScanRegister statement. Here it is only a declaration which ScanInPort and ScanOutPort of the module is operated together with the control signals of the ScanInterface. There can be more than one Chain statement in a ScanInterface. In this case, each scan chain name must be unique.

Arguments

- Chain *chain_name* ;

  A repeatable, required keyword-string pair that defines the name of a scan chain within a given ScanInterface. There can be more than one scan chain in any given ScanInterface. However, the names of the scan chains within a ScanInterface must all be unique.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent does not have any built-in attributes for the ScanInterface/Chain statement.

- Port *port_name* ;

  A required keyword-string pair, with the port_name referencing a ScanInPort or ScanOutPort port of the module. Within each Chain statement, there are exactly two Port statements, one referencing the scan-in port, and the other referencing the scan-out port.

- DefaultLoadValue *load_value* ;

  An optional keyword-string pair that defines the value to be loaded into the single scan chain defined by a pair port names of portfunction type ScanInPort and ScanOutPort of a black-box module. The DefaultLoadValue property affects the behavior of the iScan command only. The iScan command is currently not supported.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Example 1

The following example shows a module with two client ScanInterfaces, each operating two scan chains.

```
Module chip1 {
   ScanInPort  siA1 ;
   ScanInPort  siA2 ;
   ScanOutPort soA1 { Source blockA.so1 ; }
   ScanOutPort soA2 { Source blockA.so2 ; }
   ScanInPort  siB1 ;
   ScanInPort  siB2 ;
   ScanOutPort soB1 { Source blockB.so1 ; }
   ScanOutPort soB2 { Source blockB.so2 ; }
   ShiftEnPort   se1 ;
   CaptureEnPort ce1 ;
   UpdateEnPort  ue1 ;   ShiftEnPort   se2 ;
   CaptureEnPort ce2 ;
   UpdateEnPort  ue2 ;   TCKPort tck ;
   ScanInterface cA12 {
      Port se1; Port ce1; Port ue1;
      Chain c1 { Port siA1; Port soA1 ; }
      Chain c2 { Port siA2; Port soA2 ; }
   }         ScanInterface cB12 {
      Port se2; Port ce2; Port ue2;
      Chain c1 { Port siB1; Port soB1 ; }
      Chain c2 { Port siB2; Port soB2 ; }
   }
   [...]
}
```

Example 2

In this variation of Example 1 above, the ShiftEnPort, CaptureEnPort, and UpdateEnPort ports are shared among both ScanInterface. The selection between the two interfaces is accomplished through the two SelectPort sel1 and sel2, respectively.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module chip2 {
   ScanInPort  siA1 ;
   ScanInPort  siA2 ;
   ScanOutPort soA1 { Source blockA.so1 ; }
   ScanOutPort soA2 { Source blockA.so2 ; }
   ScanInPort  siB1 ;
   ScanInPort  siB2 ;
   ScanOutPort soB1 { Source blockB.so1 ; }
   ScanOutPort soB2 { Source blockB.so2 ; }
   SelectPort   selA ;
   SelectPort   selB ;   ShiftEnPort   se ;
   CaptureEnPort ce ;
   UpdateEnPort  ue ;   TCKPort tck ;   ScanInterface cA12 {
      Port selA;
      Chain c1 { Port siA1; Port soA1 ; }
      Chain c2 { Port siA2; Port soA2 ; }
   }         ScanInterface cB12 {
      Port selB;
      Chain c1 { Port siB1; Port soB1 ; }
      Chain c2 { Port siB2; Port soB2 ; }
   }
   [...]
}
```

Example 3

This example is another variation of Example 1. The difference is that all ports are busses and not individual ports as in Example 1. The operation of the modules is identical.

```
Module chip3 {
   ScanInPort  si[3:0] ;
   ScanOutPort so[3:0] { Source blockA.so1, blockA.so2,
                                 blockB.so1, blockB.so2 ;
                       }   ShiftEnPort   se[1:0] ;
   CaptureEnPort ce[1:0] ;
   UpdateEnPort  ue[1:0] ;   TCKPort tck ;   ScanInterface cA12 {
      Port se[0]; Port ce[0]; Port ue[0];
      Chain c1 { Port si[0]; Port so[0] ; }
      Chain c2 { Port si[1]; Port so[1] ; }
   }         ScanInterface cB12 {
      Port se[1] ; Port ce[1]; Port ue[1];
      Chain c1 { Port si[2]; Port so[2] ; }
      Chain c2 { Port si[3]; Port so[3] ; }
   }
   [...]
}
```

Related Topics

ScanInterface

ScanRegister

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ClockMux

Creates and names a clock multiplexer.

Usage

```
Module module_name {
  ClockMux mux_name SelectedBy mux_select_signal {
      Attribute att_name = att_value;
    mux_select_value : clock_source_name ;
  }
}
```

Description

The ClockMux statement creates and names a clock multiplexer.

IJTAG differentiates between a scan multiplexer, a data multiplexer, and a clock multiplexer. All three share the same syntax (other than the keyword), but their semantics differ. The scan multiplexer only switches between scan signals, the data multiplexer only switches between data signals, and the clock multiplexer only switches between clock signals. No signal types may be mixed for any of the multiplexers. The select signal of the multiplexers may be sourced only by data signals.

Arguments

- *mux_name*

  A string that identifies the name of a data multiplexer of the module. The name of the ScanMux must be unique among all port names, and among all the following objects: ScanRegister, DataRegister, ScanMux, DataMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  Using this name, the outgoing signal of the multiplexer can be referenced. The width of *mux_name* must match the with of the *clock_source_name*.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The possibility to attach attributes to multiplexers is not part of the IEEE 1687-2014 standard. It is an extension specific to Tessent Shell. The attribute name and the attribute value is defined by you and may be ignored or rejected by any IJTAG software tool.

- *mux_select_value* : *clock_source_name*

  A repeatable string-string pair, connected by a colon symbol ':'. A DataMux must have at least one mux_select_value - clock_source_name pair. The width of the *mux_select_value* must match the width of the *mux_select_signal*.

  The *mux_select_value* describes the value the select signal (*mux_select_signal*) of the mux must assume for the *mux_name* signal to connect to the *clock_source_name* signal. The *mux_select_value* may contain x-values. The collection of all specified *mux_select_value*

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

parameters need not cover the entire number space. Any select value not listed causes the *mux_name* signal to assume x-values of the correct width.

A mux_select_value may consist of multiple different select values connected by OR ("|"). The width of each individual select value must match the width of the mux_select_signal.

Examples

The following shows usages of all the multiplexer types.

```
Module chip2 {
   ScanInPort  siA ;
   ScanInPort  siB ;
   ScanOutPort so      { Source scanM ; }
   ShiftEnPort   se1 ;
   CaptureEnPort ce1 ;
   UpdateEnPort  ue1 ;
   ShiftEnPort   se2 ;
   CaptureEnPort ce2 ;
   UpdateEnPort  ue2 ;
   TCKPort tck ;
   ClockPort clkA ;
   ClockPort clkB ;
   ToClockPort clk { Source clockM ; }
   DataInPort   dselA ;
   DataInPort   dselB[2:0] ;
   DataOutPort do[3:0] { Source dataM[3:0] ; }
   ScanInterface cA12 {
      Port se1; Port ce1; Port ue1;
      Chain c1 { Port siA; Port so ; }
   }
   ScanInterface cB12 {
      Port se2; Port ce2; Port ue2;
      Chain c1 { Port siB; Port so ; }
   }
   ScanRegister regA[7:0] {
      ScanInSource siA ;
      ResetValue   8'b0 ;
   }
   ScanRegister regB[7:0] {
      ScanInSource siB ;
      ResetValue   8'b0 ;
   }
   ScanMux scanM SelectedBy dselA {
      1'b0 : regA[0] ;
      1'b1 : regB[0] ;
   }
   ClockMux clockM SelectedBy dselA {
      1'b0 : clkA ;
      1'b1 : clkB ;
   }
   DataMux dataM[3:0] SelectedBy dselB[2:0] {
      3'b000|3'b100  : regA[7:4] ;
      3'bx01         : regA[3:0] ;
      3'bx1x         : regB[5:2] ;
   }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

ScanMux

DataMux

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ClockPort

Declares the name of a functional clock input port of a Module.

Usage

```
Module module_name {
  ClockPort port_name {
    Attribute att_name = att_value ;
    DifferentialInvOf other_clock_port_name ;
  }
}
```

Description

A port function that declares the name of a functional clock input port of a Module (that is, a clock that is not a test clock).

A test clock is defined through the TCKPort statement. A module may have zero or more clock ports.

Note that IJTAG has no means of defining the actual speed of a functional clock. This is left to the software implementing the standard. In Tessent, you can use the add_clocks command to define clock properties, like the period, or if the clock is synchronous or asynchronous to the test clock. Inside of ICL, only the ToClockPort statement can be used to model an oscillator-like behavior.

Arguments

- *port_name*

  A required string that identifies the name of a clock port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ClockPort of the module that is also used for the respective ports of the module in the design.

- DifferentialInvOf *other_clock_port_name* ;

  An optional keyword-string pair. Differential functional clocks are declared through the DifferentialInvOf keyword. The *other_clock_port_name* string references another ClockPort *port_name* of the same module.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example declares one test clock and two functional, non-test clocks, an ordinary one and a differential one, the latter of which consists of two clock pins “pdiff” and “ndiff”.

```
Module top {
   TCKPort tck;
   ClockPort sclk;
   ClockPort pdiff;
   ClockPort ndiff { DifferentialInvOf pdiff ; }
}
```

Related Topics

TCKPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DataInPort

Declares the name of the data in port of a Module.

Usage

```
Module module_name {
  DataInPort port_name {
    RefEnum enum_name ;
    DefaultLoadValue default_load_value ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the data in port of a Module.

A module may have zero or more data in ports. The number and bus-width of all declared data in ports is not correlated to the number and bus-width of any DataOutPort.

A data in port is used to execute a parallel write (that is, not a scan operation), to ports of the module. For a scan operation, use the ScanInPort port function instead. If you allow parallel writes to ports of the module, you must declare the DataInPort function or functions for the module. If a module does not have DataInPort port functions, the tool will not allow parallel data write. Certain additional restrictions apply as explained next.

There should be at most one RefEnum per DataInPort, and the widths of the data within the referenced enumeration table should match the width of the data in port.

If a data in port is used to select a scan path, for example, as the select of a ScanMux, it may not be the target of a write operation. This ensures that the scan selection remains under control of the software application.

If you connect a data in port to an Instance or DataRegister with an AddressValue statement, the tool will concatenate all data in ports in the order of declaration until it achieves the required data width. See the discussion of AddressValue for further explanation and an example.

Arguments

- *port_name*

  A required string that identifies the name of a data in port of the module. The port names for all ports of a module must be unique. In addition, you may not use the port name for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL DataInPort of the module that is also used for the respective ports of the module in the design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- RefEnum *enum_name* ;

  An optional keyword-string pair that references the name of an Enum table. Using the enumeration table, the writing to the data in port can use synonyms listed in the table, instead of bit strings. Understand that the enumeration table is not necessarily a complete list. You are still allowed to write any value to the data in ports, including those not listed in the enumeration table. A DataInPort may have at most one RefEnum statement.

- DefaultLoadValue *default_load_value* ;

  An optional keyword-string pair that identifies the value that each DataIn port assumes if the tool had not written any user-defined value to it beforehand and needs to write to other bits of the Ports. If you use default_load_value, you must match its the width to the width of the port.

  For DataInPorts on the current design, the tool automatically forces those DataInPorts to the specified *default_load_value* at the beginning of the pattern and after an iReset operation. If you do not specify a DefaultLoadValue, the tool forces the DataInPort to “0”. There is one exception to this behavior. If you declare the DataInPort as an inout port in the design module having the same name as the top level ICL module, the tool drives the port to Z at the beginning of the pattern. The port remains Z until it is the target of an explicit iWrite command or its value is needed to justify other iRead or iWrite commands.

  Note

  The presence of the DefaultLoadValue statement alone is not sufficient to establish a particular value for the stability analysis after test_setup. If you mean to treat a value at a DataInPort as stable during stability analysis, then you must force the port to that value, for example by means of an iWrite in one of the test_setup iCalls.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, it only defines the structure of the Attribute parameter by the language. You define the attribute name and the attribute value, and any IJTAG software tool may ignore it. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

Example 1

The following example shows an ICL module definition with only DataInPort, DataOutPort, and a ClockPort port functions. This module does not include any scan interface. Its only way of data IO is through the parallel data port in[7:0] and out[7:0]. Observe that the body of the module is empty, not containing any other ICL objects, like DataRegister, or LogicSignal.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module instrument {
   DataInPort  in[7:0];
   DataOutPort out[7:0];
   ClockPort   clk;
}
```

Example 2

The following example shows an ICL module definition with both a parallel data input/output and a scan interface (implicitly defined, since there is only one scan-in / scan-out pair). Observe how it uses the data in port inside the ScanRegister. It is declared as the source of the capture values of the scan register, which means that the scan register captures the value from the data in ports fq on the positive edge of the CaptureEnPort ce, under the condition that the SelectPort en is also high. Finally, observe the usage of the RefEnum and Enum statements.

```
Module tdr_ref {
   ScanInPort       si;
   ScanOutPort      so       {  Source R[0];   }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;
   DataInPort       fq[7:0]  { RefEnum OptionalValues; }
   DataOutPort      td[7:0]  { Source R[7:0]; RefEnum OptionalValues; }
   ScanRegister R[7:0] {
      CaptureSource fq[7:0];
      ScanInSource  si;
      RefEnum       OptionalValues ;
   }   Enum OptionalValues {
       reset = 8'b0 ;
       set   = 8'b11111111 ;
       chain = 8'b11001100 ;
       green = 8'hee ;
       blue  = 8'd198 ;
   }
}
```

Example 3

The following example shows how you can use a data input port to select a scan path. In such a case, the data input port falls under the control of the Tessent tool, which determines the necessary data in values to fulfill any read or write requests to the scan chains.

```
Module msel {
  ScanOutPort so { Source smux; }
  DataInPort  dataSel;
  ScanMux smux SelectedBy dataSel {
    1'b0 : tdr0[0];
    1'b1 : tdr1[0];
  }
  [...]
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

DataOutPort

DataInOutPort

ScanInPort

ScanOutPort

Enum

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DataInOutPort

Declares the name of the data inout port (bidirectional port) of a Module.

Usage

Syntax 1

```
Module module_name {
   DataInOutPort port_name ;
}
```

Syntax 2

```
Module module_name {
   DataInOutPort port_name {
      Source source_name ;
      Attribute att_name = att_value ;
   }
}
```

Description

A port function that declares the name of the data inout port (bidirectional data port) of a Module.

This port function is not part of the IEEE 1687-2014 standard. Its only use model within Tessent is the representation of the SPIO port of the Two-Pin Serial Port (TPSP) interface (see "TwoPinSerialPort" on page 4055). It cannot be used for any other purpose.

Arguments

- *port_name*

  A required string that identifies the name of a data inout port of the module. The port names for all ports of a module must be unique. In addition, you may not use the port name for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL DataInOutPort of the module that is also used for the respective ports of the module in the design.

- Source *source_name* ;

  An optional keyword-string pair that identifies the source of the parallel output data, such as a DataOutPort or a DataInOutPort of another module instance. A DataInOutPort may have at most one Source statement. Further, the width of the Source must match the width of the DataInOutPort.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and icl_pin as

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

defined in this manual. Most attributes are read-only and are automatically generated by the tool.

Examples

The following example shows the typical use of the DataInOutPort construct as part of the interface of a design with a TPSP controller.

```
Module chip {
    ClockPort SPCLK {
      Attribute function_modifier = "two_pin_serial_clock_port";
   }
   DataInOutPort SPIO {
      Source tpsp_inst.spio_out;
      Attribute function_modifier = "two_pin_serial_inout_port";
   }
   Instance tpsp_inst Of tpsp_controller {
      InputPort spio_in = SPIO;
      InputPort spclk   = SPCLK;
      InputPort tdo     = block_inst.tdo;
   }
   Instance block_inst Of block {
      InputPort tms  = tpsp_inst.tms;
      InputPort trst = tpsp_inst.trst;
      InputPort tck  = tpsp_inst.tck;
      InputPort tdi  = tpsp_inst.tdi;
   }
}
```

Related Topics

DataInPort

DataOutPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DataMux

Creates and names a data multiplexer.

Usage

```
Module module_name {
  DataMux mux_name SelectedBy mux_select_signal {
      Attribute att_name = att_value;
    mux_select_value : data_source_name ;
  }
}
```

Description

The DataMux statement creates and names a data multiplexer.

IJTAG differentiates between a scan multiplexer, a data multiplexer, and a clock multiplexer. All three share the same syntax (other than the keyword), but their semantics differ. The scan multiplexer only switches between scan signals, the data multiplexer only switches between data signals, and the clock multiplexer only switches between clock signals. No signal types may be mixed for any of the multiplexers. The select signal of the multiplexers may be sourced only by data signals.

A DataMux can also be used for the gating of reset signals, tms signals and trst signals. See chapter "How to Model Global Reset, Local Reset and Embedded TAPs" in the *Tessent IJTAG User's Manual* for the details about the reset, trst and tms gating.

Arguments

- *mux_name*

  A string that identifies the name of a data multiplexer of the module. The name of the ScanMux must be unique among all port names, and among all the following objects: ScanRegister, DataRegister, ScanMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  Using this name, the outgoing signal of the multiplexer can be referenced. The width of *mux_name* must match the with of the *data_source_name*.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The possibility to attach attributes to multiplexers is not part of the IEEE 1687-2014 standard. It is an extension specific to Tessent Shell. The attribute name and the attribute value is defined by you and may be ignored or rejected by any IJTAG software tool.

- *mux_select_value* : *data_source_name*

  A repeatable string-string pair, connected by a colon symbol ':'. A DataMux must have at least one mux_select_value - data_source_name pair. The width of the *mux_select_value* must match the width of the *mux_select_signal*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The *mux_select_value* describes the value the select signal (*mux_select_signal*) of the mux must assume for the *mux_name* signal to connect to the *data_source_name* signal. The *mux_select_value* may contain x-values. The collection of all specified *mux_select_value* parameters need not cover the entire number space. Any select value not listed causes the *mux_name* signal to assume x-values of the correct width.

A mux_select_value may consist of multiple different select values connected by OR ("|"). The width of each individual select value must match the width of the mux_select_signal.

Examples

The following shows usages of all the multiplexer types.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module chip2 {
   ScanInPort  siA ;
   ScanInPort  siB ;
   ScanOutPort so      { Source scanM ; }
   ShiftEnPort   se1 ;
   CaptureEnPort ce1 ;
   UpdateEnPort  ue1 ;
   ShiftEnPort   se2 ;
   CaptureEnPort ce2 ;
   UpdateEnPort  ue2 ;
   TCKPort tck ;
   ClockPort clkA ;
   ClockPort clkB ;
   ToClockPort clk { Source clockM ; }
   DataInPort   dselA ;
   DataInPort   dselB[2:0] ;
   DataOutPort do[3:0] { Source dataM[3:0] ; }
   ScanInterface cA12 {
      Port se1; Port ce1; Port ue1;
      Chain c1 { Port siA; Port so ; }
   }
   ScanInterface cB12 {
      Port se2; Port ce2; Port ue2;
      Chain c1 { Port siB; Port so ; }
   }
   ScanRegister regA[7:0] {
      ScanInSource siA ;
      ResetValue   8'b0 ;
   }
   ScanRegister regB[7:0] {
      ScanInSource siB ;
      ResetValue   8'b0 ;
   }
   ScanMux scanM SelectedBy dselA {
      1'b0 : regA[0] ;
      1'b1 : regB[0] ;
   }
   ClockMux clockM SelectedBy dselA {
      1'b0 : clkA ;
      1'b1 : clkB ;
   }
   DataMux dataM[3:0] SelectedBy dselB[2:0] {
      3'b000|3'b100  : regA[7:4] ;
      3'bx01         : regA[3:0] ;
      3'bx1x         : regB[5:2] ;
   }
}
```

Related Topics

ScanMux

ClockMux

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DataOutPort

Declares the name of the data out port of a Module.

Usage

Syntax 1

```
Module module_name {
  DataOutPort port_name ;
}
```

Syntax 2

```
Module module_name {
  DataOutPort port_name {
    Source source_name ;
    Enable enable_name ;
    RefEnum enum_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the data out port of a Module.

A module may have zero or more data out ports. The number and bus-width of all declared data out ports is not correlated to the number and bus-width of any DataInPort.

A data out port is used to execute a parallel read (that is, not a scan operation) to ports of the module or to readable objects inside the module, like DataRegister. If data is to be scanned-out, use the ScanOutPort port function instead. If a parallel read from ports or objects of the module is to be allowed, the DataOutPort function(s) must be declared for the module. If a module does not have DataOutPort port functions, no parallel data read is allowed. Certain additional restrictions apply as explained next.

If the data output port signal has SelectPort or ToSelectPort, which is part of an ScanInterface in its fan-in, then it may not be target of an iWrite or iRead operation. This ensures that the scan selection remains under control of the software application.

If a data out port is connected to an Instance or DataRegister with an AddressValue statement, all data out ports are concatenated in the order of declaration until the required data width is achieved. See the discussion of AddressValue for further explanation and an example.

If none of the parameters Source, Enable, Attribute, or RefEnum are used, you must use Syntax 1. However, in this case, the data connection is broken at the DataOutPort. Retargeting cannot use this port, and if no other solution can be found, reading and writing from the broken data connection results in an error during PDL command retargeting.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- *port_name*

  A required string that identifies the name of a data out port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL DataOutPort of the module that is also used for the respective ports of the module in the design.

- Source *source_name* ;

  An optional keyword-string pair that identifies the source of the parallel output data, like a DataRegister or a DataOutPort of another module instance. A DataOutPort may have at most one Source statement. Further, the width of the Source must match the width of the DataOutPort.

- Enable *enable_name* ;

  An optional keyword-string pair that identifies the enable signal of the data out port. The enable signal must become active in order to connect the data output port with the data source. The activation and deactivation of the enable signal is taken care of by Tessent. Please see the to learn more about the usage of the enable signal. A DataOutPort may have at most one Enable statement.

- RefEnum *enum_name* ;

  An optional keyword-string pair that references the name of a Related Topics table. Using the enumeration table, the writing to the data in port can use synonyms listed in the table, instead of bit strings. It must be understood that the enumeration table is not necessarily a complete list. It is still allowed to write to the data in ports any value, including those not listed in the enumeration table. A DataOutPort may have at most one RefEnum statement.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and icl_pin as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following example shows an ICL module definition with both a parallel data input/output and a scan interface (implicitly defined, since there is only one scan-in / scan-out pair). Observe how the data in port is used inside the ScanRegister. It is declared as the source of the capture values of the scan register, which means that the scan register captures the value from the data in ports fq on the positive edge of the CaptureEnPort ce, under the condition that the SelectPort en is also high. Finally, observe the usage of the RefEnum and Related Topics statements.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module tdr_ref {
   ScanInPort       si;
   ScanOutPort      so      {  Source R[0];   }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;
   DataInPort       fq[7:0]  { RefEnum OptionalValues; }
   DataOutPort      td[7:0]  { Source R[7:0]; RefEnum OptionalValues; }
   ScanRegister R[7:0] {
      CaptureSource fq[7:0];
      ScanInSource  si;
      RefEnum       OptionalValues ;
   }   Enum OptionalValues {
       reset = 8'b0 ;
       set   = 8'b11111111 ;
       chain = 8'b11001100 ;
       green = 8'hee ;
       blue  = 8'd198 ;
   }
}
```

Related Topics

DataInPort

DataInOutPort

ScanInPort

ScanOutPort

Enum

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DataRegister

Defines and names a data register of a certain width and with certain properties.

Usage

Syntax 1

```
Module module_name {
  DataRegister register_name {
    WriteDataSource concat_signal ;
    WriteEnSource enable_source ;
    ResetValue reset_value ;
    DefaultLoadValue load_value ;
    RefEnum enum_name ;
    Attribute att_name = att_value;
  }
}
```

Syntax 2

```
Module module_name {
  DataRegister register_name {
    AddressValue address_value ;

    ResetValue reset_value ;
    DefaultLoadValue load_value ;
    RefEnum enum_name ;
    Attribute att_name = att_value;
  }
}
```

Description

The statement DataRegister defines and names a data register of a certain width and with certain properties. These properties, like the reset value, are described through parameters of the data register. The DataRegister is used for parallel reading and writing as opposed to a ScanRegister, which offers a scan-based access.

There are three, mutually exclusive types of DataRegisters: A Selectable DataRegister, for which a signal must be defined from which the DataRegister captures data; an Addressable DataRegister, for which an AddressValue must be specified; and a CallBack DataRegister. You should use Syntax 1 if you have a Selectable DataRegister. You should use Syntax 2 if you have an Addressable DataRegister.

Further, the Addressable DataRegister can only be used inside a OneHotDataGroup, and inside a OneHotDataGroup there can only be Addressable DataRegisters.

Note

Currently Tessent does not support the Callback functionality.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Arguments

- *register_name*

  A required string that identifies the name of a data register of the module. The name of the DataRegister must be unique among all port names, and among all the following objects: ScanRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

- *ResetValue reset_value*

  An optional keyword-string pair that identifies the value that each data register cell assumes as a response to an active reset signal of the module. In more detail, it is the update stage of the scan register that assumes the declared reset value. If used, the width of *reset_value* must match the width of the register. If the ResetValue parameter is not used, the data register does not respond to the module's incoming reset signal. However, if the module contains data registers with a ResetValue parameter, the module must have only one ResetPort statement.

  When both the DefaultLoadValue parameter and the ResetValue parameter exist in the same DataRegister statement, the non-x bit values of *reset_value* must match those of *default_load_value*.

- *DefaultLoadValue default_load_value*

  An optional keyword-string pair that identifies the value that each data register cell assumes if no user-defined value had been written to it beforehand and if no reset had been triggered. If used, the width of *default_load_value* must match the width of the register.

  When both the DefaultLoadValue parameter and the ResetValue parameter exist in the same DataRegister statement, the non-x bit values of *reset_value* must match those of *default_load_value*.

- *RefEnum enum_name*

  An optional keyword-string pair that references the name of an Enum table. Using the enumeration table, reading and writing to the data register can use synonyms listed in the table, instead of bit strings. Understand that the enumeration table is not necessarily a complete list. It is still allowed to read or write any value, including those not listed in the enumeration table. A DataRegister statement may have at most one RefEnum parameter. In addition, the width of the elements in the enumeration table must match the width of the data register.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin_attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- WriteDataSource *concat_signal*

  A keyword and string pair required for a selectable DataRegister. *concat_signal* references a single signal or a concatenation of signals from where the DataRegister captures data in response to a write request when WriteEnSource turns active (high). concat_signal must refer either to the module's DataInPorts, another DataRegister, or to a ScanRegister.

- WriteEnSource *enable_source*

  A keyword and string pair required for a selectable DataRegister. *enable_source* references a 1-bit wide signal which determines when the DataRegister can capture data from WriteDataSource. The DataRegister captures data in response to a write request when WriteEnSource turns active (high).

- AddressValue *address_value*

  A keyword and number pair required if an Addressable DataRegister; that is, the DataRegister is part of the OneHotDataGroup. The AddressValue parameter may not be used if the DataRegister if not part of a OneHotDataGroup. Within the OneHotDataGroup, the *number* declares a part of the address that is associated with the DataRegister. The final address of a DataRegister is determined by the sum of all addresses along the hierarchical instance path leading to the data register. In this sense *number* is only one element of the sum, comprising the address of a data register of the OneHotDataGroup.

  The DataRegister is loaded from the logical data-in signal with the right most indices first, when the DataRegister's address has been applied and the module's WriteEnPort is active (high). The address of a DataRegister is computed by adding all address_values of all Instances leading to the DataRegister.

  Similarly, reading from the Addressable DataRegister requires the module's ReadEnPort being active (high), and the address of the DataRegister being applied.

Examples

The following example shows a selectable data register. An example of addressable data registers can be found with the OneHotDataGroup statement, as well as with the Instance statement.

Observe the usage of the Parameter statement as well as Parameter overwrites. Further observe how the write enable signal is computed for the data registers in module bank using a LogicSignal statement and concatenation of the parameterized signals of the comparisons.

All together, this example shows how a selectable DataRegister can be used very much in a way of an addressable DataRegister, but without any of the constraints that come with an addressable DataRegister. In this particular implementation, the read and write enable signal of bank are explicitly managed (see for example, the usage of Dx_Mux). Depending on your DataRegister implementation, you may be able to relax this and allow reading from and writing to a data register simultaneously. Tessent sets up constraints for the IJTAG software such that no race conditions can occur.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module chip_selectable_dataregister {
   DataInPort  write_enable ;
   DataInPort  read_enable ;
   DataInPort  address[7:0] ;
   DataInPort  dataIn[15:0] ;
   DataOutPort dataOut[15:0] { Source outMux[15:0] ; }
   Parameter bank1Address = 4'b0011 ;
   Parameter bank2Address = 4'b1100 ;
   DataMux outMux[15:0] SelectedBy address[7:4] {
     $bank1Address : bank1.dataOut[15:0] ;
     $bank2Address : bank2.dataOut[15:0] ;
   }
   Instance bank1 Of bank {
     InputPort we = write_enable ;
     InputPort re = read_enable ;
     InputPort dataIn[15:0] = dataIn[15:0] ;
     InputPort address[3:0] = address[3:0] ;
   }
   Instance bank2 Of bank {
     InputPort we = write_enable ;
     InputPort re = read_enable ;
     InputPort dataIn[15:0] = dataIn[15:0] ;
     InputPort address[3:0] = address[3:0] ;
     Parameter D0_Address = 4'b1000 ;
     Parameter D1_Address = 4'b1100 ;
   }
}
Module bank {
   DataInPort    we ;  Alias weR = we { iApplyEndState 1'b0 ; }
   DataInPort    re ;
   DataInPort    address[3:0];
   DataInPort    dataIn[15:0];
   DataOutPort   dataOut[15:0] { Source Dx_Mux[15:0] ;
 Enable re; }   Parameter D0_Address = 4'b0000 ;
   Parameter D1_Address = 4'b0001 ;
   DataMux Dx_Mux[15:0] SelectedBy we,address[3:0] {
     1'b0, $D0_Address : D0[15:0] ;
     1'b0, $D1_Address : D1[15:0] ;
   }
   LogicSignal D0_wen {  re,we,address[3:0] == 2'b01, $D0_Address ; }
   DataRegister D0[15:0] {
     ResetValue       16'b0 ;
     WriteEnSource    D0_wen ;
     WriteDataSource  dataIn[15:0] ;
   }
   LogicSignal D1_wen {  re,we,address[3:0] == 2'b01, $D1_Address ; }
   DataRegister D1[15:0] {
     ResetValue       16'b0 ;
     WriteEnSource    D1_wen ;
     WriteDataSource  dataIn[15:0] ;
   }
}
```

Related Topics

DataInPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DataOutPort

ScanRegister

OneHotDataGroup

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Enum

Adds an entry to an enumeration table that links a mnemonic string to an arbitrary value.

Usage

```
Module module_name {
  Enum enum_name {
    name = value ;
  }
}
```

Description

With the Enum statement, ICL introduces an enumeration table that links a mnemonic string to an arbitrary value. This string can then be used in lieu of the value at places like DataInPort, DataOutPort, and LogicSignal. For the ScanRegister and the DataRegister the string can be used for the DefaultLoadValue, ResetValue, and CaptureSource (ScanRegister only).

Further, PDL commands like iWrite or iRead can also use the name instead of the value. Choosing meaningful mnemonics can make both, ICL and PDL, much more human readable.

In no usage of the enumeration table are you limited to only the names listed in the table. You are free to continue using any legal value, and in particular values not listed in the table. The example below shows such a case.

Arguments

- *enum_name*

  A required string that identifies the name of the enumeration table. Using this name the RefEnum parameter inside the ScanRegister statement, for example, can reference the enumeration table. The *enum_name* must be unique among all Enum statements within the module.

- *name = value*

  A required, repeatable string-value pair, connected with the '=' symbol. These pairs make up the enumeration table. All values must be of the same width, and the names must be unique within the same Enum. An enumeration table must contain at least one name-value pair.

Examples

The following enumeration table shows several examples of value definitions. The width of the value depends on the usage of the table entries in other objects, like ScanRegister or DataInPort, referenced by RefEnum. within these statements. The number type is not relevant as shown. Also observe that two Parameter were defined, which are used in the enumeration table 'Modes'. As usual, these parameters could be overwritten differently in each instance of the module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module instrument {
   DataInPort  in[7:0];
   DataOutPort out[7:0];
   ClockPort   clk;

   Alias enable      = in[7]            { RefEnum YesNo;    }
   Alias mode[3:0]   = in[6:5],in[3:2]  { RefEnum Modes;    }
   Alias p1[2:0]     = in[4],in[1:0];
   Alias go          = out[0]           { RefEnum PassFail; }
   Alias done        = out[1]           { RefEnum YesNo;    }
   Alias count[5:0]  = out[7:2];
   Parameter userEnum1val = 4'b1111 ;
   Parameter userEnum2val = 4'h22 ;
   Enum PassFail {
      Pass = 1'b1;
      Fail = 1'b0;
   }
   Enum YesNo {
      Yes  = 1'b1;
      No   = 1'b0;
   }
   Enum Modes {
      none      = 4'b0 ;
      red       = 4'b0011 ;
      blue      = 4'haa ;
      green     = 4'd9 ;
      userEnum1 = $userEnum1val ;
      userEnum2 = $userEnum2val ;
   }
}
```

Next, here is a snippet of a dofile, writing to the ‘mode’ Alias of the module, using the names defined in the ‘Mode’ enumeration table which is associated with the alias through the RefEnum parameter. The done Alias is then read from ‘out’.

```
open_pattern_set enumTest
  iWrite mode red
  iApply
  iWrite mode 0b1010
  iApply
  iRead  done Yes
  iApply
close_pattern_set
write_patterns enumTest.pdl -pdl
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following lists the contents of the saved PDL file enumTest.pdl:

```
iProcsForModule instrument
iProc enumTest {} {
   iWrite in[6] 0b0
   iWrite in[5] 0b0
   iWrite in[3] 0b1
   iWrite in[2] 0b1
   iApply
   iWrite in[6] 0b1
   iWrite in[5] 0b0
   iWrite in[3] 0b1
   iWrite in[2] 0b0
   iApply
   iRead out[1] 0b1
   iApply
}
```

Related Topics

iRead

LogicSignal

ScanRegister

iWrite

DataInPort

DataOutPort

DataRegister

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Instance

Names an instance of an ICL module.

Usage

Syntax 1

```
Module module_name {
  Instance instance_name Of module_name {
    Attribute att_name = att_value ;
    InputPort moduleinputport_name = concat_signal ;
    Parameter parameter_name = parameter_value ;
    AddressValue number ;
  }
}
```

Syntax 2

```
Module module_name {
  Instance instance_name Of module_name ;
}
```

Description

The Instance statement names an instance of an ICL module. It also declares the address and/or all input connections of the instance to source signals of other instances or the top level module. Note that in ICL no output connections are listed in the Instance statement.

In addition to the input port connections, Parameters defined within the module definition may be overwritten. LocalParameter cannot be overwritten though.

Syntax 2 must be used if the module has no input ports and none of the other parameters is used.

Arguments

- *instance_name Of module_name*

  A required string-string pair, connected with the case-sensitive keyword ‘Of’, defining the name of the instance and referencing an existing module name of which an instance is to be created. Within a module, all instance names must be unique. Also, no module may contain an instance of itself anywhere through its hierarchy.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value are defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin_attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- InputPort *moduleinputport_name* = *concat_signal*

  A repeatable, required keyword and string-value pair, connected with the '=' symbol. For each input port defined in the module there must be exactly one such line in the Instance statement, however exceptions apply as explained below. moduleinputport_name must refer to an input port name of the referenced module. For modules without any input ports the InputPort parameter may not be used. The order of the input port declaration in the referenced module and the order of the InputPort parameters can be different.

  concat_signal may reference a single signal or a concatenation of signals of the same or compatible type. The type of *concat_signal* must match the type of the input port definition in the module, or must be a compatible type. *concat_signal* may only be one of these signals: reset, scan, data, read enable, write enable, clock, TCK, shift enable, capture enable, update enable, TMS, or TRST. The width of the module's port definition and *concat_signal* must match. For an indexed input port of the module, the InputPort parameter may be used as often as necessary to cover the entire index range in any order. However, overlapping indices or ranges are not allowed. For instances without the AddressValue parameter, the InputPort parameter may connect the module's WriteEnPort, ReadEnPort, AddressPort, or SelectPort to data signals; that is, a data signal is a compatible source for those port functions.

  Note that not all input ports of a module must be connected explicitly. For a non-handoff module; that is, for a module that is not the top level module you hand over to a third party, several connections are automatically implied, if their source in the parent module can be determined unambiguously. It is legal to list these connections through the InputPort parameter, but it is not necessary. The following connections are automatically implied for non-hand-off module instances that do not have the AddressValue statement: TCK, shift enable, capture enable, update enable.

- Parameter *parameter_name* = *parameter_value*

  A repeatable, optional keyword and string-value pair, connected with the "=" symbol. The referenced *parameter_name* must have been defined in the module referenced by *module_name*; also the type of *parameter_value* must match the type used in the module for the parameter.

  Using the Parameter statement again within Instance overwrites the respective parameter value set in the module definition, but only for the current instance. Another instance of the module may use another overwrite value, or may use the original value defined in the module. Through this construct, ICL can define parameterized modules, where the final parameter is defined during each instantiation of the module.

- AddressValue *number*

  A required keyword and number pair, if this instance is part of the OneHotDataGroup. The AddressValue parameter may not be used if the instance if not part of a OneHotDataGroup. Within the OneHotDataGroup, the *number* declares a part of the address that is associated with the instance. The final address of a DataRegister is determined by the sum of all addresses along the hierarchical instance path leading to the data register. In this sense

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

number is only one element of the sum, comprising the address of a data register of the OneHotDataGroup.

The following connections are automatically implied for non-hand-off module instances that do have the AddressValue statement: TCK, the instance's AddressPort is assumed to be connected to the logical address signal of the parent module, the instance's WriteEnPort is assumed to be connected to the WriteEnPort of the parent module, the instance's ReadEnPort is assumed to be connected to the ReadEnPort of the parent module, and the instance's DataInPort is assumed to be connected to the logical data in signals of the parent module.

Note

In IJTAG, the Instance statement describes the "AllowBroadcastingOnScanInterface" parameter. This parameter is currently not supported in Tessent.

Examples

Example 1

The following example demonstrates the usage of automatic default connections in the common case of a scan interface. Observe that the instance declaration for tdr_inst of module tdr only declares the scan input port connection. TCK and all the enable signals are connected fully automatically to the respective signals of the parent module. Explicit connections are only necessary, if there are multiple signals of the same type in the parent module, for example two select ports.

```
Module chip_tdr {
   ScanInPort        si;
   ScanOutPort       so        { Source tdr_inst.so; }
   ShiftEnPort       se;
   SelectPort        en;
   CaptureEnPort     ce;
   UpdateEnPort      ue;
   TCKPort           tck;
   Instance tdr_inst Of tdr {
     InputPort si = si ;
   }
}
Module tdr {
   ScanInPort        si;
   ScanOutPort       so        { Source R[0]; }
   ShiftEnPort       se;
   SelectPort        en;
   CaptureEnPort     ce;
   UpdateEnPort      ue;
   TCKPort           tck;
   ScanRegister R[7:0] {
      ScanInSource      si;
      ResetValue        8'b0 ;
   }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Example 2

The following example demonstrates not only the usage of the OneHotDataGroup, but also the usage of the InputPort parameter and some of the compatible signal connections. Observe that module “top” does have neither a ReadEnPort, nor a WriteEnPort. Instead it has 2 DataInPort statements that are connected to the respective input ports of the instances of module “phy”. Since the connection of the data signals of the parent module (that is, top) to the instances is done implicitly, by the order of declaration, it is important that the 16 data input bits of DataIn are declared before the 2 bits of the write and read enable. Also note that the module top defines just one AddressPort of 8 bits, which is implicitly mapped to the two 4-bit AddressPorts of phy in their order of declaration. Further, observe that the AddressPort of “bank” only 2-bits wide, since only two bits are required for this module’s operation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module top {
   AddressPort   address[7:0] ;
   DataInPort    dataIn[15:0] ;
   DataInPort    write_enable ;
   DataInPort    read_enable ;
   DataOutPort   dataOut[15:0] {
     Source outMux[15:0] ;
     Enable read_enable ;
   }
   OneHotDataGroup outMux[15:0] {
     Instance phy1 Of phy {
       AddressValue 2'b01, 6'b000000 ;
       InputPort we = write_enable ;
       InputPort re = read_enable ;
     }
     Instance phy2 Of phy {
       AddressValue 2'b10, 6'b000000 ;
       InputPort we = write_enable ;
       InputPort re = read_enable ;
     }
   }
}
Module phy {
   WriteEnPort 7 we ;
   ReadEnPort    re ;
   AddressPort   address1[3:0] ;
   AddressPort   address2[3:0] ;
   DataInPort    dataIn1[7:0] ;
   DataInPort    dataIn2[7:0] ;
   DataOutPort   dataOut[15:0] { Source outMux[15:0] ;
 }
   OneHotDataGroup outMux[15:0] {
     Instance bank1 Of bank {
       AddressValue 4'b0001,2'b00 ;
     }
     Instance bank2 Of bank {
       AddressValue 4'b0010,2'b00 ;
     }
   }
}
Module bank {
   WriteEnPort   we ;
   ReadEnPort    re ;
   AddressPort   address[1:0];
   DataInPort    dataIn[15:0];
   DataOutPort   dataOut[15:0] { Source outMux[15:0] ;
 }   OneHotDataGroup outMux[15:0] {
//   AddressValue 2'bx,4'bx,2'b<num>
     DataRegister D0[15:0] { AddressValue 2'b00 ; }
     DataRegister D1[15:0] { AddressValue 2'b01 ; }
     DataRegister D2[15:0] { AddressValue 2'b10 ; }
     DataRegister D3[15:0] { AddressValue 2'b11 ; }
   }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

ScanInterface

DataRegister

ScanRegister

OneHotDataGroup

OneHotScanGroup

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

IRSelectPort

Declares the name of the IR select port of a Module.

Usage

```
Module module_name {
  IRSelectPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the IR select port of a Module. This signal is used to select between the instruction register of a TAP controller and another scan register, usually a data register.

Arguments

- *port_name*

  A required string that identifies the name of the IR select port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several icl_port attributes and icl_pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

LocalParameter

Has the same semantic and use model as the Parameter statement but cannot be overwritten in the Instance statement.

Usage

```
Module module_name {
  LocalParameter parameter_name = parameter_value ;
}
```

Description

A LocalParameter has the same semantic and use model as the Parameter statement. The only difference is that a LocalParameter cannot be overwritten in the Instance statement.

Arguments

- *parameter_name* = *parameter_value*

  A required string-value pair, connected with the '=' symbol. The parameter name must be unique across all Parameter and LocalParameter statements within the module. The parameter value may be exactly one of the following: an unsized number, a sized number, or a string. A parameter reference representing an unsized number may only be used where an unsized number is expected. Similarly for the sized number. A parameter reference representing a string may only be used in an attribute within any statement in the module, or within a Callback function. Places where a parameter reference is allowed include an attribute value, the index value for a vector identifier, or where a size or unsized number is allowed. For example, you cannot use a parameter to substitute any arbitrary number, such as creating object names, like an Instance name "inst$size", where "size" is a parameter

  Although parameters can be used to define other parameters, a circular definition is not allowed.

Related Topics

Parameter

Instance

Alias

Enum

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

LogicSignal

Creates a new signal that can then be referenced in other ICL statements.

Usage

```
Module module_name {
  LogicSignal signal_name {
    expression ;
  }
}
```

Description

The LogicSignal statement creates a new signal named *signal_name*, which itself can then be referenced in other ICL statements, and in particular other LogicSignal statements.

Using the LogicSignal statement, logical operations between signals like AND or EXOR can be implemented.

Arguments

- *signal_name*

  A required string that identifies the newly created signal name. The value it assumes depends on the result of the evaluation of *expression*.

  The name of the LogicSignal must be unique among all port names, and among all the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, Alias.

  In any case, the width of signal_name is always 1. In particular, comparisons and Boolean logic operations return 1'b1, if the expression evaluates to true, and 1'b0, if the expression evaluates to false.

- *expression*

  A required string that defines a Boolean expression that evaluates to a single-bit result, based on the signals and their value used in the expression.

  The common Boolean and Logical operators are supported. In decreasing order of preference, these are: Boolean not (!), Bitwise not (~), logical AND (&&) and logical OR(||), bitwise AND (&), bitwise OR (|), and bitwise XOR (^), logical equality (==) and inequality (!=).

  All operators are left-to-right associative, and operators with equal precedence are evaluated left-to-right. Associativity and precedence may be expressly specified with parentheses ().

  Valid operands are registers, ports, sized and unsized numbers, logical signals, as well as parameters and aliases that resolve into those. In addition, signals can be concatenated.

  For expressions containing unsized numbers, it must be possible to determine the required width of the unsized number unambiguously. If the width must be extended, the additional

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

bits are either filled with 0, if the most significant bit is either 0 or 1, or the bits are filled with unknown (x).

For the logical equality and inequality operation, the width of the operands on either side of the operator must be the same.

Any parameter reference is resolved before the expression is evaluated.

Enumeration table entries can be used for operands which are used in their entirety in the expression, and not just a part of the operand. The operand must have declared its binding to this enumeration by means of the RefEnum statement. Enumeration values are allowed only in combination with equality (==) and inequality (!=) operators. See the example below.

Examples

The following example shows several usages of the LogicalSignal statement. sig1 demonstrates the usage of an enumeration table value entry. Note that ‘mode[3:0]’ is the entire operand for which the Enum ‘Modes’ applies. Therefore, any table entry may be used for this operand. sig2 shows the usage of concatenation, the expansion of an unsigned number and the usage of a parameter reference. Observe that the width of the right operand is 5, whereas the width of the sized elements of the left operand is 3. Therefore, the unsigned number ‘0’ is expanded to width 2, matching the width of the right operand. Lastly, sig3 shows an example of bit-operations.

```
Module instrument {
   [ ... ]
   DataInPort  in[7:0];
   [ ... ]
   Alias enable      = in[7]           { RefEnum YesNo;    }
   Alias mode[3:0]   = in[6:5],in[3:2] { RefEnum Modes; }
   Parameter user1 = 4'b1010 ;
LogicSignal sig1 { (mode==red) && (enable==1'b1) ; }
   LogicSignal sig2 {  in[4],1'b1,in[0],0 == 1'b0,$user1 ; }
   LogicSignal sig3 { (in[6:5] | in[3:2]) != 2'b10 ; }
   Enum Modes {
      none      = 4'b0 ;
      red       = 4'b0011 ;
      blue      = 4'haa ;
      green     = 4'd9 ;
   }
   [ ... ]
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Module

Defines the name of an ICL module.

Usage

```
Module module_name {
  Attribute att_name = att_value;
  Parameter param_name = param_value;
  LocalParameter param_name = param_value;
  ScanInterface interface_name { … }
//---- Port Functions
  ScanInPort port_name { … }
  ShiftEnPort port_name { … }
  CaptureEnPort port_name { … }
  UpdateEnPort port_name { … }
  DataInPort port_name { … }
  DataOutPort port_name { … }
  SelectPort port_name { … }
  ResetPort port_name { … }
  TMSPort port_name { … }
  TCKPort port_name { … }
  ClockPort port_name { … }
  TRSTPort port_name { … }
  AddressPort port_name { … }
  WriteEnPort port_name { … }
  ReadEnPort port_name { … }
  ScanOutPort port_name { … }
  ToShiftEnPort port_name { … }
  ToUpdateEnPort port_name { … }
  ToCaptureEnPort port_name { … }
  ToResetPort port_name { … }
  ToSelectPort port_name { … }
  ToClockPort port_name { … }
  ToTCKPort port_name { … }
  ToTMSPort port_name { … }
  ToTRSTPort port_name { … }
  ToIRSelectPort port_name { … }

  Instance instance_name Of another_module_name { … }
  LogicSignal signal_name { … }
  ScanRegister register_name { … }
  DataRegister register_name { … }
  ScanMux mux_name SelectedBy selector { … }
  DataMux mux_name SelectedBy selector { … }
  ClockMux mux_name SelectedBy selector { … }
  OneHotDataGroup name { … }
  OneHotScanGroup name { … }
  Enum enum_name { … }
  Alias alias_name = element_list { … }
  AccessLink instance_name Of link_type { … }
}
```

Description

The syntax above lists the complete contents (statements) of an ICL module. Not all statements may be used at the same time.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Each ICL statement is explained in detail in the following pages. All statements are hyperlinked from here to the respective statement description, as well as back to this module description.

Arguments

- module_name

  The Module statement defines the name of an ICL module. The name of the module must be unique among all loaded modules. To build an ICL design, you use the Instance statement.

Related Topics

NameSpace

Instance

UseNameSpace

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

NameSpace

Places all subsequent modules in a specific namespace.

Note

Currently Tessent does not support the NameSpace statement.

Related Topics

UseNameSpace

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

OneHotDataGroup

Implements a convenient way of declaring a named, hierarchically distributed data multiplexer.

Usage

```
Module module_name {
  OneHotDataGroup onehotdatagroup_name {
    Instance ;
    DataRegister ;
    Port concat_signal ;
  }
}
```

Description

The OneHotDataGroup statement implements a convenient way of declaring a named, hierarchically distributed data multiplexer. You would use a OneHotDataGroup if various inputs to the multiplexer are sourced by data registers distributed across multiple instances. This statement simplifies the declaration of such a data multiplexer by automatically making the correct data input and data output connections, including read, write, and address connections.

A OneHotDataGroup may use any combination of the three options declaring the source input signal of the multiplexer. The order of the declarations is not relevant, since each source should have an enable signal defined in such a way that at most one source is selected at a time. The OneHotDataGroup then assumes the value of this selected, unique source, or the x-value, if none of the sources is selected, or if multiple data sources are selected simultaneously.

Arguments

- *onehotdatagroup_name*

  A required string, declaring the name of the OneHotDataGroup. This name can be considered as the name of the distributed multiplexer. As usual, *onehotdatagroup_name* can be used to refer to the output signal of this multiplexer, for example as a source of a DataOutPort statement. The width of *onehotdatagroup_name* must be larger or equal to the width of the connected output ports of the referenced Instance or DataRegister. *onehotdatagroup_name* assumes the value of the one selected source, or the x-value, if none of its sources are enabled.

- Instance

  An optional, repeatable Instance statement. This instance must use the AddressValue parameter.

- DataRegister

  An optional, repeatable DataRegister statement. This DataRegister must use the AddressValue parameter.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- *Port concat_signal*

  An optional, repeatable keyword-string pair. *concat_signal* may reference a data output single signal or a concatenation of such signals. *concat_signal* may also be a hierarchical instance pathname ending at a data source signal. The referenced signal is considered an input port of the multiplexer represented by the OneHotDataGroup statement. For the enable signal required for each source of the OneHotDataGroup's source signal, one of the following rules apply:

  - For a referenced DataOutPort, the Enable statement is included in the DataOutPort declaration.
  - For a referenced DataOutPort, the Source parameter references another data signal, which has an enabling condition.
  - The instance referenced in the instance pathname in a hierarchical *concat_signal* has an AddressValue parameter.
  - A primary input port of the top level module is assumed to have an enabling condition from the outside.

Examples

The following shows an example of a OneHotDataGroup. Observe how the address of a DataRegister is determined. For example the DataRegister with the hierarchical name phy1.bank1.D1 has the address of 8'b01 000000 + 6'b0001 00 + 2'b01 = 8'b01 0001 01. Similarly, the DataRegister phy2.R has the address 8'b10 000000 + 6'b0010 00 = 8'b10 0010 00.

The Example 2 of the Instance statement shows a variation of this example.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module top {
   AddressPort   address[7:0] ;
   DataInPort    dataIn[15:0] ;
   WriteEnPort   write_enable ;
   ReadEnPort    read_enable ;
   DataOutPort   dataOut[15:0] { Source outMux[15:0] ; }
   OneHotDataGroup outMux[15:0] {
     Instance phy1 Of phy {
       AddressValue 2'b01, 6'b000000 ;
     }
     Instance phy2 Of phy {
       AddressValue 2'b10, 6'b000000 ;
     }
   }
}
Module phy {
   DataInPort    dataIn1[7:0] ;
   DataInPort    dataIn2[7:0] ;
   WriteEnPort   we ;
   ReadEnPort    re ;
   AddressPort   address1[3:0] ;
   AddressPort   address2[3:0] ;
   DataOutPort   dataOut[15:0] { Source outMux[15:0] ; }
   OneHotDataGroup outMux[15:0] {
     DataRegister R[15:0] {
       AddressValue 4'b0010,2'b00 ;
     }
     Instance bank1 Of bank {
       AddressValue 4'b0001,2'b00 ;
     }
   }
}
Module bank {
   DataInPort    dataIn[15:0];
   WriteEnPort   we ;
   ReadEnPort    re ;
   AddressPort   address[1:0];
   DataOutPort   dataOut[15:0] { Source outMux[15:0] ; }
   OneHotDataGroup outMux[15:0] {
     DataRegister D0[15:0] { AddressValue 2'b00 ; }
     DataRegister D1[15:0] { AddressValue 2'b01 ; }
     DataRegister D2[15:0] { AddressValue 2'b10 ; }
     DataRegister D3[15:0] { AddressValue 2'b11 ; }
   }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
set_context patterns -ijtag
read_icl ./example_fixed.icl
set_current_design
set_system_mode analysis
open_pattern_set p1
iWrite phy1.bank1.D3[15:0] 0xaaaa
iApply
iRead phy2.R[15:0] 0xbbbb
iApply
close_pattern_set
write_pat p1.pdl -pdl -replace
cat p1.pdl
```

Related Topics

Instance

OneHotScanGroup

DataRegister

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

OneHotScanGroup

Implements a convenient way of declaring a named, hierarchically distributed scan multiplexer.

Usage

```
Module module_name {
  OneHotScanGroup onehotscangroup_name {
    Port source_name ;
  }
}
```

Description

The OneHotScanGroup statement implements a convenient way of declaring a named, hierarchically distributed scan multiplexer. You would use a OneHotScanGroup if various inputs to the multiplexer are sourced by scan-out ports distributed across multiple instances. This statement simplifies the declaration of such a scan multiplexer by automatically making the correct scan input and scan output connections.

Arguments

- *onehotscangroup_name*

  A required string, declaring the name of the OneHotScanGroup. This name can be considered as the name of the distributed multiplexer. As usual, *onehotscangroup_name* can be used to refer to the output signal of this multiplexer, for example as a source of a ScanOutPort statement. The width of *source_name* must be equal to the width of *onehotdatagroup_name*. Each scan source must have an enable signal defined in such a way that at most one source is selected at a time when the OneHotScanGroup is in the active scan path. This enable signal is not necessarily directly at the ScanOutPort of *source_name*, but might be located deeper in the hierarchy, for example, through cascaded OneHotScanGroups. When selected, the *onehotdatagroup_name* then assumes the value of this selected source, or the x-value, if none of the sources is selected, or multiple sources would had been selected.

- *Port source_name*

  A required, repeatable keyword-string pair. *source_name* may reference a scan output port single signal or a concatenation of such signals. *source_name* may also be a hierarchical instance pathname ending at a scan output port signal. The referenced scan output port signals are considered an input port of the multiplexer represented by the OneHotScanGroup statement.

Examples

The following shows a example of a OneHotScanGroup. Observe how easy it is to declare the hierarchically distributed scan multiplexer “outScanMux” and the usage of the OneHotScanGroup’s name in the Source statement of the top level scan-out port.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module chip_onehotscangroup {
   ScanInPort       si;
   ScanOutPort      so       { Source outScanMux; }
   ShiftEnPort      se;
   SelectPort       en[3:0];
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;
   OneHotScanGroup outScanMux {
     Port tdr_0.so ;
     Port tdr_1.so ;
     Port tdr_2.so ;
     Port tdr_3.so ;
   }
   Instance tdr_0 Of tdr {
     InputPort si = si ;
     InputPort en = en[0];
   }
   Instance tdr_1 Of tdr {
     InputPort si = si ;
     InputPort en = en[1];
   }
   Instance tdr_2 Of tdr {
     InputPort si = si ;
     InputPort en = en[2];
   }
   Instance tdr_3 Of tdr {
     InputPort si = si ;
     InputPort en = en[3];
   }
}
Module tdr {   ScanInPort       si;
   ScanOutPort      so       { Source R[0]; Enable en; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;
   ScanRegister R[7:0] {
      ScanInSource     si;
      ResetValue       8'b0 ;
   }
}
```

Related Topics

Instance

OneHotDataGroup

ScanRegister

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Parameter

Define a named variable and assigns a default value.

Usage

Syntax 1 : Defining a parameter

```
Module module_name {
  Parameter parameter_name = parameter_value ;
}
```

Syntax 2 : Optionally overwriting a parameter in an Instance statement

```
Module module_name {
  Instance instance_name of other_module_name {
    Parameter parameter_name = parameter_value ;
  }
}
```

Description

With the help of the Parameter statement you can define a named variable and assign a default value.

The default value is specified in the module that declares the Parameter. The value can be overwritten with every instance of the Module. Through this, ICL allows you to create parameterized modules.

You build a reference to the Parameter by using the '$' symbol in front of the parameter_name.

The parameter reference can be used in many locations, for example in an Enum statement. The reference is resolved to the current value before the statement the Parameter is evaluated.

Arguments

- Parameter *parameter_name* = *parameter_value*

  A required string-value pair, connected with the '=' symbol. The parameter name must be unique across all Parameter and LocalParameter statements within the module. The parameter value may be exactly one of the following: an unsized number, a sized number, or a string. A parameter reference representing an unsized number may only be used where an unsized number is expected. Similarly for the sized number. A parameter reference representing a string may only be used in an attribute within any statement in the module, or within a Callback function. Places where a parameter reference is allowed include an attribute value, the index value for a vector identifier, or where a size or unsized number is allowed. For example, you cannot use a parameter to substitute any arbitrary number, such as creating object names, like an Instance name "inst$size", where "size" is a parameter

  Although parameters can be used to define other parameters, a circular definition is not allowed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Currently Tessent does not support the Callback functionality.

Examples

Example 1

The following example shows several usages of Parameter. In particular observe that the Parameter ‘Width’ is used to parameterize the values in the enumeration table.

```
Module inst {
   ScanInPort       si;
   ScanOutPort      so      {  Source R[0];   }
   TCKPort          tck;
   Parameter  Width  = 8 ;
   Parameter  Regmax = $Width - 1 ;
   Parameter  userValue = 8'hff ;
   Parameter  Val_A  = 4'b1001, $userValue, 4'b1111 ;
   Parameter  RegName = "R" ;
   LocalParameter  Regmin = 0 ;

   ScanRegister R[$Regmax:$Regmin] {
      Attribute     Register = "$RegName($RegMax:$RegMin)" ;
      ScanInSource  si;
      ResetValue    Reset ;
      RefEnum       SomeValues ;
   }   Enum SomeValues {
       Reset = $Width'b0 ;
       User1 = $userValue ;
   }
}
```

Example 2

The following example uses the module defined above. In its instantiation, the parameters ‘Width’ and ‘userValue’ is overwritten. This allows ICL to create parameterized modules, where the size of registers is fixed during the instantiation of the module and not during the declaration of the module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module top {
   ScanInPort       si;
   ScanOutPort      so      {  Source inst2.so;   }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;

   Instance inst1 Of inst {
      InputPort si = si ;
      Parameter Width = 16 ;
      Parameter userValue = 16'hffff ;
   }
   Instance inst2 Of inst {
      InputPort si = inst1.so ;
      Parameter Width = 4 ;
      Parameter userValue = 4'ha ;
   }
}
```

Related Topics

LocalParameter

Instance

Alias

Enum

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ReadEnPort

Declares the name of the read enable input port of a Module.

Usage

```
Module module_name {
  ReadEnPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the read enable input port of a Module. It is used if the module contains addressable instances or data registers.

This port serves as the read enable to the addressable instance and data registers.

A read operation is executed in two steps. At first the ReadEnPort must be activated (high) and the address must be supplied (AddressPort). In the second step, the capture is executed and the data is being read. During this second step the ReadEnPort is automatically deactivated (low) by means of an automatic iApplyEndState property (see the Alias statement).

Arguments

- *port_name*

  A required string that identifies the name of a read enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ScanInPort of the module that is also used for the respective ports of the module in the design.

  The source of the read enable port name signal may only be a read enable port of the parent module, or scan register bits of a ScanRegister statement. In both cases, this connection must be direct, and may not include any logic signals.

  A module may have at most one ReadEnPort statement. A module must have a ReadEnPort statement, if the module contains readable, addressable data registers or readable, addressable instances.

  A read enable port is assumed to have an automatic iApplyEndState property. See the Alias statement to learn more about iApplyEndState.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several icl_port attributes and icl_pin

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

AddressPort

Alias

DataRegister

Instance

WriteEnPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ResetPort

Declares the name of the reset input port of a Module's client ScanInterface.

Usage

```
Module module_name {
  ResetPort port_name {
    Attribute att_name = att_value ;
    ActivePolarity 0 | 1 ;
  }
}
```

Description

A port function that declares the name of the reset input port of a Module's client ScanInterface.

You have the option to chose the active polarity of the port.

A ResetPort can be explicitly connected to various sorts of drivers by means of the InputPort binding in the instantiation of the module, causing the ResetPort to become active under different conditions (synchronous reset, asynchronous reset, enforced local reset and so on). If the driver has not been specified explicitly, then the ResetPort may be implicitly connected to a reset signal in the parent module. See the chapter "How to Model Global Reset, Local Reset and Embedded TAPs" in the *Tessent IJTAG User's Manual* for the details on the reset functionality and the different ways of connecting a ResetPort.

Arguments

- *port_name*

  A required string that identifies the name of a reset port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ResetPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

- ActivePolarity 0 | 1

  An optional keyword-value pair, that defines if the reset port of the module is considered active low (value = 0), or active high (value =1). If the ActivePolarity statement is not given, the active polarity of the reset port is assumed to be high.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

ScanOutPort

ScanInterface

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ScanInPort

Declares the name of a scan input port of a Module.

Usage

```
Module module_name {
  ScanInPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of a scan input port of a Module.

A module may have zero or more scan-in ports. However, if a module has at least one ScanOutPort defined, then there must be at least one ScanInPort.

If a module has exactly one ScanInPort, than the module may also have a ScanInterface defined. However, if a module has more than one ScanInPort, than there must be at least one ScanInterface definition.

The scan-in port is used to execute a serial write (that is, not a parallel write operation). If a parallel write is to be done, use the DataInPort port function instead.

Arguments

- *port_name*

  A required string that identifies the name of a scan-in port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ScanInPort of the module that is also used for the respective ports of the design module.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following example shows an ICL module definition with a single scan-in / scan-out chain, therefore, no ScanInterface is required. The scan-in port is named “si”. Observe that the port name is reused in the body of the ScanRegister R[7:0] declaration. There, it declares that the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

scan register receives its scan data from the object named “si”, which happens to be the ScanInPort of the Module.

```
Module tdr2 {
   ScanInPort       si;
   ScanOutPort      so  { Source R[0]; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;
   ScanRegister R[7:0] {
      ScanInSource     si;
      ResetValue       8'b0 ;
      DefaultLoadValue 8'b0 ;
   }
}
```

Related Topics

CaptureEnPort

UpdateEnPort

ShiftEnPort

ScanOutPort

ScanInterface

DataInPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ScanInterface

Defines a set of ports of the module that together perform a scan operation.

Usage

```
Module module_name {
  ScanInterface interface_name {
    Attribute att_name = att_value ;
    Port port_name ;
    DefaultLoadValue load_value ;
    Chain chain_name { ... }
  }
}
```

Description

Refer to "ScanInterfaces and Associations Between Ports and ScanInterfaces" in the *Tessent IJTAG User's Manual* for an explanation of the creation, inference, checking, and introspection of ScanInterfaces.

Defines a set of ports of the module that together perform a scan operation.

The ScanInterface defines a set of ports of the module that together perform a scan operation. A typical example of ports include the shift enable, capture, and update enable ports, together with at least a pair of scan-in and scan-out ports. However, the ScanInterface is applied also to the TAP, since a shift-based interface to or from a module is also defined; that is, a scan interface. See the discussion on *port_names* below, which defines the sets of ports of the four recognized use models of the ScanInterface — the client interface, the host interface, the client TAP interface, and the host TAP interface. A module may have more than one scan interface, in particular several client interfaces or interfaces for different use models. However, only one scan interface may be active at the same time. This means, for example, that a set of scan chains that are supposed to be operated together must all be in the same ScanInterface definition.

For a handoff module, with more than one portfunction of type ScanInPort or ScanOutPort, there must be as many ScanInterface statements as there are scan interfaces in the module. This does not mean, for example, that the number of scan-in and scan-out port pairs defines the number of ScanInterface statements. Since several pairs of scan-in and scan-out ports may belong to the same scan interface, which means they are supposed to be operated simultaneously, the number of ScanInterface statements may be smaller.

The examples below are for simple ScanInterface definitions. Additional examples, especially those for multiple parallel scan chains, are provided in the ScanInterface/Chain section.

Arguments

- *interface_name*

  A required string that defines the name of the scan interface of the module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool.

- Port *port_name* ;

  A repeatable, optional keyword-string pair that references the name of an existing port of the module. Four use models of a ScanInterface can be distinguished. The port_names for a given ScanInterface must all belong to the same use model.

  - The client interface — Through a client scan interface a module receives all signals necessary to operate a defined set of scan chains. Ports of the following port functions form a client scan interface: at least one pair of ScanInPort and ScanOutPort, one ShiftEnPort and/or at least one SelectPort, and CaptureEnPort, UpdateEnPort, ResetPort, or TCKPort. There can be more than one CaptureEnPort or more than one UpdateEnPort in one client interface, as long as all ports with the same port function have distinct function_modifier attributes. If the shift enable, capture enable, or update enable ports are not stated and the module has exactly one of each, these ports are implied for the scan interface using the modules set of port functions, or for non-handoff modules, the set or implied set of port functions. If more than one pair of scan-in and scan-out ports belong to the same scan interface, the Chain object must be used. In this case, no port name should reference a scan-in and scan-out port, nor should the DefaultLoadValue be used.

    If a non-handoff module has exactly one client interface, than the ScanInterface statement is not required. The tool assumes a “default” client interface.

  - The host interface — Through a host scan interface a module defines shift operating control signals for another module. Ports of the following port functions form a host scan interface: one ScanInPort and/or one ScanOutPort, one ToShiftEnPort and/or at least one ToSelectPort, and ToCaptureEnPort, ToUpdateEnPort, ToResetPort, or ToTCKPort. There can be more than one ToCaptureEnPort or more than one ToUpdateEnPort in one host interface, as long as all ports with the same port function have distinct function_modifier attributes. Similar to the client interface, if the module has exactly one ToShiftEnPort, ToCaptureEnPort, or ToUpdateEnPort, these ports are assumed for each host scan interface, for which they have not been stated already.

  - The client TAP interface — Through a client TAP interface, the module receives all signals necessary to operate a TAP controller: One of each ScanInPort, ScanOutPort, and TMSPort, and at most one of TRSTPort and TCKPort.

  - The host TAP interface — Through a host TAP interface, the module defines signals for another module’s client TAP interface: One ScanInPort and/or one ScanOutPort, one ToTMSPort, and at least one of ToTRSTPort and ToTCKPort.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

ScanInterfaces with multiple CaptureEnPorts, UpdateEnPorts, ToCaptureEnPorts or ToUpdateEnPorts are not standard-compliant. Third-party IJTAG tools might reject the processing of ICL files with such ScanInterface specifications.

- DefaultLoadValue *load_value* ;

  An optional keyword-string pair that defines the value to be loaded into the single scan chain defined by a pair port names of portfunction type ScanInPort and ScanOutPort of a black-box module. The DefaultLoadValue property affects the behavior of the iScan command only. The iScan command is currently not supported.

- Chain *chain_name* ;

  A repeatable, optional keyword-string pair that declares the name of a scan chain of the module. All chain names must be unique within a ScanInterface. If the ScanInterface has only one ScanInPort/ScanOutPort pair, than Chain object is not required.

Examples

Example 1

The following example shows three equivalent description of an ICL module with a single scan-in / scan-out client scan interface. This is not a complete list, other variations are possible. Note that for the module tdr4_1, no ScanInterface statement is given. It is not necessary, since the module has exactly one client scan interface. However, you may provide a ScanInterface as shown for modules tdr4_2, and tdr4_3.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module tdr4_1 {
   ScanInPort       si;
   ScanOutPort      so       { Source R[0]; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;
   ScanRegister R[7:0] {
      ScanInSource     si;
      ResetValue       8'b0 ;
   }
}
Module tdr4_2 {
   ScanInPort       si;
   ScanOutPort      so       { Source R[0]; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;
   ScanInterface interface1 {
     Port en ;
     Port si; Port so;
   }
   ScanRegister R[7:0] {
      ScanInSource     si;
      ResetValue       8'b0 ;
   }
}
Module tdr4_3 {
   ScanInPort       si;
   ScanOutPort      so       { Source R[0]; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;
   ScanInterface interface2 {
     Port se ;
     Chain chain1 {
        Port si; Port so;
     }
   }
   ScanRegister R[7:0] {
      ScanInSource     si;
      ResetValue       8'b0 ;
   }
}
```

Example 2

The next example shows the port functions and both the client and host TAP interfaces of a typical TAP controller module.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module tap {
   TCKPort         tck;
   ScanInPort      tdi;
   ScanOutPort     tdo       {  Source IRMux; }
   TMSPort         tms;
   TRSTPort        trst;
   ToSelectPort    tdrEn     {  Source sel;  }
   ScanInPort      fromTdr;
   ToSelectPort    tapEn     {  Source IR[2]; }
   ToCaptureEnPort tce;
   ToShiftEnPort   tse;
   ToUpdateEnPort  tue;
   ScanInterface clientTAP {
     Port tdi; Port tdo;
     Port tms; Port trst;
   }
   ScanInterface hostTAP   {
     Port fromTdr1; Port tdrEn1;
     Port tce; Port tse; Port tue;
   }
   [...]
}
```

Example 3

The next example shows an example implementation of a SIB (Segment Insertion Bit) module. This module implements a scan chain switch by selecting either of two scan input ports. The module also generates an outgoing enable signal that is correlated to the second scan-in port.

```
Module sib {
   ScanInPort       si;
   ScanOutPort      so   { Source SIB; }
   ShiftEnPort      se;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   SelectPort       en;
   TCKPort          tck;
   ScanInPort       fso;
   ToSelectPort     ten  { Source And; }
   ScanInterface client {
     Port si; Port so;
     Port se; Port ce; Port ue; Port en;
   }
   ScanInterface host   {
     Port fso; Port ten;
   }
   ScanRegister SIB {
      ResetValue     1'b0;
      ScanInSource   Mux;
   }
   ScanMux Mux SelectedBy SIB {
      1'b0 : si;
      1'b1 : fso;
   }
   LogicSignal And { en,SIB == 2'b11;  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

Chain

ScanInPort

SelectPort

ShiftEnPort

CaptureEnPort

UpdateEnPort

ResetPort

TCKPort

TMSPort

TRSTPort

ScanOutPort

ToShiftEnPort

ToCaptureEnPort

ToUpdateEnPort

ToResetPort

ToTCKPort

ToTMSPort

ToTRSTPort

ScanRegister

ScanMux

LogicSignal

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ScanMux

Creates and names a scan multiplexer.

Usage

```
Module module_name {
  ScanMux mux_name SelectedBy mux_select_signal {
      Attribute att_name = att_value;
    mux_select_value : scan_source_name ;
  }
}
```

Description

The ScanMux statement creates and names a scan multiplexer.

IJTAG differentiates between a scan multiplexer, a data multiplexer, and a clock multiplexer. All three share the same syntax (other than the keyword), but their semantics differ. The scan multiplexer only switches between scan signals, the data multiplexer only switches between data signals, and the clock multiplexer only switches between clock signals. Do not mix any signal types for any of the multiplexers. You may source the select signal of the multiplexers only by data signals. Note that the parallel output of a scan register is also a data signal.

A common design structure is to have a ScanMux, with a 1-bit TDR at the output of the ScanMux, and the parallel output of the TDR connected to the select input of the mux. In this configuration, the TDR behind the mux is always on the scan path, independent of which scan input ports of the mux have been selected. Through this TDR, Tessent can select either of the scan input ports of the mux as needed.

Arguments

- *mux_name*

  A string that identifies the name of a scan multiplexer of the module. The name of the ScanMux must be unique among all port names, and among all the following objects: ScanRegister, DataRegister, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  Using this name, the tool can reference the outgoing signal of the multiplexer. The width of *mux_name* must match the width of the *scan_source_name*.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The possibility to attach attributes to multiplexers is not part of the IEEE 1687-2014 standard. It is an extension specific to Tessent Shell. You define the attribute name and the attribute value and any IJTAG software tool may ignore or reject it. Tessent Shell currently understands only one ScanMux attribute, which is exclude_codes_from_icl_verify. See Scan Register Integrity Test in the *Tessent IJTAG User's Manual* for details. All other attributes are ignored.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- *mux_select_value* : *scan_source_name*

  A repeatable string-string pair, connected by a colon symbol ':'. A ScanMux must have at least one mux_select_value - scan_source_name pair. The width of the *mux_select_value* must match the width of the *mux_select_signal*.

  The *mux_select_value* describes the value the select signal (*mux_select_signal*) of the mux must assume for the *mux_name* signal to connect to the *scan_source_name* signal. The *mux_select_value* may contain x-values. The collection of all specified *mux_select_value* parameters need not cover the entire number space. Any select value not listed causes the *mux_name* signal to assume x-values of the correct width.

  A mux_select_value may consist of multiple different select values connected by OR ("|"). The width of each individual select value must match the width of the mux_select_signal.

  Note

  If multiple *mux_select_values* have the same scan path length to a common source or have the same *scan_source_name*, the tool prioritizes these *mux_select_values* during IJTAG retargeting based on their order in the ICL file. This does not affect the optimality of the retargeting result but allows for a fine-grained control over otherwise equivalent scan paths.

  See "How to Model a ScanMux Selection Preference" in the *Tessent IJTAG User's Manual* for details.

Examples

The following shows usages of all the multiplexer types.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module chip2 {
   ScanInPort  siA ;
   ScanInPort  siB ;
   ScanOutPort so      { Source scanM ; }
   ShiftEnPort   se1 ;
   CaptureEnPort ce1 ;
   UpdateEnPort  ue1 ;
   ShiftEnPort   se2 ;
   CaptureEnPort ce2 ;
   UpdateEnPort  ue2 ;
   TCKPort tck ;
   ClockPort clkA ;
   ClockPort clkB ;
   ToClockPort clk { Source clockM ; }
   DataInPort   dselA ;
   DataInPort   dselB[2:0] ;
   DataOutPort do[3:0] { Source dataM[3:0] ; }
   ScanInterface cA12 {
      Port se1; Port ce1; Port ue1;
      Chain c1 { Port siA; Port so ; }
   }
   ScanInterface cB12 {
      Port se2; Port ce2; Port ue2;
      Chain c1 { Port siB; Port so ; }
   }
   ScanRegister regA[7:0] {
      ScanInSource siA ;
      ResetValue   8'b0 ;
   }
   ScanRegister regB[7:0] {
      ScanInSource siB ;
      ResetValue   8'b0 ;
   }
   ScanMux scanM SelectedBy dselA {
      1'b0 : regA[0] ;
      1'b1 : regB[0] ;
   }
   ClockMux clockM SelectedBy dselA {
      1'b0 : clkA ;
      1'b1 : clkB ;
   }
   DataMux dataM[3:0] SelectedBy dselB[2:0] {
      3'b000|3'b100  : regA[7:4] ;
      3'bx01         : regA[3:0] ;
      3'bx1x         : regB[5:2] ;
   }
}
```

Related Topics

DataMux

ClockMux

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ScanOutPort

Declares the name of a scan-out port of a Module.

Usage

Syntax 1

```
Module module_name {
  ScanOutPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ScanOutPort port_name {
    Source source_name ;
    Enable enable_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of a scan-out port of a Module.

A module may have zero or more scan-out ports. However, if a module has at least one ScanInPort defined, then there must be at least one ScanOutPort.

The scan-out port is used to execute a serial read (that is, not a parallel read operation). For a parallel read, use the DataOutPort port function instead.

If none of the parameters Source, Enable, or Attribute are used, you must use Syntax 1. However, in this case, the scan chain is broken at the ScanOutPort. Retargeting cannot use this port, and if no other solution can be found, reading and writing from the broken scan segment results in an error during PDL command retargeting.

Arguments

- *port_name*

  A required string that identifies the name of a scan-out port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ScanOutPort of the module that is also used for the respective ports of the design module.

- Source *source_name* ;

  A keyword-string pair that identifies the source of the scan-out data, like a ScanRegister or a ScanOutPort of another module instance. A ScanOutPort may have at most one Source statement. The Source statement is not needed for black box modules, which are currently

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

not supported by Tessent. If the source_name references a ScanRegister, it should reference the right most bit of the register. Further, the width of the Source must match the width of the ScanOutPort.

- Enable *enable_name* ;

  A keyword-string pair that identifies the enable signal of the scan-out port. The enable signal must become active in order to connect the scan-out port with the scan source. The activation and deactivation of the enable signal is taken care of by Tessent. Please see the OneHotScanGroup to learn more about the usage of the enable signal. A ScanOutPort may have at most one Enable statement.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin_attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following example shows the usage of the ScanOutPort and its Source statement. Observe that the width of the scan-out port is 2, hence the concatenation of the right most bits of both scan registers. Please see the discussion and examples of the OneHotScanGroup to learn more about the usage of the ScanOutPort's enable signal.

```
Module tdr2x {
   ScanInPort       si[1:0];
   ScanOutPort      so[1:0] { Source R0[0],R1[0]; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;
   ScanInterface chain2x {
     Port se ;
     Chain c0 { Port si[0]; Port so[0]; }
     Chain c1 { Port si[1]; Port so[1]; }
   }

   ScanRegister R0[9:0] {
      ScanInSource     si[0];
      ResetValue       10'b0 ;
   }   ScanRegister R1[4:0] {
      ScanInSource     si[1];
      ResetValue       5'b0 ;
   }
}
```

Related Topics

ScanInPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DataOutPort

ScanInterface

OneHotScanGroup

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ScanRegister

Defines and names a scan register of a certain width and with certain properties.

Usage

```
Module module_name {
  ScanRegister register_name {
    Attribute att_name = att_value;
    ScanInSource scan_in_source;
    CaptureSource capture_source;
    DefaultLoadValue default_load_value;
    ResetValue reset_value;
    RefEnum enum_name;
  }
}
```

Description

The statement ScanRegister defines and names a scan register of a certain width and with certain properties. These properties, like the reset value, are described through parameters of the scan register.

Arguments

- *register_name*

  A required string that identifies the name of a the scan register of the module. The name of the ScanRegister must be unique among all port names, and among all the following objects: DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

- ScanInSource *scan_in_source*

  A required keyword-string pair that identifies the signal that is providing the scan data to the left most bit of the scan chain. If *scan_in_source* references another scan register, it must reference the right-most bit of that scan register (that is, the ‘scan out bit’).

  For ranged scan registers; that is, scan registers that have a width > 1, the scan register always shifts from left to right, independently if the left most index is larger than the right most index, or vice-versa.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- CaptureSource *capture_source*

  An optional keyword-string pair that identifies the signal from which the ScanRegister captures data in response to the CaptureEnPort and TCKPort signal. If used, the width of *capture_source* must match the width of the register.

- DefaultLoadValue *default_load_value*

  An optional keyword-string pair that identifies the value that each scan register cell assumes if no user-defined value had been written to it beforehand and if no reset had been triggered. If used, the width of *default_load_value* must match the width of the register.

  When both the DefaultLoadValue parameter and the ResetValue parameter exist in the same Scan Register statement, the non-x bit values of *reset_value* must match those of *default_load_value*.

- ResetValue *reset_value*

  An optional keyword-string pair that identifies the value that each scan register cell assumes as a response to an active reset signal of the module. In more detail, it is the update stage of the scan register that assumes the declared reset value. If used, the width of *reset_value* must match the width of the register. If the ResetValue parameter is not used, the scan register does not respond to the module's incoming reset signal. However, if the module contains scan registers with a ResetValue parameter, the module must have only one ResetPort statement.

  When both the DefaultLoadValue parameter the ResetValue parameter exist in the same Scan Register statement, the non-x bit values of *reset_value* must match those of *default_load_value*.

- RefEnum *enum_name*

  An optional keyword-string pair that references the name of an table. Using the enumeration table, reading and writing to the scan register can use synonyms listed in the table, instead of bit strings. Understand that the enumeration table is not necessarily a complete list. It is still allowed to read or write any value, including those not listed in the enumeration table. A ScanRegister statement may have at most one RefEnum parameter. In addition, the width of the elements in the enumeration table must match the width of the scan register.

Additional Rules

A ScanRegister is considered to be in the active scan path if it responds to the ShiftEnPort signal associated to the ScanInterface through an active ScanInterface. It holds its data as long as the ShiftEnPort remains inactive (low) between the capture- and the update-cycle.

Any scan register bit that controls parts of a scan network (that is, the select line of a ScanMux, a SelectPort or ToSelectPort), or is part of a ScanInterface, anywhere in the hierarchy of the design, has a non-x reset value (that is, a reset value of 0 or 1).

If a scan register bit is the source of a data signal, the scan register must have an update stage that (or an equivalent implementation) which changes only in response to an active UpdateEnPort signal.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

It is important to understand the rules which determine what value gets loaded into the scan register bits. Assume no user-defined care bits have ever been written to any scan register bits. Now an iApply block accesses the ScanRegister. The scan register bits assume a value in this order of preference: The specified values required to satisfy the operations in the iApply block; the value DefaultLoadValue specifies, if is a 0 or 1; the value ResetValue specifies, if it is a 0 or 1; any scan register bits still unassigned a 0 or 1, are loaded with a 0.

If a scan register receives a reset signal, all scan register bits assume the value specified in *reset_value*. For the remaining x-values apply the sequence of events outlined in the paragraph above.

If a ScanRegister is on the active scan path (that is, a shift operation may happen), any scan register bits that are not required to satisfy the operations in the current iApply block remains unchanged from their current value.

Examples

Example 1

The following shows a complete example ICL module definition of a TAP controller. Observe that the two different ScanRegisters: ‘IR’ has a width of 3, whereas ‘bypass’ has a width of 1. ‘IR’ must have as ResetValue parameter defined, since it controls the ScanMux DRMux. Further, observe the usage of the name of the scan register in the expression of the LogicSignal ‘sel1’.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module tap {
   TCKPort         tck;
   ScanInPort      tdi;
   ScanOutPort     tdo       {  Source IRMux; }
   TMSPort         tms;
   TRSTPort        trst;
   ToSelectPort    tdrEn1    {  Source sel1;  }
   ScanInPort      fromTdr1;
   ToSelectPort    tapEn1    {  Source IR[2]; }
   ToCaptureEnPort tce;
   ToShiftEnPort   tse;
   ToUpdateEnPort  tue;
   ScanInterface clientTAP {
     Port tdi; Port tdo;
     Port tms; Port trst;
   }
   ScanInterface hostTAP   {
     Port fromTdr1; Port tdrEn1;
     Port tce; Port tse; Port tue;
   }
   Instance fsm_i Of fsm  {
      InputPort tck  = tck;
      InputPort tms  = tms;
      InputPort trst = trst;
    }
   ScanRegister IR[2:0] {
      CaptureSource  3'b001;
      ResetValue     3'b000;
      ScanInSource   tdi;
   }
   ScanRegister bypass {
      CaptureSource  1'b0;
      ScanInSource   tdi;
   }
   ScanMux IRMux SelectedBy fsm_i.irSel {
      1'b0 : DRMux;
      1'b1 : IR[0];
   }
   ScanMux DRMux SelectedBy IR[2:0] {
      3'bx00 : bypass;
      3'bx01 : fromTdr1;
   }
   LogicSignal sel1 {  IR[2:0] == 3'bx01; }
}
Module fsm {
   TCKPort        tck;
   TMSPort        tms;
   TRSTPort       trst;
   ToIRSelectPort irSel;
   ToResetPort    tlr;
}
```

Example 2

The following example shows an ICL module definition of a sensor where CaptureSource can be used to combine multiple DataInPorts to form the width of the ScanRegister.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module sensor {
   TCKPort        tck;
   TRSTPort       rst;
   ScanInPort     si;
   ScanOutPort    so;      {  Source  S_reg[0]; }
   CaptureEnPort  capture_en;
   ShiftEnPort    shift_en;
   UpdateEnPort   update_en;
   SelectPort     sel_en;
   DataInPort     sdata_in [2:0];
   DataInPort     sens_from;
   DataInPort     cntrl [1:0];
ScanRegister S_reg [5:0] {
   ResetValue     6'b000000;
   CaptureSource  sdata_in, sens_from, cntrl ;
   ScanInSource   si;
  }
}
```

Related Topics

ScanInPort

DataOutPort

iApply

DataRegister

ShiftEnPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SelectPort

Declares the name of the select input port of a Module's client ScanInterface.

Usage

```
Module module_name {
  SelectPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the select input port of a Module's client ScanInterface.

A module may have zero or more select ports. It is used to enable a scan interface of a module. When the select port is active (high), the scan interface is active. However, if the select port is inactive (low), the scan interface does not respond to any other control signal defined for this scan interface. The SelectPort statement for a non-handoff module is optional and implied of the module has a scan interface. The SelectPort is only required if the port is part of the ScanInterface statement. A SelectPort which is not part of a scan interface and which is not part of the scan network configuration logic is equivalent to a DataInPort.

Arguments

- *port_name*

  A required string that identifies the name of a select port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL SelectPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

In this example, the ShiftEnPort, CaptureEnPort, and UpdateEnPort ports are shared among both ScanInterface. The selection between the two interfaces is accomplished through the two SelectPort sel1 and sel2, respectively.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module chip2 {
   ScanInPort  siA1 ;
   ScanInPort  siA2 ;
   ScanOutPort soA1 { Source blockA.so1 ; }
   ScanOutPort soA2 { Source blockA.so2 ; }
   ScanInPort  siB1 ;
   ScanInPort  siB2 ;
   ScanOutPort soB1 { Source blockB.so1 ; }
   ScanOutPort soB2 { Source blockB.so2 ; }
   SelectPort   selA ;
   SelectPort   selB ;
   ShiftEnPort   se ;
   CaptureEnPort ce ;
   UpdateEnPort  ue ;
   TCKPort tck ;
   ScanInterface cA12 {
      Port selA;
      Chain c1 { Port siA1; Port soA1 ; }
      Chain c2 { Port siA2; Port soA2 ; }
   }
      ScanInterface cB12 {
      Port selB;
      Chain c1 { Port siB1; Port soB1 ; }
      Chain c2 { Port siB2; Port soB2 ; }
   }
   [...]
}
```

Related Topics

ScanInPort

DataInPort

ScanInterface

Chain

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ShiftEnPort

Declares the name of the shift enable port of a Module's client ScanInterface.

Usage

```
Module module_name {
  ShiftEnPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the shift enable port of a Module's client ScanInterface.

This port is used to enable the shift operation into a module's ScanRegister elements. A module may have zero or more shift enable ports. For an internal module; that is, not a hand-off module, a single shift enable port is implied if there are scan in/out ports in the module. In this case, a ShiftEnPort need not be declared. The ShiftEnPort must be declared if:

- The module is a hand-off module.
- There is more than one shift enable port in the ICL module definition. This may happen if there are multiple scan in/out ports defined that are operated independently.

The shift enable signal is an active high signal. A ShiftEnPort of an instance is sourced by a ToShiftEnPort. A ScanRegister object is expected to hold its scan data, as long as the shift enable port is low, independently on the number of clock cycles.

Arguments

- *port_name*

  A required string that identifies the name of a shift enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ShiftEnPort of the module that is also used for the respective ports of the design module.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin_attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

The following example shows an ICL module definition with a single scan-in / scan-out chain, therefore, no client ScanInterface is required. The shift enable port is named “se”. Observe that the body of the Module does not further use the shift enable port name, nor is its active-high value used or stated anywhere. Its connection to the scan chain is done implicitly. Please see the examples of the ScanInterface to learn how you can use multiple scan chains, potentially with multiple shift enable signals.

```
Module tdr2 {
   ScanInPort       si;
   ScanOutPort      so  { Source R[0]; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;
   ScanRegister R[7:0] {
      ScanInSource     si;
      ResetValue       8'b0 ;
      DefaultLoadValue 8'b0 ;
   }
}
```

Related Topics

CaptureEnPort

UpdateEnPort

ScanInPort

ScanOutPort

ScanInterface

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TCKPort

Declares the name of the test clock (TCK) input port of a Module.

Usage

```
Module module_name {
  TCKPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the test clock (TCK) input port of a Module.

The declaration of the TCKPort is optional for an internal module (that is, not for a hand-off module), even if the module contains scan registers. In this case, the test clock is implicitly assumed.

A module may have more than one TCKPort statement. In this case, all declared test clocks are considered equivalent and with the same polarity.

Non test clock ports are defined through the ClockPort statement.

Note that IJTAG has no means of defining the actual speed of a test clock. This is left to the software implementing the standard. In Tessent, you can use the add_clocks command to define clock properties, like the period or the off-value.

Arguments

- *port_name*

  A required string that identifies the name of the test clock port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotScanGroup, OneHotDataGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL TCKPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

ScanInterface

ScanRegister

ClockPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TMSPort

Declares the name of the TMS input port of a Module's client TAP ScanInterface.

Usage

```
Module module_name {
  TMSPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the TMS input port of a Module's client TAP ScanInterface.

If a module has a client TAP interface, the TMSPort declaration may not be omitted. If the module has a ToIRSelectPort, the module must also have exactly one TMSPort.

IJTAG supports top-level TAPs, as well as any number of embedded TAP controllers.

A TMSPort can be explicitly connected to various sorts of drivers by means of the InputPort binding in the instantiation of the module, causing the TMSPort either to be in its normal mode of operation or to be continuously driven with the value '1' (which causes the associated TAP controller to be held in "reset" state) or to be continuously driven with the value '0' (which causes the associated TAP controller to be held in "idle" state). If the driver has not been specified explicitly, then the TMSPort may be implicitly connected to a tms signal in the parent module. See chapter "How to Model Global Reset, Local Reset and Embedded TAPs" in the *Tessent IJTAG User's Manual* for the details on embedded TAPs and the different ways of connecting a TMSPort.

Arguments

- *port_name*

  A required string that identifies the name of the TMS port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias

  It is recommended to use a port name for the ICL TMSPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several icl_port attributes and icl_pin

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following shows a complete example ICL module definition of a TAP controller. Observe that the state machine is not described, there is only a generic module that defines the instruction register select signal. IJTAG has built-in the knowledge of a TAP's state machine.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module tap {
   TCKPort         tck;
   ScanInPort      tdi;
   ScanOutPort     tdo       {  Source IRMux; }
   TMSPort         tms;
   TRSTPort        trst;
   ToSelectPort    tdrEn1    {  Source sel1;  }
   ScanInPort      fromTdr1;
   ToSelectPort    tapEn1    {  Source IR[2]; }
   ToCaptureEnPort tce;
   ToShiftEnPort   tse;
   ToUpdateEnPort  tue;
   ScanInterface clientTAP {
     Port tdi; Port tdo;
     Port tms; Port trst;
   }
   ScanInterface hostTAP   {
     Port fromTdr1; Port tdrEn1;
     Port tce; Port tse; Port tue;
   }
   Instance fsm_i Of fsm  {
      InputPort tck  = tck;
      InputPort tms  = tms;
      InputPort trst = trst;
    }
   ScanRegister IR[2:0] {
      CaptureSource  3'b001;
      ResetValue     3'b000;
      ScanInSource   tdi;
   }

   ScanRegister bypass {
      CaptureSource  1'b0;
      ScanInSource   tdi;
   }   ScanMux IRMux SelectedBy fsm_i.irSel {
      1'b0 : DRMux;
      1'b1 : IR[0];
   }   ScanMux DRMux SelectedBy IR[2:0] {
      3'bx00 : bypass;
      3'bx01 : fromTdr1;
   }   LogicSignal sel1 {  IR[2:0] == 3'bx01; }
}Module fsm {
   TCKPort        tck;
   TMSPort        tms;
   TRSTPort       trst;
   ToIRSelectPort irSel;
   ToResetPort    tlr;
}
```

Related Topics

ScanInterface

TRSTPort

ToIRSelectPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ToCaptureEnPort

Declares the name of the to-capture enable output port of a Module's host ScanInterface.

Usage

Syntax 1

```
Module module_name {
  ToCaptureEnPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToCaptureEnPort port_name {
    Source source_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the to-capture enable output port of a Module's host ScanInterface.

This port is used to trigger the capture operation of another module's ScanRegister elements. It is an active high output signal. A module may have zero or more to-update enable ports.

If neither of the parameters Source or Attribute are used, you must use Syntax 1.

Arguments

- *port_name*

  A required string that identifies the name of a shift enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToCaptureEnPort of the module that is also used for the respective ports of the design module.

- Source *source_name* ;

  An optional keyword-string pair that identifies the source of the to-capture enable signal. A ToCaptureEnPort may have at most one Source statement. Further, the width of the Source must match the width of the ToCaptureEnPort.

  The Source may point to a CaptureEnPort of the same module, a ToCaptureEnPort of an instance or a DataMux which has a capture signal at one of its inputs and the constant value 1'b0 at the other input. If the DataMux is configured such that it selects the constant value 1'b0, the capture enable pulse is assumed to be suppressed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The suppressing of the enable signal described above is currently not supported in Tessent.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several icl_port attributes and icl_pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following shows a complete example ICL module definition of a TAP controller. Observe that the state machine is not described, there is only a generic module that defines the instruction register select signal. IJTAG has built-in knowledge of a TAP's state machine. Also observe the usage of the various To*EnPort statements.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module tap {
   TCKPort         tck;
   ScanInPort      tdi;
   ScanOutPort     tdo       {  Source IRMux; }
   TMSPort         tms;
   TRSTPort        trst;
   ToSelectPort    tdrEn1    {  Source sel1;  }
   ScanInPort      fromTdr1;
   ToSelectPort    tapEn1    {  Source IR[2]; }
   ToCaptureEnPort tce;
   ToShiftEnPort   tse;
   ToUpdateEnPort  tue;   ScanInterface clientTAP {
     Port tdi; Port tdo;
     Port tms; Port trst;
   }
   ScanInterface hostTAP   {
     Port fromTdr1; Port tdrEn1;
     Port tce; Port tse; Port tue;
   }   Instance fsm_i Of fsm  {
      InputPort tck  = tck;
      InputPort tms  = tms;
      InputPort trst = trst;
    }   ScanRegister IR[2:0] {
      CaptureSource  3'b001;
      ResetValue     3'b000;
      ScanInSource   tdi;
   }

   ScanRegister bypass {
      CaptureSource  1'b0;
      ScanInSource   tdi;
   }   ScanMux IRMux SelectedBy fsm_i.irSel {
      1'b0 : DRMux;
      1'b1 : IR[0];
   }   ScanMux DRMux SelectedBy IR[2:0] {
      3'bx00 : bypass;
      3'bx01 : fromTdr1;
   }   LogicSignal sel1 {  IR[2:0] == 3'bx01; }
}Module fsm {
   TCKPort        tck;
   TMSPort        tms;
   TRSTPort       trst;
   ToIRSelectPort irSel;
   ToResetPort    tlr;
}
```

Related Topics

TMSPort

ScanInterface

ToIRSelectPort

ToTCKPort

ToSelectPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ToUpdateEnPort

ToShiftEnPort

ToResetPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ToClockPort

Declares the name of a functional to-clock output port of a Module.

Usage

Syntax 1

```
Module module_name {
  ToClockPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToClockPort port_name {
    Source source_name ;
    FreqMultiplier pos_int ;
    FreqDivider pos_int ;
    DifferentialInvOf other_clock_port_name ;
    Period pos_int unit ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of a functional to-clock output port of a Module (that is, a clock that is not a test clock).

The test clock is defined through the ToTCKPort statement. A module may have zero or more to-clock ports.

A ToClockPort serves as the source of a functional clock to another module. It has three principal features:

- Modify an existing incoming functional clock with respect to its frequency by multiplication and division of the period
- Serve as an oscillator clock generation with a defined outgoing clock frequency
- Convert from differential to single-ended clock signals and vice versa.

If none of the parameters are used, you must use Syntax 1.

Arguments

- *port_name*

  A required string that identifies the name of a to-clock port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

It is recommended to use a port name for the ICL ToClockPort of the module that is also used for the respective ports of the module in the design.

- Source *source_name*

  An optional keyword-string pair that identifies the source of the to-clock signal. A ToClockPort may have at most one Source statement. The width of the Source must match the width of the ToClockPort. The usage of Source is mutually exclusive with the usage of Period as well as with the usage of DifferentialInvOf within the same ToClockPort statement.

- FreqMultiplier *pos_int*

  An optional keyword-positive integer number pair that identifies the multiplication factor of the to-clock outgoing clock signal with respect to an incoming clock signal of the module, referenced by Source. A ToClockPort may have at most one FreqMultiplier statement. The usage of FreqMultiplier is mutually exclusive with the usage of Period as well as with the usage of DifferentialInvOf within the same ToClockPort statement.

- FreqDivider *pos_int*

  An optional keyword-positive integer number pair that identifies the division factor of the to-clock outgoing clock signal with respect to an incoming clock signal of the module, referenced by Source. A ToClockPort may have at most one FreqDivider statement. The usage of FreqDivider is mutually exclusive with the usage of Period as well as with the usage of DifferentialInvOf within the same ToClockPort statement.

- DifferentialInvOf *other_clock_port_name*

  An optional keyword-string pair. Differential functional clocks are declared through the DifferentialInvOf keyword. The *other_clock_port_name* string references another ClockPort *port_name* of the same module. The usage of DifferentialInvOf is mutually exclusive from the usage of Source, FreqMultiplier, FreqDivider, and Period, leaving the DifferentialInvOf the only parameter other than the Attribute within a ToClockPort statement.

- Period *pos_int unit*

  An optional keyword-positive integer number pair, followed by an optional unit string. If used, the unit string is exactly one of these time units: s, ms, us, ns, ps. If unit is not used, the default is “ns”.

  Using the Period parameter, you can create a module that does not require an incoming clock signal to generate an outgoing clock signal of a specified period. This module is like an oscillator.

  The usage of Period is mutually exclusive from the usage of Source, FreqMultiplier, FreqDivider, and DifferentialInvOf, leaving the Period as the only parameter other than the Attribute within a ToClockPort statement.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several icl_port attributes and icl_pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following example represents a programmable clock generation logic that selectively uses one of two incoming reference clocks and manipulates either with a selectable set of modifiers.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module chip_clock {
   ScanInPort    si ;
   ScanOutPort   so { Source clockGen_inst.so ; }
   ShiftEnPort   se ;
   CaptureEnPort ce ;
   UpdateEnPort  ue ;
   TCKPort   tck ;
   ClockPort refClkA ;
   ClockPort refClkB ;
   DataInPort  din[7:0] ;
   DataOutPort dout[7:0] { Source inst_inst.out ; }
   Instance clockGen_inst Of clockGen {
      InputPort si = si ;
      InputPort refClkA = refClkA ;
      InputPort refClkB = refClkB ;
      Parameter sclkMult = 1 ;
      Parameter sclkDiv  = 3 ;
   }
   Instance inst_inst Of instrument {
      InputPort clk = clockGen_inst.sclk ;
      InputPort in  = din ;
   }
}
Module clockGen {
   ScanInPort    si ;
   ScanOutPort   so { Source clkSelReg[0] ; }
   ClockPort refClkA ;
   ClockPort refClkB ;
   Parameter sclkMult = 1 ;
   Parameter sclkDiv  = 1 ;
   ToClockPort sclk {
     FreqMultiplier $sclkMult ;
     FreqDivider $sclkDiv ;
     Source refSelM ;
   }
   ScanRegister clkSelReg[2:0] {
      ScanInSource si ;
      ResetValue 3'b100 ;
   }
   ClockMux refSelM SelectedBy clkSelReg[2] {
      1'b1 : refClkA ;
      1'b0 : refClkB ;
   }
}
   Module instrument {
   DataInPort  in[7:0];
   DataOutPort out[7:0];
   ClockPort   clk;

   Alias enable      = in[7]           { RefEnum YesNo;    }
   Alias mode[3:0]   = in[6:5],in[3:2] { RefEnum Modes;    }
   Alias p1[2:0]     = in[4],in[1:0];
   Alias go          = out[0]          { RefEnum PassFail; }
   Alias done        = out[1]          { RefEnum YesNo;    }
   Alias count[5:0]  = out[7:2];   Enum PassFail {
      Pass = 1'b1;
      Fail = 1'b0;
   }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
        Enum YesNo {
           Yes  = 1'b1;
           No   = 1'b0;
        }
        Enum Modes {
           red   = 4'b0011;
           blue  = 4'b1000;
           green = 4'b0100;
        }
     }
```

Related Topics

ClockPort

TCKPort

ToTCKPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ToIRSelectPort

Declares the name of the to-IR Select port of a Module.

Usage

Syntax 1

```
Module module_name {
  ToIRSelectPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToIRSelectPort port_name {
    Source source_name ;
  Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the to-IR Select port of a Module.

This signal is used to select between the instruction register of a TAP controller and another scan register, usually a data register. The to-IR Select signal is always associated with the TMS signal, and if present, with the TRST signal.

If neither of the parameters Source or Attribute are used, you must use Syntax 1.

Arguments

- *port_name*

  A required string that identifies the name of a to-IR Select port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToIRSelectPort of the module that is also used for the respective ports of the design module.

- Source *source_name*

  An optional keyword-string pair that identifies the source of the to-IR select signal. A ToIRSelectPort may have at most one Source statement. The Source may point to an IRSelectPort of the same module or a ToIRSelectPort of an instance.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several icl_port attributes and icl_pin

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

TMSPort

TRSTPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ToResetPort

Declares the name of the to-reset output port of a Module's host ScanInterface.

Usage

Syntax 1

```
Module module_name {
  ToResetPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToResetPort port_name {
    Attribute att_name = att_value ;
    Source source_name ;
    ActivePolarity 0 | 1;
  }
}
```

Description

A port function that declares the name of the to-reset output port of a Module's host ScanInterface.

The to-reset port is used as the source of the ResetPort of another module. You have the option to chose the active polarity of the to-reset port.

It is important to differentiate between the active polarity, 0 or 1, and when a to-reset signal becomes active / inactive. For example, it is possible that the to-reset signal's active polarity is 0, but it is enabled if the module's input reset port turns active high (1).

If none of the parameters Source, Attribute, or ActivePolarity are used, you must use Syntax 1.

Arguments

- *port_name*

  A required string that identifies the name of a to-reset port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotScanGroup, OneHotDataGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToResetPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several icl_portattributes and icl_pin

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

- ActivePolarity 0 | 1

  An optional keyword-value pair, that defines if the to-reset port of the module is considered active low (value = 0), or active high (value =1). A module may have at most one ActivePolarity statement. If the ActivePolarity statement is not given, the active polarity of the to-reset port is assumed to be the 1 (high).

  The activity/inactivity of the to-reset signal is given through the single ResetPort statement of the module, or, if the module does not have a ResetPort or more than one, through the global reset signal. In other words, if the reset signal is active, the to-reset signal is active as well, however, the ActivePolarity may be different.

- Source source_name

  An optional keyword-string pair that identifies the source of the to-reset signal. A ToResetPort may have at most one Source statement. However, if the module has more than one ResetPort statement, then the Source must be specified. Further, the width of the Source must match the width of the ToResetPort.

  The Source property can point to various sorts of drivers within the module, causing the ToResetPort to become active under different conditions (synchronous reset, asynchronous reset, enforced local reset and so on). If the Source has not been specified, then the ToResetPort may be implicitly connected to a reset signal in the current module. See chapter "How to Model Global Reset, Local Reset and Embedded TAPs" in the *Tessent IJTAG User's Manual* for the details on the reset functionality and the different ways of connecting a ToResetPort.

Examples

The following shows an example ICL netlist that switches the active polarity of the reset signal between instances. Observe that the outgoing active polarity of res.trp is low, so is the incoming polarity of the module instance inst_inst.re. It is an error to connect a ToResetPort to a ResetPort with different polarities.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module chip_reset {
   ScanInPort  si ;
   ScanOutPort so { Source inst_inst.so ; }   ShiftEnPort   se ;
   CaptureEnPort ce ;
   UpdateEnPort  ue ;
   SelectPort    selC ;   ResetPort re { ActivePolarity 1 ; }
   TCKPort tck ;   Instance res_inst Of res {
      InputPort re   = re ;
   }   Instance inst_inst Of inst {
      InputPort si  = si ;
      InputPort se  = se;
      InputPort ce  = ce ;
      InputPort ue  = ue ;
      InputPort sel = selC ;
      InputPort re  = res_inst.trp ;
   }
}
Module res {
   ResetPort re ;
   ToResetPort trp { ActivePolarity 0 ; }
}
Module inst {
   ScanInPort  si ;
   ScanOutPort so { Source R[0] ; }
   ShiftEnPort   se  ;
   CaptureEnPort ce  ;
   UpdateEnPort  ue  ;
   SelectPort    sel ;
   TCKPort tck ;

   ResetPort re { ActivePolarity 0; }
   ScanRegister R[7:0] {
      ScanInSource si ;
      ResetValue 8'b0 ;
   }
}
```

Related Topics

ScanInterface

ToTCKPort

ToUpdateEnPort

ToSelectPort

ToCaptureEnPort

ToShiftEnPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ToSelectPort

Declares the name of the to-select port of a Module's host ScanInterface.

Usage

Syntax 1

```
Module module_name {
  ToSelectPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToSelectPort port_name {
    Source source_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the to-select port of a Module's host ScanInterface.

This port is typically used to enable the scan operation in another module. It is an active high output signal. A module may have none, one, or several to-select ports.

Arguments

- *port_name*

  A required string that identifies the name of a to-select port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToSelectPort of the module that is also used for the respective ports of the design module.

- Source *source_name*

  An optional keyword-string pair that identifies the source of the to-select signal. The width of the Source must match the width of the ToUpdateEnPort.

  An optional keyword-string pair that identifies the source of the to-select signal. The width of the Source must match the width of the ToUpdateEnPort. If the to-select diagonal fans out into a ScanInterface, which is referenced by an AllowBroadcastingOnScanInterface property, anywhere in the hierarchy, then the Source statement is required.

  Note

  Currently Tessent does not support any broadcasting.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several icl_port attributes and icl_pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following shows an example ICL netlist that switches the active polarity of the reset signal between instances. Observe that the outgoing active polarity of res.trp is low, so is the incoming polarity of the module instance inst_inst.re. Further, the SelectPort signal for the instance inst_inst is computed based on the top level DataIn signal din.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module chip {
   ScanInPort  si ;
   ScanOutPort so { Source inst_inst.so ; }
   ShiftEnPort   se ;
   CaptureEnPort ce ;
   UpdateEnPort  ue ;
   DataInPort din ;
   ResetPort re { ActivePolarity 1 ; }
   TCKPort tck ;
   Instance res_inst Of res {
      InputPort gres = din ;
      InputPort re   = re ;
   }   Instance inst_inst Of inst {
      InputPort si  = si ;
      InputPort se  = se;
      InputPort ce  = ce ;
      InputPort ue  = ue ;
      InputPort sel = res_inst.tsel ;
      InputPort re  = res_inst.trp ;
   }
}
Module res {
   ResetPort    re ;
   DataInPort   gres ;
   ToResetPort  trp  { ActivePolarity 0 ; }
   ToSelectPort tsel { Source gres; }
}
Module inst {
   ScanInPort  si ;
   ScanOutPort so { Source R[0] ; }
   ShiftEnPort   se ;
   CaptureEnPort ce ;
   UpdateEnPort  ue ;
   SelectPort    sel ;
   TCKPort tck ;

   ResetPort re { ActivePolarity 0; }
   ScanRegister R[7:0] {
      ScanInSource si ;
      ResetValue 8'b0 ;
   }
}
```

Related Topics

ToShiftEnPort

ToUpdateEnPort

ToResetPort

ToCaptureEnPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ToShiftEnPort

Declares the name of the to-shift enable port of a Module's host ScanInterface.

Usage

Syntax 1

```
Module module_name {
  ToShiftEnPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToShiftEnPort port_name {
    Source source_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the to-shift enable port of a Module's host ScanInterface.

This port is used to enable the shift operation of another module's ScanRegister elements. It is an active high output signal. A module may have zero or more to-shift enable ports.

If neither of the parameters Source or Attribute are used, you must use Syntax 1.

Arguments

- *port_name*

  A required string that identifies the name of a shift enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToShiftEnPort of the module that is also used for the respective ports of the design module.

- Source *source_name* ;

  An optional keyword-string pair that identifies the source of the to-shift enable signal. A ToShiftEnPort may have at most one Source statement. Further, the width of the Source must match the width of the ToShiftEnPort.

  A to-shift enable signal pointing to a shift enable signal may evaluate to a constant 0 or a constant 1. If it evaluates to a constant 0, the scan network controlled by the to-shift enable signal is considered disabled. No read or write operation into this scan chain segment may occur. If it evaluates to a constant 1, it is considered an error.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The features associated with the Source specification are currently not supported. Tessent currently ignores the Source specification..

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several icl_port attributes and icl_pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following shows a complete example ICL module definition of a TAP controller. Observe that the state machine is not described, there is only a generic module that defines the instruction register select signal. IJTAG has built-in the knowledge of a TAP's state machine. Also observe the usage of the various To*EnPort statements.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module tap {
   TCKPort         tck;
   ScanInPort      tdi;
   ScanOutPort     tdo       {  Source IRMux; }
   TMSPort         tms;
   TRSTPort        trst;
   ToSelectPort    tdrEn1    {  Source sel1;  }
   ScanInPort      fromTdr1;
   ToSelectPort    tapEn1    {  Source IR[2]; }
   ToCaptureEnPort tce;
   ToShiftEnPort   tse;
   ToUpdateEnPort  tue;   ScanInterface clientTAP {
     Port tdi; Port tdo;
     Port tms; Port trst;
   }   ScanInterface hostTAP   {
     Port fromTdr1; Port tdrEn1;
     Port tce; Port tse; Port tue;
   }   Instance fsm_i Of fsm  {
      InputPort tck  = tck;
      InputPort tms  = tms;
      InputPort trst = trst;
    }   ScanRegister IR[2:0] {
      CaptureSource  3'b001;
      ResetValue     3'b000;
      ScanInSource   tdi;
   }

   ScanRegister bypass {
      CaptureSource  1'b0;
      ScanInSource   tdi;
   }   ScanMux IRMux SelectedBy fsm_i.irSel {
      1'b0 : DRMux;
      1'b1 : IR[0];
   }   ScanMux DRMux SelectedBy IR[2:0] {
      3'bx00 : bypass;
      3'bx01 : fromTdr1;
   }   LogicSignal sel1 {  IR[2:0] == 3'bx01; }
}Module fsm {
   TCKPort        tck;
   TMSPort        tms;
   TRSTPort       trst;
   ToIRSelectPort irSel;
   ToResetPort    tlr;
}
```

Related Topics

TMSPort

ScanInterface

ToIRSelectPort

ToTCKPort

ToSelectPort

ToCaptureEnPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ToUpdateEnPort

ToResetPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ToTCKPort

Declares the name of a test to-test clock output port of a Module.

Usage

Syntax 1

```
Module module_name {
  ToTCKPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToTCKPort port_name {
    Source source_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of a test to-test clock output port of a Module (that is, a clock that is not a functional clock).

The test clock is defined through the ToClockPort statement. A module may have zero or more to-test clock ports. If a module has several to-test clock outputs, all clocks are considered equivalent in period and polarity.

Unlike the ToClockPort statement, there are no modifiers in the ToTCKPort statement. The outgoing to-test clock signal is just a pass-through of the incoming test clock.

If no parameter is used, you must use Syntax 1.

Arguments

- *port_name*

  A required string that identifies the name of a to-clock port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToClockPort of the module that is also used for the respective ports of the module in the design.

- Source *source_name*

  An optional keyword-string pair that identifies the source of the to_tck signal. A ToTCKPort may have at most one Source statement. The width of the Source must match the width of the ToTCKPort. The Source property is not described in the IEEE 1687-2014 standard.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several icl_port attributes and icl_pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following example modifies the example given in ToClockPort. Here, a TCKPort and a ToTCKPort port was added to the module. Observe that ToTCKPort is just passing through the test clock. No manipulation of the test clock signal can happen.

```
Module clockGen {
   ScanInPort    si ;
   ScanOutPort   so { Source clkSelReg[0] ; }
   TCKPort   tck ;
   ClockPort refClkA ;
   ClockPort refClkB ;
   Parameter sclkMult = 1 ;
   Parameter sclkDiv  = 1 ;
   ToTCKPort   ttck ;
   ToClockPort sclk {
     FreqMultiplier $sclkMult ;
     FreqDivider $sclkDiv ;
     Source refSelM ;
   }
   ScanRegister clkSelReg[2:0] {
      ScanInSource si ;
      ResetValue 3'b100 ;
   }
   ClockMux refSelM SelectedBy clkSelReg[2] {
      1'b1 : refClkA ;
      1'b0 : refClkB ;
   }
}
```

Related Topics

ClockPort

TCKPort

ToClockPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ToTMSPort

Declares the name of the to-TMS port of a Module's host TAP ScanInterface.

Usage

Syntax 1

```
Module module_name {
  ToTMSPort port_name ;
  }
}
```

Syntax 2

```
Module module_name {
  ToTMSPort port_name {
    Source source_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the to-TMS port of a Module's host TAP ScanInterface. This port is typically used to provide a TMS signal to an embedded TAP controller module or to module that contains an embedded TAP controller.

If Source is used, the outgoing to-TMS signal is considered gated.

If neither of the parameters Source or Attribute are used, you must use Syntax 1.

Arguments

- *port_name*

  A required string that identifies the name of a to-TMS port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToTMSPort of the module that is also used for the respective ports of the design module.

  Any module that has a ToTMSPort must also have at least one TMSPort.

- Source *source_name*

  An optional keyword-string pair that identifies the logic signal that is considered gating the to-TMS signal of the module. A ToTMSPort may have at most one Source statement. Source is required if the module has more than one TMSPort. If the module has only one TMSPort, Source is optional and if not provided, ToTMSPort then refers to the TMSPort.

  The Source property can point to various sorts of drivers within the module, causing the ToTMSPort either to be in its normal mode of operation or to be continuously driven with

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

the value '1' (which causes the downstream embedded TAP controllers to be held in "reset" state) or to be continuously driven with the value '0' (which causes the downstream embedded TAP controller to be held in "idle" state). If the Source has not been specified explicitly, then the ToTMSPort may be implicitly connected to a tms signal in the current module. See chapter "How to Model Global Reset, Local Reset and Embedded TAPs" in the *Tessent IJTAG User's Manual* for the details on embedded TAPs and the different ways of connecting a ToTMSPort.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several icl_port attributes and icl_pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

TMSPort

ToTCKPort

ToTRSTPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ToTRSTPort

Declares the name of the to-TRST port of a Module's host TAP ScanInterface.

Usage

Syntax 1

```
Module module_name {
  ToTRSTPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToTRSTPort port_name {
    Source source_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the to-TRST port of a Module's host TAP ScanInterface. This port is used to distribute the TRST signal throughout the design. The to-TRST outgoing signal is always active low (0).

A module may have zero or more to-TRST ports. If a module has several to-TRST outputs, all to-TRST signals are considered equivalent.

If Source is used, the outgoing to-TRST signal is considered gated.

If neither of the parameters Source or Attribute are used, you must use Syntax 1.

Arguments

- *port_name*

  A required string that identifies the name of a to-TMS port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToTRSTPort of the module that is also used for the respective ports of the design module.

- Source *source_name*

  An optional keyword-string pair that identifies the logic signal that is considered gating the to-TRST signal of the module. A ToTRSTPort may have at most one Source statement, and the width of the Source must match the width of the ToTRSTPort. Source is required if the module has more than one TRSTPort. If the module has only one TRSTPort, Source is optional and if not provided, ToTRSTPort then refers to the TRSTPort. Further, if the module does not have a TRSTPort, the ToTRSTPort refers to the global TRST signal.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The Source property can point to various sorts of drivers within the module, causing the ToTRSTPort to become active under different conditions (asynchronous global reset, enforced local reset and so on). If the Source has not been specified, then the ToTRSTPort may be implicitly connected to a trst signal in the current module. See chapter "How to Model Global Reset, Local Reset and Embedded TAPs" in the *Tessent IJTAG User's Manual* for the details on the reset functionality and the different ways of connecting a ToTRSTPort.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several icl_port attributes and icl_pin as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

TMSPort

ToTCKPort

ToTMSPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ToUpdateEnPort

Declares the name of the to-update enable port of a Module's host ScanInterface.

Usage

Syntax 1

```
Module module_name {
  ToUpdateEnPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToUpdateEnPort port_name {
    Source source_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the to-update enable port of a Module's host ScanInterface.

This port is used to trigger the update operation of another module's ScanRegister elements. It is an active high output signal. A module may have zero or more to-update enable ports.

If neither of the parameters Source or Attribute are used, you must use Syntax 1.

Arguments

- *port_name*

  A required string that identifies the name of a shift enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToUpdateEnPort of the module that is also used for the respective ports of the design module.

- Source *source_name* ;

  An optional keyword-string pair that identifies the source of the to-update enable signal. A ToUpdateEnPort may have at most one Source statement. Further, the width of the Source must match the width of the ToUpdateEnPort.

  The Source may point to an UpdateEnPort of the same module, a ToUpdateEnPort of an instance or a DataMux which has an update signal at one of its inputs and the constant value 1'b0 at the other input. If the DataMux is configured such that it selects the constant value 1'b0, the update enable pulse is assumed to be suppressed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

The suppressing of the enable signal described above is currently not supported in Tessent.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several icl_port attributes and icl_pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following shows a complete example ICL module definition of a TAP controller. Observe that the state machine is not described, there is only a generic module that defines the instruction register select signal. IJTAG has built-in the knowledge of a TAP's state machine. Also observe the usage of the various To*EnPort statements.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
Module tap {
   TCKPort         tck;
   ScanInPort      tdi;
   ScanOutPort     tdo       {  Source IRMux; }
   TMSPort         tms;
   TRSTPort        trst;
   ToSelectPort    tdrEn1    {  Source sel1;  }
   ScanInPort      fromTdr1;
   ToSelectPort    tapEn1    {  Source IR[2]; }
   ToCaptureEnPort tce;
   ToShiftEnPort   tse;
   ToUpdateEnPort  tue;   ScanInterface clientTAP {
     Port tdi; Port tdo;
     Port tms; Port trst;
   }
   ScanInterface hostTAP   {
     Port fromTdr1; Port tdrEn1;
     Port tce; Port tse; Port tue;
   }   Instance fsm_i Of fsm  {
      InputPort tck  = tck;
      InputPort tms  = tms;
      InputPort trst = trst;
    }   ScanRegister IR[2:0] {
      CaptureSource  3'b001;
      ResetValue     3'b000;
      ScanInSource   tdi;
   }

   ScanRegister bypass {
      CaptureSource  1'b0;
      ScanInSource   tdi;
   }   ScanMux IRMux SelectedBy fsm_i.irSel {
      1'b0 : DRMux;
      1'b1 : IR[0];
   }   ScanMux DRMux SelectedBy IR[2:0] {
      3'bx00 : bypass;
      3'bx01 : fromTdr1;
   }   LogicSignal sel1 {  IR[2:0] == 3'bx01; }
}Module fsm {
   TCKPort        tck;
   TMSPort        tms;
   TRSTPort       trst;
   ToIRSelectPort irSel;
   ToResetPort    tlr;
}
```

Related Topics

TMSPort

ScanInterface

ToIRSelectPort

ToTCKPort

ToSelectPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ToCaptureEnPort

ToShiftEnPort

ToResetPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TRSTPort

Declares the name of a TAP reset input port of a Module's client TAP ScanInterface.

Usage

```
Module module_name {
  TRSTPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of a TAP reset input port of a Module's client TAP ScanInterface. The active polarity of the port is low.

A TRSTPort can be explicitly connected to various sorts of drivers by means of the InputPort binding in the instantiation of the module, causing the TRSTPort to become active under different conditions (asynchronous global reset, enforced local reset and so on). If the driver has not been specified explicitly, then the TRSTPort may be implicitly connected to a trst signal in the parent module. See chapter "How to Model Global Reset, Local Reset and Embedded TAPs" in the *Tessent IJTAG User's Manual* for the details on the reset functionality and the different ways of connecting a TRSTPort.

Arguments

- *port_name*

  A required string that identifies the name of the TRST port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL TRSTPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several icl_port attributes and icl_pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

ResetPort

TMSPort

ScanInterface

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

UpdateEnPort

Declares the name of the update enable port of a Module's client ScanInterface.

Usage

```
Module module_name {
  UpdateEnPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the update enable port of a Module's client ScanInterface.

This port is used to control the update operation of a module's ScanRegister elements. A module may have zero or more capture enable ports. A module may have zero or more update enable ports. For an internal module; that is, not a hand-off module, a single update enable port is implied if there are scan in/out ports in the module. In this case, an UpdateEnPort need not be declared. The UpdateEnPort must be declared if:

- The module is a hand-off module.
- There is more than one update enable port in the ICL module definition. This may happen if there are multiple scan in/out ports defined that are operated independently.

The update enable signal is an active high signal. An UpdateEnPort of an instance is sourced by an UpdateEnPort of the parent module, a ToUpdateEnPort of another instance or a DataMux which has an update signal at one of its inputs and the constant value 1'b0 at the other input. If the DataMux is configured such that it selects the constant value 1'b0, the update enable pulse is assumed to be suppressed. An UpdateEnPort of an instance is sourced by an UpdateEnPort of the parent module, a ToUpdateEnPort of another instance or a DataMux which has an update signal at one of its inputs and the constant value 1'b0 at the other input. If the DataMux is configured such that it selects the constant value 1'b0, the update enable pulse is assumed to be suppressed.

Note

The suppressing of the enable signal described above is currently not supported in Tessent.

Arguments

- *port_name*

  A required string that identifies the name of an update enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

It is recommended to use a port name for the ICL UpdateEnPort of the module that is also used for the respective ports of the module in the design.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Examples

The following example shows an ICL module definition with a single scan-in / scan-out chain, therefore, no client ScanInterface is required. The update enable port is named “ue”. Observe that the body of the Module does not further use the update enable port name, nor is its active-high value used or stated anywhere. Its connection to the scan chain is done implicitly. Please see the examples of the ScanInterface to learn how you can use multiple scan chains, potentially with multiple update enable signals.

```
Module tdr2 {
   ScanInPort       si;
   ScanOutPort      so  { Source R[0]; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;
   ScanRegister R[7:0] {
      ScanInSource     si;
      ResetValue       8'b0 ;
      DefaultLoadValue 8'b0 ;
   }
}
```

Related Topics

ShiftEnPort

CaptureEnPort

ScanInPort

ScanOutPort

ScanInterface

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

UseNameSpace

References all subsequent modules from a specific namespace defined earlier through the NameSpace statement.

Note

Currently Tessent does not support the UseNameSpace statement.

Related Topics

NameSpace

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

WriteEnPort

Declares the name of the write enable input port of a Module.

Usage

```
Module module_name {
  WriteEnPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the write enable input port of a Module. It is used if the module contains addressable instances or data registers.

This port serves as the write enable to the addressable instance and data registers.

Arguments

- *port_name*

  A required string that identifies the name of a write enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ScanInPort of the module that is also used for the respective ports of the module in the design.

  The source of the write enable port name signal may only be a write enable port of the parent module, or scan register bits of a ScanRegister statement. In both cases, this connection must be direct, and may not include any logic signals.

  A module may have at most one WriteEnPort statement. A module must have a WriteEnPort statement, if the module contains writable, addressable data registers or writable, addressable instances.

- Attribute *att_name* = *att_value* ;

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several icl_port attributes and icl_pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

Instance

ReadEnPort

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DataRegister

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 17
Adding Custom Plugin Functionality

This chapter describes how you can augment the functionality of Tessent Shell with your own Tcl packages, commands and metadata definitions. It explains the concept of a plugin directory that you populate and how to get it auto loaded when Tessent Shell boots up.

The chapter also describes the best practice method to minimize the amount of code that is actually parsed at boot time and shows you how to have your code loaded on demand the first time your custom commands are used.

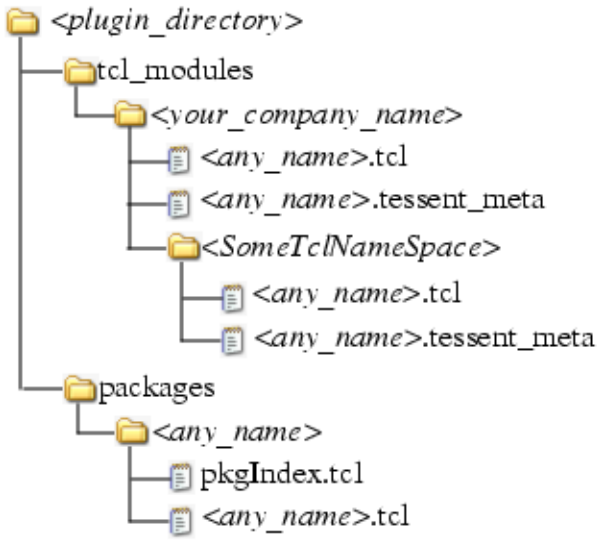
Finally, it describes how to create a proc called “u” to enable you to quickly reload the content of your plugin while you are developing and debugging it. The “u” proc is illustrated in Example 1 for tcl_modules and in Example 2 for packages.

plugin directory 4981

plugin directory

A directory that contains special sub directories with Tcl and metadata files to be auto loaded when Tessent Shell boots up.

Summary



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Description

A directory that groups two subdirectories containing tcl code and metadata to load into Tessent Shell at boot time in order to augment its feature set with your custom Tcl code. The *plugin_directory* can be named anything you like. You point to your *plugin_directory* using the TESSENT_PLUGIN_PATH environment variable. Just like the PATH environment variable, you can specify more than one search location by separating multiple directory names with a colon.

You can also name your *plugin_directory* "tessent_plugin" and place it one level above the bin directory of Tessent Shell, By doing so, the directory is automatically loaded without requiring the end user to set their TESSENT_PLUGIN_PATH environment variable. A user must define the environment variable TESSENT_PLUGIN_IGNORE_DEFAULT_PATH to suppress the auto loading of this plugin directory.

The file *tessent_plugin/tcl_module/<your_company>/defaults.tcl* is the best place to include the definition of your company defaults as it is automatically be seen by all users of the Tessent Shell tool. The Tcl file simply needs to include a this command:

```
read_config_data -from_string {
     DefaultsSpecification(company) {
    <your defaults here>
  }
}
```

The content and purpose of each sub directory and file is explained in the next section.

Components

- tcl_modules/*your_company_name*

  The directory *tcl_modules* is where you define new commands and new metadata. The name *tcl_modules* cannot be changed. Inside this directory, you use an arbitrary sub-directory name which is typically the name of your company. If you are part of a large company with many divisions and you do not want to have to coordinate with them the names of your Tcl modules, you can also append your division name to your company name.

  - *<any_name>.tcl*

    All of the Tcl files specified at this level are loaded when Tessent Shell starts. You should limit the amount of Tcl code those files contains so as not to slow down the Tessent Shell boot time. These files should only include attribute and command registrations and the registration must be declared within a "namespace eval *SomeNameSpace*" block as shown in the Example 1 below. In the same example, the create_wrapper_proc command is used around the proc name of the "register_tcl_command -tcl_proc" argument to add special code that loads the body of the proc from files named *SomeTclNameSpace*/*.tcl when the command is called for the first time. The wrapper proc also disables tool command transcripting while

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

running your proc and restores it to its original setting after your command has completed. You can use the "set_transcript_style full" command in your procs if you want them to still transcript or enable transcripting on demand based on an option as is done with the "process_dft_specification -transcript_insertion_commands" option. The wrapper proc provides handling of the Ctrl-C interrupt such that it terminates your command with the appropriate error message and error code. Finally, if your proc has a bug which results in a Tcl command to fail, it echoes the detailed error message with the calling stack so that you can quickly identify the location of the problem. See Example 1 below on how to create a proc called "u" to quickly reload the body of your procs after you have corrected issues.

- *<any_name>.tessent_meta*

  All the files having an extension ".tessent_meta" are read in the tool at boot time. The files are sorted alphabetically before being read. See the section about the "Metadata Configuration Syntax" on page 4757 for an explanation of the syntax and when you may want to use it. Realize that the metadata loading order is important when you are using the "reference" property as documented in the Wrapper section because the referenced elements must have been read before they are referenced. The alphabetical ordering of the files provides you the ability to control read ordering. Also note that metadata files contained in the *<SomeTclNameSpace>* sub-directories are always loaded after the one at the current level.

- *<SomeTclNameSpace>/<any_name>.tcl*

  The directory name *<SomeTclNameSpace>* must match the namespace name used in the *<any_name>.tcl* files above it. The create_wrapper_proc command uses the leaf namespace from where it is called to know which files to auto load the first time it is run. For example, if the create_wrapper_proc command is invoked within the "namespace eval ABC {}" block, it automatically sources any file matching "./ABC/*.tcl". The tcl code found inside the "ABC/*.tcl" do not need to be inside a "namespace eval ABC {}" block as the wrapper proc automatically adds them to the namespace.

- *<SomeTclNameSpace>/<any_name>.tessent_meta*

  Those files are loaded at boot time. They are loaded after the file found one level above. See the discussion in the "*<any_name>.tessent_meta*" section to find out when that may be important. It is best to place your ".tessent_meta" file at this lower level to avoid cluttering the *tcl_modules/<your_company_name>* directory.

- packages

  This directory contains sub directories which are automatically added to the Tcl autopath search when the "package require" command is invoked. Each sub-directory must each contains a file called pkgIndex.tcl and **.tcl* files which are loaded when the given "package require" command is issued.

  - *<any_name>/pkgIndex.tcl*

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This file has a very specific format. The format is shown below and is used to load a series of Tcl files when the associated package is requested. It is recommended to define <*my_package*> and <*my_namespace*> identically and using this format ::<*my_company*>::<*my_feature*>. With this convention, you can easily request the package and import its exposed commands to your local namespace as shown in Example 2. The *filename* strings are the name of the Tcl file implementing features of the package and they must exist in the same directory.

```
package ifneeded ::my_company::debug_utilities 1.0 \
      [string map [list @ $dir ! ::my_company::debug_utilities] {
  namespace eval {!} { source [file join {@} timer_procs.tcl] }
  namespace eval {!} { source [file join {@} conversion_procs.tcl] }
  package provide ::my_company::debug_utilities 1.0
}]
```

- <*any_name*>/<*any_name*>*.tcl*

  Those Tcl files contain procs which are to be made available to the user when he requests the package. You include a "namespace export *proc_name*" on top of each proc you want to be exportable to the end user. Leave out the "namespace export" command for any internal proc not intended for the end user. See Example 2 below for an example of such files.

Example 1

This example shows you how to create a plugin that defines a tcl_module implementing commands used to manage stacks.

Following the directory structure described above, these two files are created:

```
my_plugin/tcl_modules/my_company/stack_commands.tcl
my_plugin/tcl_modules/my_company/stack_commands/command_body.tcl
```

As you can see below, the file *stack_command.tcl* contains only register_tcl_command invocations inside a "namespace eval stack_commands" Tcl block as well as a variable definition. When any of those registered Tcl commands are called, it calls the wrapper proc that was created by the create_wrapper_proc command. This wrapper proc loads the file *stack_commands/command_body.tcl* the first time is it called and then call the proc it has just loaded.

The example uses a namespace variable called stack to hold an array of stacks. Each sub-command which operates on the stacks has access to common data through this variable.

To have this plugin automatically loaded into Tessent Shell, you simply need to set the environment variable TESSENT_PLUGIN_PATH to point to it. In this example, the variable points to *my_plugin* and *his_plugin*:

```
setenv TESSENT_PLUGIN_PATH ~/my_plugin:~him/his_plugin
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In the example below, imagine the new_stack proc had a typo and the second to last line was using the undefined variable $idd instead of $id as it should. You would get an error when you tried to use it as shown in the transcript below. Note how the error message provides the full context of the error including the calling stack. This information is essential for locating errors especially as the code becomes complex.

These kinds of mistakes are very common as you develop code and below is a useful method to fix them easily. Define a Tcl proc called "u" which you can use to reload the *.*tcl* files found in the sub directory. That way, you can edit the source file, save the update, type the u command and test the updated files without having to restart Tessent Shell. An example definition of a "u" proc is shown below. Once the "u" command was run and it reloaded the corrected file, you can see the proper operation of the stack commands in the transcript below.

```
> proc u {} {
    namespace eval ::tcl_modules::my_company::stack_commands {
      foreach file_name \
        [glob ../my_plugin/tcl_modules/my_company/stack_commands/*.tcl] {
        puts "updating $file_name"
        source $file_name
      }
    }
}
set s [new_stack]
can't read "idd": no such variable
    while executing
"set stack($idd) [list]"
    (procedure "new_stack" line 8)
    invoked from within
"new_stack {*}$args"
=================
TCL calling stack
  ::tcl_modules::my_company::stack_commands::new_stack.wrapper --
namespace ::
=================
> u
updating ../my_plugin/tcl_modules/my_company/stack_commands/
command_body.tcl
> set s [new_stack]
> push $s v1
> push $s v2
> puts [pop $s]
v2
> puts [top_of $s]
v1
> puts [pop $s]
v1
> puts [pop $s]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Content of the *stack_commands.tcl* file:

```
namespace eval stack_commands {
  variable stack  register_tcl_command push \
    -context_list {all} \
    -system_mode_list {setup analysis insertion} \
    -string_value {<stack_id> stack_id} \
    -string_value {<value> value} \
    -syntax_bnf {<stack_id> <value>} \
    -tcl_proc [create_wrapper_proc push]
  register_tcl_command pop \
    -context_list {all} \
    -system_mode_list {setup analysis insertion} \
    -string_value {<stack_id> stack_id} \
    -syntax_bnf {<stack_id>} \
    -tcl_proc [create_wrapper_proc pop]
  register_tcl_command top_of \
    -context_list {all} \
    -system_mode_list {setup analysis insertion} \
    -string_value {<stack_id> stack_id} \
    -syntax_bnf {<stack_id>} \
    -tcl_proc [create_wrapper_proc top_of]
  register_tcl_command new_stack \
    -context_list {all} \
    -system_mode_list {setup analysis insertion} \
    -syntax_bnf {} \
    -tcl_proc [create_wrapper_proc new_stack]
}
```

Content of the *stack_commands/command_body.tcl* file:

```
proc pop {args} {
  variable stack
  array set ARGS $args
  set stack_id $ARGS(stack_id)
  check_id $stack_id
  if {[llength $stack($stack_id)] > 0} {
    set res [lindex $stack($stack_id) end]
    set stack($stack_id) [lreplace $stack($stack_id) end end]
  } else {
    set res ""
  }
  return $res
}
proc push {args} {
 variable stack
 array set ARGS $args
 set stack_id $ARGS(stack_id)
 set value $ARGS(value)
 check_id $stack_id
 lappend stack($stack_id) $value
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
proc top_of {args} {
  variable stack
  array set ARGS $args
  set stack_id $ARGS(stack_id)
  check_id $stack_id
  if {[llength $stack($stack_id)] > 0} {
    set res [lindex $stack($stack_id) end]
  } else {
    set res ""
  }
  return $res
}
proc new_stack {args} {
  variable stack
  if {[array exists stack]} {
    set id [expr {[::tcl::mathfunc::max {*}[array names stack]] + 1}]
  } else {
    set id 0
  }
  set stack($id) [list]
  return $id
}
# local utility procproc check_id {stack_id} {
  variable stack
  if {![info exists stack($stack_id)]} {
    display_message "The stack id '$stack_id' is not active."
    return -code error
  }
}
```

Example 2

This example shows you how to define a package which, when requested by the user, loads and exposes a series of debug utility procs. The files for this package are the following:

```
my_plugin/packages/debug_utilities/pkgIndex.tcl
my_plugin/packages/debug_utilities/timer_procs.tcl
my_plugin/packages/debug_utilities/conversion_procs.tcl
```

As you can see below, the user simply needs to issue the “package require” command to request the loading of the package. One can use the loaded command by specifying the complete namespace path to the commands such as “::my_company::debug_utilities::set_timer_start_point A” but it is more convenient to import the commands into the local namespace using the “namespace import” command as shown below. After that, the procs becomes available locally as if they were defined in the local namespace.

Notice also the “u” proc in the example below. It is very useful again to enable debugging the source code of the procs without having to restart Tessent Shell. The “u” proc is slightly different than the one shown in Example 1 as the *pkgIndex.tcl* file must be excluded from the reload.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
> package require ::my_company::debug_utilities
1.0
> namespace import ::my_company::debug_utilities::*
> puts [join [info procs *timer*] \n]
report_timer_names
get_time_from_timer_start_point
set_timer_start_point
> puts [join [info procs *convert*] \n]
convert_to_wdhms
>
> proc u {} {
  namespace eval ::my_company::debug_utilities {
     set file_list [glob ../my_plugin/packages/debug_utilities/*.tcl]
    set pkgIndex_location [lsearch -glob $file_list */pkgIndex.tcl]
    foreach file_name [lreplace $file_list $pkgIndex_location \
                                           $pkgIndex_location] {
      puts "updating $file_name"
      source $file_name
    }
  }
}
>
> set_timer_start_point A
Setting timer 'A' start time at '15:01:21'
> set_timer_start_point B
Setting timer 'B' start time at '15:01:21'
> get_time_from_timer_start_point A
weeks 0 days 0 hours 0 minutes 0 seconds 2
> get_time_from_timer_start_point B
weeks 0 days 0 hours 0 minutes 0 seconds 7
```

Content of the *debug_utilities/pkgIndex.tcl* file:

```
package ifneeded ::my_company::debug_utilities 1.0 \
      [string map [list @ $dir ! ::my_company::debug_utilities] {
  namespace eval {!} { source [file join {@} timer_procs.tcl] }
  namespace eval {!} { source [file join {@} conversion_procs.tcl] }
  package provide ::my_company::debug_utilities 1.0
}]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Content of the *debug_utilities/timer_procs.tcl* file:

```
variable timer_start_point_dict [dict create]
namespace export set_timer_start_point
proc set_timer_start_point {timer_name} {
  variable timer_start_point_dict
  dict set timer_start_point_dict $timer_name [clock seconds]
  puts "Setting timer '$timer_name' start time at '[clock format [clock
seconds] -format "%H:%M:%S"]'"
}
namespace export report_timer_names
proc report_timer_names {} {
  variable timer_start_point_dict
  if {[dict size $timer_start_point_dict] > 0} {
    set timer_start_points [dict keys $timer_start_point_dict]
    puts "Available timer names are:\n  [join $timer_start_points "\n  "]"
  } else {
    puts "No timer name exists"
  }
}
namespace export get_time_from_timer_start_point
proc get_time_from_timer_start_point {timer_name} {
  variable timer_start_point_dict
  if {[dict exist $timer_start_point_dict $timer_name]} {
    set start_point_time_val \
       [dict get $timer_start_point_dict $timer_name]
    set time_diff [expr [clock seconds] - $start_point_time_val]
    set converted_time_dict [convert_to_wdhms $time_diff]
    return $converted_time_dict
  } else {
    display_message -error "Timer name '$timer_name' does not exist"
    return -code error
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Content of the debug_utilities/conversion_procs.tcl file:

```
namespace export convert_to_wdhms
proc convert_to_wdhms {seconds} {
  if { ![string is digit -strict $seconds]} {
    display_message -error \
        "The supplied number is not an integer: '$seconds'"
    return -code error
  }
  set week_val   [expr $seconds/604800]
  set seconds [expr $seconds - ($week_val *604800)]
  set day_val    [expr $seconds/86400]
  set seconds [expr $seconds - ($day_val *86400)]
  set hour_val   [expr $seconds/3600]
  set seconds [expr $seconds - ($hour_val*3600)]
  set minute_val [expr $seconds/60]
  set second_val [expr ($seconds - ($minute_val*60))]
  set time_conversion_results_dict\
           [dict create weeks $week_val \
                        days $day_val \
                        hours $hour_val \
                        minutes $minute_val\
                        seconds $second_val]
  return $time_conversion_results_dict
}
```

Related Topics

register_tcl_command

register_attribute

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 18
Generation of ScanDEF

This chapter provides detailed information about ScanDEF.

ScanDEF File Generation 4991
ScanDEF File Format 4992

ScanDEF File Generation

You use the command write_scan_order to generate the ScanDEF file.

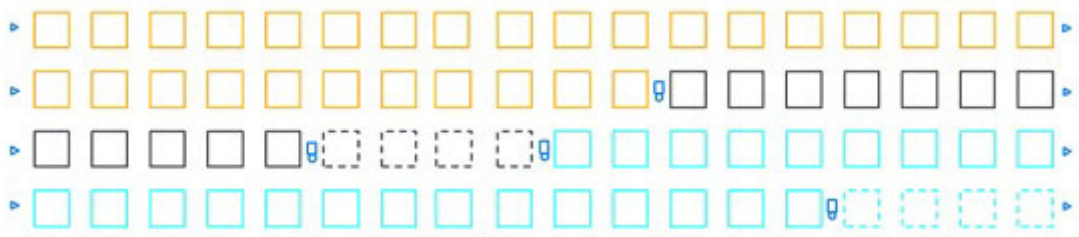
In the legacy Tessent Shell scan insertion flow (which is equivalent to a single scan mode environment), every scan chain's make is unique and therefore can be written out into a scanDEF file using a simple structure. However, the same scan elements can be stitched into different scan chains in the Hierarchical Scan Insertion flow using a multi scan mode environment. Furthermore, since a single scanDEF file written out in the new flow would have to contain all scan elements regardless of the scan mode, it uses the unique scan segments rather than the unique scan chains to create the scan element groupings used by the layout tools for the scan chains optimization.

Each scan segment comprises a set of scan path cells of a scan chain between the so called persistent components, which are buffer, mux, and/or (NOTE: only 2x1 actual MUXs are used; so, when there are more than 2 inputs, the AND/OR muxing is being used), and retiming cells. These persistent components are written out into the START/STOP sections of each scanDEF segment corresponding to the scan segment, while the scan path cells of a scan segment are written out into the ORDERED and/or FLOATING lists in the body section of a scanDEF segment.

In the simplest case of scan chains in the single scan mode environment (see Figure 18-1), with four scan chains and three clock domains, each scan chain are sub-divided into the scan segments based on the clock/edge domain information, where the persistent components consist of a buffer at the beginning and end of each scan chain and a retiming cell at the clock/edge domain transition points.

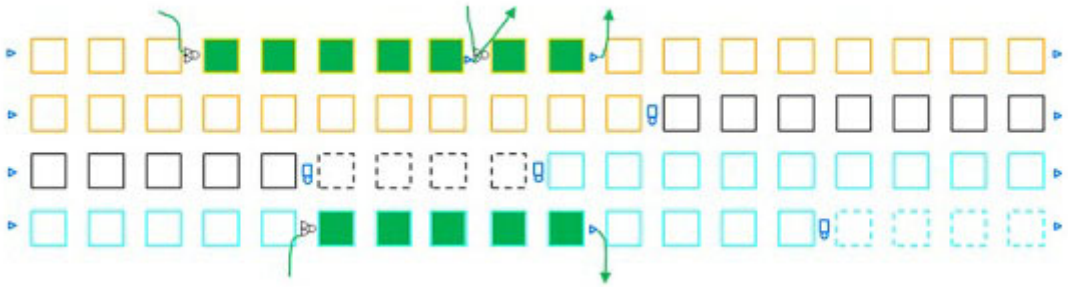
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 18-1. Single Scan Mode Environment



In the case of multi mode environment (see Figure 18-2), having InTest and ExTest scan modes, with four scan chains in the InTest mode, three wrapper chains in the ExTest mode, and three clock domains, each scan chain is subdivided into the scan segments based on the clock/edge domain information plus the scan modes scan chains information. The persistent components in this case consists of a buffer or mux (and/or) at the beginning of each scan chain, a buffer at the end of each scan chain, and a retiming cell at the clock/edge domain transition points.

Figure 18-2. Multi Scan Mode Environment



In general, the scan insertion tool always ensures that persistent gates exist at the beginning and end of scan chains, and when transitioning between constraint sets that need it. In addition, for power domain island transitions, persistent gates get inserted at the boundary of each power domain islands in order to simplify the insertion of level shifters.

Modules which have the “exclude_from_concatenated_netlist” attribute specified are considered as hard macros and are not included in the concatenated netlist. Moreover, when writing the scanDEF file, any traced scan elements from existing segments on these modules are omitted between the scanIn and scanOut sections of these segments, and stop at the SI/SO boundary, as is done for a cell library.

ScanDEF File Format

The ScanDEF file contains all unique scan segments contributing to all scan chains in all scan modes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following is an example of a typical ScanDEF segment:

```
- scan_segment_0
 + START tessent_persistent_cell_mux_stm1s1_intsi17_intsi13_i Y
 + ORDERED
   uWA/uA/f1 ( IN SI ) ( OUT Q )
   uWA/uA/bb/f2 ( IN SI ) ( OUT so )
   uWA/uA/f3 ( IN SI ) ( OUT Q )
 + FLOATING
   uWA/uA/f21 ( IN SI ) ( OUT Q )
 + STOP uWA/tessent_persistent_cell_buf_extso6_i A
 # Partition constraints - clock domain: clk(+); chain type: core; scan
mode(s): InTest ExTest
 + PARTITION partition_1  MAXBITS 4 ;
```

- Each ScanDEF segment starts with a unique segment name with the following format: **scan_segment_%d**
- The START section references an output pin of a persistent component, which is either a buffer, mux, or a retiming cell.
- The STOP section references an input pin of a persistent component, which is either a buffer or a retiming cell.
- Typically, the body of a ScanDEF segment contains at least one FLOATING or ORDERED list. The tool creates an ORDERED list for the cells in the known scan segment scan path (unless the scan segment contains only 1 element) and the cells in a shift register. The tool writes all other cells into the FLOATING list. Only one FLOATING list per scan segment is allowed, per the language specification.
- Also, a ScanDEF segment contains a PARTITION name. The PARTITION statement enables reordering tools to determine the inter-segment compatibility for element swapping (both FLOATING elements and ORDERED lists as a whole). That is, partition names determine the segments compatibility for repartitioning by swapping elements between them. Therefore, segments with matching PARTITION names constitute a swap-compatible group. The MAXBITS constraint specifies the length of the segments included in the same partition, which is typically set to the number of elements in the segment's body.

  Note

  If you do not provide a PARTITION name, the tool assumes the segment to be in its own single partition, and you can only perform reordering within this segment.

  The tool generates the PARTITION name based on the following criteria for a given scan segment:

  - Clock domain, clock edge, and unbalanced clock pins.
  - This segment's scan chains belong to:
    - Clusters

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Power Domains
- Scan Modes

  - Chain type (core, input_wrapper, output_wrapper).

The “Partition constraints” comment line mentions which criteria it uses.

ScanDEF File Example

Consider the following scan chain configuration: two scan modes (two scan chains in int_mode, one scan_chain in ext_mode) and three clock domains, including one with inverted polarity. For such a design, the report_scan_cells command generates the following output:

```
Scan mode 'int_mode' scan cells:
-----------------------------------------------------------------------------------------
        Chain   Group                          ShiftReg   Library            Clock    Clock
CellNo  Name    Name   Pathname               ID/CellNo  ModelName  ScanOut  Pinname  Polarity
-----------------------------------------------------------------------------------------
-   int_mode_chain0 dummy  /ts_lockup_latchn_clkc0_intno17_i -/- latchN       clk2    (-)
0   int_mode_chain0 dummy  /ud8                2/2        dff        QB       clk2    (+)
1   int_mode_chain0 dummy  /ud7                2/1        sff        Q        clk2    (+)
2   int_mode_chain0 dummy  /ud15               -/-        sff        Q        clk2    (+)
3   int_mode_chain0 dummy  /ud14               -/-        sff        Q        clk2    (+)
4   int_mode_chain0 dummy  /ud13               -/-        sff        Q        clk2    (+)
5   int_mode_chain0 dummy  /ud12               -/-        sff        Q        clk2    (+)
6   int_mode_chain0 dummy  /ud11               -/-        sff        Q        clk2    (+)
7   int_mode_chain0 dummy  /ud9                -/-        sff        Q        clk2    (+)
0   int_mode_chain1 dummy  /ud4                1/2        dff        QB       clk1    (-)
1   int_mode_chain1 dummy  /ud3                1/1        sff        Q        clk1    (-)
-   int_mode_chain1 dummy  /ts_lockup_latchn_clkc1_intno4_i -/- latchN        clk1    (-)
2   int_mode_chain1 dummy  /ud5                -/-        sff        Q        clk1    (+)
3   int_mode_chain1 dummy  /ud2                -/-        sff        Q        clk1    (+)
4   int_mode_chain1 dummy  /ud10               -/-        sff        Q        clk1    (+)
5   int_mode_chain1 dummy  /ud1                -/-        sff        Q        clk1    (+)
6   int_mode_chain1 dummy  /ud6                -/-        sff        Q        clk1    (+)
-----------------------------------------------------------------------------------------


Scan mode 'ext_mode' scan cells:
-----------------------------------------------------------------------------------------
        Chain   Group                          ShiftReg   Library            Clock    Clock
CellNo  Name    Name   Pathname               ID/CellNo  ModelName  ScanOut  Pinname  Polarity
-----------------------------------------------------------------------------------------
-   ext_mode_chain0 dummy  /ts_lockup_latchn_clkc1_intno4_i -/- latchN       clk1     (-)
0   ext_mode_chain0 dummy  /ud5                -/-        sff        Q        clk1     (+)
1   ext_mode_chain0 dummy  /ud1                -/-        sff        Q        clk1     (+)
2   ext_mode_chain0 dummy  /ud6                -/-        sff        Q        clk1     (+)
-----------------------------------------------------------------------------------------
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The respective ScanDEF output looks as follows:

```
#
#    DESC: ScanDEF written by Tessent Shell on <>
#

VERSION 5.7 ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN A ;
UNITS DISTANCE MICRONS 1000 ;

SCANCHAINS 5 ;

- scan_segment_0
  + START tessent_persistent_cell_buf_extsi19_i Y
  + FLOATING
      ud9 ( IN SI ) ( OUT Q )
      ud11 ( IN SI ) ( OUT Q )
      ud12 ( IN SI ) ( OUT Q )
      ud13 ( IN SI ) ( OUT Q )
      ud14 ( IN SI ) ( OUT Q )
      ud15 ( IN SI ) ( OUT Q )
  + ORDERED
      ud7 ( IN SI ) ( OUT Q )
      ud8 ( IN D ) ( OUT QB )
  + STOP ts_lockup_latchn_clkc0_intno17_i d
  # Partition constraints - clock domain: clk2(+);
  #                         chain type: core;
  #                         scan mode(s): int_mode
  + PARTITION partition_1  MAXBITS 8 ;


- scan_segment_1
  + START tessent_persistent_cell_mux_stm1s3_extsi21_extsi20_i Y
  + ORDERED
      ts_sihw_mux_sea0_not_so5__si5_i ( IN A1 ) ( OUT Y ) ( BITS 0 )
      ud6 ( IN SI ) ( OUT Q )
  + ORDERED
      ts_sihw_mux_sea0_not_so15__si15_i ( IN A1 ) ( OUT Y ) ( BITS 0 )
      ud1 ( IN SI ) ( OUT Q )
  + STOP tessent_persistent_cell_buf_extso15_i A
  # Partition constraints - clock domain: clk1(+);
  #                         chain type: input_wrapper;
  #                         scan mode(s): int_mode ext_mode
  + PARTITION partition_2  MAXBITS 2 ;


- scan_segment_2
  + START tessent_persistent_cell_buf_extso15_i Y
  + FLOATING
      ud10 ( IN SI ) ( OUT Q )
      ud2 ( IN SI ) ( OUT Q )
  + STOP tessent_persistent_cell_mux_stm1s3_intno15_intno18_i A1
  # Partition constraints - clock domain: clk1(+);
  #                         chain type: core;
  #                         scan mode(s): int_mode
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
    + PARTITION partition_3  MAXBITS 2 ;


  - scan_segment_3
    + START tessent_persistent_cell_mux_stm1s3_intno15_intno18_i Y
    + ORDERED
        ts_sohw_mux_sea0_not_so4__si4_i ( IN A1 ) ( OUT Y ) ( BITS 0 )
        ud5 ( IN SI ) ( OUT Q )
    + STOP ts_lockup_latchn_clkc1_intno4_i d
    # Partition constraints - clock domain: clk1(+);
    #                         chain type: output_wrapper;
    #                         scan mode(s): int_mode ext_mode
    + PARTITION partition_4  MAXBITS 1 ;


  - scan_segment_4
    + START ts_lockup_latchn_clkc1_intno4_i q
    + ORDERED
        ud3 ( IN SI ) ( OUT Q )
        ud4 ( IN D ) ( OUT QB )
    + STOP tessent_persistent_cell_buf_extso16_i A
    # Partition constraints - clock domain: clk1(-);
    #                         chain type: core;
    #                         scan mode(s): int_mode
    + PARTITION partition_5  MAXBITS 2 ;


  END SCANCHAINS

  END DESIGN
```

See Pre-Existing Scan Segments in the *Tessent Scan and ATPG User's Manual* for more information and an example of a ScanDEF file section for a scan segment that spans multiple lower-level modules.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Appendix A
Parameter File Format and Keywords

This appendix contains reference material about the parameter file statements.

What is a Parameter File? **5000**

Parameter File Syntax **5001**

Parameter File Keywords **5002**

A Note about the ALL_* Parameter File Keywords 5006
ALL_EDT_INTERNAL_USE_NUMBER_SHIFTS 5006
ALL_EXCLUDE_POWER_GROUND 5006
ALL_EXCLUDE_UNUSED 5007
ALL_FIXED_CYCLES 5008
ALL_FLATTEN_TIMING 5009
ALL_FULL_TIMEPLATES 5009
ALL_FULL_VARIABLES 5009
ALL_HEADER_COMMENT 5010
ALL_IDDQ_TESTER_CYCLE 5010
ALL_IGNORE_SIMULATION_ONLY 5011
ALL_MAX_LOOP 5011
ALL_MIN_SCAN_LOAD 5012
ALL_NO_CYCLE_OPT 5012
ALL_NO_LOOP 5013
ALL_NO_PATTERN_TYPE 5013
ALL_NONSCAN_CONSTANT 5013
ALL_ORIGINAL_INDEX 5014
ALL_OVERRIDE_STROBE_METHOD 5014
ALL_PRESERVE_FORCE_X 5015
ALL_SINGLE_QUOTED_SSN_ANNOTATIONS 5015
ALL_SSN_MAXIMUM_BUS_SPEED 5016
ALL_TIME_RESOLUTION 5017
ALL_USE_CYCLE_LOOP 5017
CTL_ONE_PAT 5017
CTL_NO_PROTO 5017
CTL_STIL_0 5018
CTL_STIL_1 5018
FTDL_DESIGNER 5019
FTDL_MAX_PAT_BLOCK 5019
FTDL_REVISION 5019
FTDL_TCOMMENT 5020
FTDL_TEST_NAME 5020
FTDL_TEST_NUM 5020

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

FTDL_TEST_SUFFIX 5020
FTDL_ZMODE_MES 5021
SIM_ANNOTATE_QUIET 5021
SIM_CHAIN_ERROR 5022
SIM_CLOCK_MONITOR 5022
SIM_COMPARE_SUMMARY 5022
SIM_COMPARE_X 5023
SIM_DELAY_SCAN_RELEASE 5024
SIM_DIAG_FILE 5025
SIM_DISTRIBUTED_PAT 5026
SIM_DUMPFILE_PATH 5026
SIM_EARLY_RELEASE 5027
SIM_EARLY_RELEASE_LAST 5027
SIM_EARLY_RELEASE_TIME 5028
SIM_FILL_BIDISCAN 5028
SIM_FORCE_Z_AT_CYCLE 5029
SIM_INCLUDE 5029
SIM_INSTANCE_ARGUMENTS 5029
SIM_INSTANCE_NAME 5030
SIM_KEEP_PATH 5030
SIM_MASK_FILE 5031
SIM_MIN_NAME_FILE 5031
SIM_MISCOMPARE_LIMIT 5032
SIM_MODULE_INCLUDE 5032
SIM_MODULE_NAME 5033
SIM_NAME_FILE 5033
SIM_NO_CHAINNAME_FILE 5033
SIM_NO_INTERNAL_COMPARE 5034
SIM_NO_MEASURE_IN 5034
SIM_PARALLEL_DELAY_SHIFT_CLOCKS 5034
SIM_PARALLEL_EARLY_FORCE_AND_MEASURE 5035
SIM_PARALLEL_MONITOR 5035
SIM_POST_SHIFT 5036
SIM_PRE_SHIFT 5037
SIM_PRECISION 5037
SIM_SERIAL_CHAIN 5037
SIM_SHIFT_DEBUG 5038
SIM_SSN_MAXIMUM_BUS_SPEED 5039
SIM_SSN_TIMING_CHECKS 5039
SIM_STATUS_MSG 5040
SIM_STRETCH_SCAN_RELEASE 5040
SIM_TIMEPLATE_COMM 5041
SIM_TMP_REG_LENGTH 5041
SIM_TOP_NAME 5041
SIM_UNCOMPRESS_COMMAND 5042

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SIM_VECTOR_COMM 5042
SIM_VECTYPE_SIGNAL 5043
STIL_2005_SCAN 5045
STIL_ADD_TIME_0_FORCE_PREVIOUS_EVENT 5045
STIL_ADD_TIME_0_MEASURE_OFF_EVENT 5046
STIL_ALLOW_SCAN_BIDI_WAVEFORM 5046
STIL_ANNOTATION_INTERVAL 5047
STIL_CHAIN_CELLS 5048
STIL_CHANNEL_CELLS 5048
STIL_CLOCK_WFC_PULSE 5049
STIL_DUAL_MODE 5049
STIL_EXCLUDE_UNUSED_BUS_BITS 5050
STIL_FULL_CHAIN 5050
STIL_IDDQ_UPPERCASE 5051
STIL_MIN_QUOTE 5051
STIL_NESTED_MACRO 5052
STIL_NO_OPTIMIZE_VEC 5052
STIL_NO_SCANSTRUCTURE 5053
STIL_NO_SCANX 5053
STIL_NO_WFC_T 5054
STIL_NOMEASURE_CLOCK 5054
STIL_PAT_ANN 5054
STIL_PAT_CMT 5055
STIL_PAT_LAB 5055
STIL_QUAD_MODE 5055
STIL_QUOTE_ALL 5056
STIL_SCAN_ANN1 5056
STIL_SCAN_IN_OUT_USE_PSEUDO_PORT_NAME 5057
STIL_STRUCTURAL 5057
STIL_TI_TITLE 5059
STIL_TIMING_BLOCK_NAME 5060
STIL_TRIM_PULSE 5060
STIL_VECTOR_ANN 5061
STIL_VERG_ESC 5062
STIL_WFC_KEEP_N 5063
STIL_WFC_KEEP_Z 5063
STIL_WRAP_ANNOTATION 5064
STIL2005_ONE_FILE 5064
TITDL_CHAIN_SETTYPE 5065
TITDL_CUSTOMER 5065
TITDL_GROUP_TIMING 5065
TITDL_KEEP_SCANOUT 5066
TITDL_LIBRARY 5066
TITDL_PARTNUM 5066
TITDL_PSEUDO_PREFIX 5067

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TITDL_REVISION 5067
TITDL_SET_DESCRIPTION 5068
TITDL_SETNAME 5068
TITDL_SETTYPE 5068
WGL_ADD_LASTX 5069
WGL_ALT_BIDI 5069
WGL_ALT_VECT_ANN 5070
WGL_ANNOTATION_INTERVAL 5070
WGL_EDGE_STROBE 5071
WGL_EXCLUDE_UNUSED_BUS_BITS 5071
WGL_FULL_CHAIN 5072
WGL_FULL_SCANGROUP 5073
WGL_GROUP_PIN 5074
WGL_INV_SC 5074
WGL_LOOP_ONLY 5075
WGL_NO_SCANX 5075
WGL_NOMEASURE_CLOCK 5075
WGL_ONE_ILLINOIS 5076
WGL_PATTERN_NAME 5076
WGL_TRIM_LEADING_P 5077
WGL_TRIM_PULSE 5077
WGL_VECTOR_ANN 5078
WGL_VERG_ESC 5078
WGL_VTRAN_PADSC 5079
WGL_WRAP_ANNOTATION 5079

What is a Parameter File?

The parameter file specifies field names and values for use in certain Vector Interfaces output formats.

The tool uses this file when you specify with the “write_patterns -Parameter” command.

The tool adds a section at the beginning of STIL, WGL, and Verilog output files that captures the environment data and the switches specified with the write_patterns command.

- In the STIL format, this information is added to the HEADER{} block as an annotation.
- In the WGL format, this information is added to the header block as a series of annotation statements.
- In the Verilog format, this information is added to the beginning of the Verilog testbench as a comment.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Parameter File Syntax

The parameter file is a text file containing a series of parameter statements.

Each statement uses the following format:

```
KEYWORD arguments;
```

Where:

- **UPPERCASE** — Indicates a keyword and must be entered exactly as shown.
- ***Italic*** — Indicates a user-supplied argument.
- {} — Indicates a multi-option argument. Include only one option and do not enter the braces.
- **Underline** — Indicates the default setting for a multi-option argument.

The parameter file has the following rules:

- All parameter statements must end with a semicolon.
- Boolean parameters must have a literal value of "1", "Y", "Yes", "T", or "True" for true; "0", "N", "No", "F", or "False" for false. The value is case insensitive, that is "Yes" and "YES" are equivalent. The tool issues an error if the Boolean or integer parameter is invalid.
- Comments are marked with double slashes "//" at the beginning of a line. The tool ignores everything between the slashes and the end of line character.
- You can use each parameter only once per file. If you use a parameter more than once, the latter entry overrides the previous one.
- Each parameter statement must occupy a single line. You cannot embed carriage returns within a parameter statement.
- Each parameter statement line cannot exceed 2048 characters. There are no restrictions on the parameter file size.

Note

All parameters are optional, so you can include or exclude as needed. When you omit a parameter, the tool uses the default setting. Refer to each parameter description for the default behavior.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Parameter File Keywords

This is a list of the available parameter statements.

A Note about the ALL_* Parameter File Keywords 5006
ALL_EDT_INTERNAL_USE_NUMBER_SHIFTS 5006
ALL_EXCLUDE_POWER_GROUND 5006
ALL_EXCLUDE_UNUSED 5007
ALL_FIXED_CYCLES 5008
ALL_FLATTEN_TIMING 5009
ALL_FULL_TIMEPLATES 5009
ALL_FULL_VARIABLES 5009
ALL_HEADER_COMMENT 5010
ALL_IDDQ_TESTER_CYCLE 5010
ALL_IGNORE_SIMULATION_ONLY 5011
ALL_MAX_LOOP 5011
ALL_MIN_SCAN_LOAD 5012
ALL_NO_CYCLE_OPT 5012
ALL_NO_LOOP 5013
ALL_NO_PATTERN_TYPE 5013
ALL_NONSCAN_CONSTANT 5013
ALL_ORIGINAL_INDEX 5014
ALL_OVERRIDE_STROBE_METHOD 5014
ALL_PRESERVE_FORCE_X 5015
ALL_SINGLE_QUOTED_SSN_ANNOTATIONS 5015
ALL_SSN_MAXIMUM_BUS_SPEED 5016
ALL_TIME_RESOLUTION 5017
ALL_USE_CYCLE_LOOP 5017
CTL_ONE_PAT 5017
CTL_NO_PROTO 5017
CTL_STIL_0 5018
CTL_STIL_1 5018
FTDL_DESIGNER 5019
FTDL_MAX_PAT_BLOCK 5019
FTDL_REVISION 5019
FTDL_TCOMMENT 5020

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

FTDL_TEST_NAME 5020
FTDL_TEST_NUM 5020
FTDL_TEST_SUFFIX 5020
FTDL_ZMODE_MES 5021
SIM_ANNOTATE_QUIET 5021
SIM_CHAIN_ERROR 5022
SIM_CLOCK_MONITOR 5022
SIM_COMPARE_SUMMARY 5022
SIM_COMPARE_X 5023
SIM_DELAY_SCAN_RELEASE 5024
SIM_DIAG_FILE 5025
SIM_DISTRIBUTED_PAT 5026
SIM_DUMPFILE_PATH 5026
SIM_EARLY_RELEASE 5027
SIM_EARLY_RELEASE_LAST 5027
SIM_EARLY_RELEASE_TIME 5028
SIM_FILL_BIDISCAN 5028
SIM_FORCE_Z_AT_CYCLE 5029
SIM_INCLUDE 5029
SIM_INSTANCE_ARGUMENTS 5029
SIM_INSTANCE_NAME 5030
SIM_KEEP_PATH 5030
SIM_MASK_FILE 5031
SIM_MIN_NAME_FILE 5031
SIM_MISCOMPARE_LIMIT 5032
SIM_MODULE_INCLUDE 5032
SIM_MODULE_NAME 5033
SIM_NAME_FILE 5033
SIM_NO_CHAINNAME_FILE 5033
SIM_NO_INTERNAL_COMPARE 5034
SIM_NO_MEASURE_IN 5034
SIM_PARALLEL_DELAY_SHIFT_CLOCKS 5034
SIM_PARALLEL_EARLY_FORCE_AND_MEASURE 5035
SIM_PARALLEL_MONITOR 5035
SIM_POST_SHIFT 5036

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SIM_PRE_SHIFT 5037
SIM_PRECISION 5037
SIM_SERIAL_CHAIN 5037
SIM_SHIFT_DEBUG 5038
SIM_SSN_MAXIMUM_BUS_SPEED 5039
SIM_SSN_TIMING_CHECKS 5039
SIM_STATUS_MSG 5040
SIM_STRETCH_SCAN_RELEASE 5040
SIM_TIMEPLATE_COMM 5041
SIM_TMP_REG_LENGTH 5041
SIM_TOP_NAME 5041
SIM_UNCOMPRESS_COMMAND 5042
SIM_VECTOR_COMM 5042
SIM_VECTYPE_SIGNAL 5043
STIL_2005_SCAN 5045
STIL_ADD_TIME_0_FORCE_PREVIOUS_EVENT 5045
STIL_ADD_TIME_0_MEASURE_OFF_EVENT 5046
STIL_ALLOW_SCAN_BIDI_WAVEFORM 5046
STIL_ANNOTATION_INTERVAL 5047
STIL_CHAIN_CELLS 5048
STIL_CHANNEL_CELLS 5048
STIL_CLOCK_WFC_PULSE 5049
STIL_DUAL_MODE 5049
STIL_EXCLUDE_UNUSED_BUS_BITS 5050
STIL_FULL_CHAIN 5050
STIL_IDDQ_UPPERCASE 5051
STIL_MIN_QUOTE 5051
STIL_NESTED_MACRO 5052
STIL_NO_OPTIMIZE_VEC 5052
STIL_NO_SCANSTRUCTURE 5053
STIL_NO_SCANX 5053
STIL_NO_WFC_T 5054
STIL_NOMEASURE_CLOCK 5054
STIL_PAT_ANN 5054
STIL_PAT_CMT 5055

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

STIL_PAT_LAB 5055
STIL_QUAD_MODE 5055
STIL_QUOTE_ALL 5056
STIL_SCAN_ANN1 5056
STIL_SCAN_IN_OUT_USE_PSEUDO_PORT_NAME 5057
STIL_STRUCTURAL 5057
STIL_TI_TITLE 5059
STIL_TIMING_BLOCK_NAME 5060
STIL_TRIM_PULSE 5060
STIL_VECTOR_ANN 5061
STIL_VERG_ESC 5062
STIL_WFC_KEEP_N 5063
STIL_WFC_KEEP_Z 5063
STIL_WRAP_ANNOTATION 5064
STIL2005_ONE_FILE 5064
TITDL_CHAIN_SETTYPE 5065
TITDL_CUSTOMER 5065
TITDL_GROUP_TIMING 5065
TITDL_KEEP_SCANOUT 5066
TITDL_LIBRARY 5066
TITDL_PARTNUM 5066
TITDL_PSEUDO_PREFIX 5067
TITDL_REVISION 5067
TITDL_SET_DESCRIPTION 5068
TITDL_SETNAME 5068
TITDL_SETTYPE 5068
WGL_ADD_LASTX 5069
WGL_ALT_BIDI 5069
WGL_ALT_VECT_ANN 5070
WGL_ANNOTATION_INTERVAL 5070
WGL_EDGE_STROBE 5071
WGL_EXCLUDE_UNUSED_BUS_BITS 5071
WGL_FULL_CHAIN 5072
WGL_FULL_SCANGROUP 5073
WGL_GROUP_PIN 5074

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

WGL_INV_SC 5074
WGL_LOOP_ONLY 5075
WGL_NO_SCANX 5075
WGL_NOMEASURE_CLOCK 5075
WGL_ONE_ILLINOIS 5076
WGL_PATTERN_NAME 5076
WGL_TRIM_LEADING_P 5077
WGL_TRIM_PULSE 5077
WGL_VECTOR_ANN 5078
WGL_VERG_ESC 5078
WGL_VTRAN_PADSC 5079
WGL_WRAP_ANNOTATION 5079

A Note about the ALL_* Parameter File Keywords

All of the ALL_* parameter file keywords affect all pattern formats except for binary and ASCII.

ALL_EDT_INTERNAL_USE_NUMBER_SHIFTS

This keyword applies the set_number_shifts specification to edt_internal patterns.

Usage

```
ALL_EDT_INTERNAL_USE_NUMBER_SHIFTS {0 | 1};
```

By default, when you use the set_number_shifts command with EDT on (compressed scan), the command specifies the number of external shifts used for loading the EDT channels (write_patterns -edt_external). However, when writing patterns using -edt_internal, the set_number_shifts specification does not apply. Setting ALL_EDT_INTERNAL_USE_NUMBER_SHIFTS to 1 applies the set_number_shifts specification to any edt_internal patterns that are written. This applies to serial Verilog testbenches and the other tester formats.

ALL_EXCLUDE_POWER_GROUND

This keyword excludes any port with a “function” value of “power” or “ground” from the written pattern.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Usage

```
ALL_EXCLUDE_POWER_GROUND {AUTO | 1 | 0};
```

By default, the keyword excludes such ports in tester formats, such as WGL and STIL, and does not exclude such ports for Verilog or VHDL. Setting this keyword to 1 excludes power and ground ports from all formats (except binary and ASCII); setting this keyword to 0 prevents exclusion of power and ground ports.

Note that binary and ASCII patterns are not affected by this keyword.

This parameter keyword is affected by the include_all_power_pins property in the Patterns/AdvancedOptions wrapper of the PatternsSpecification.

ALL_EXCLUDE_UNUSED

Removes certain pins from patterns.

Usage

```
ALL_EXCLUDE_UNUSED {0 | 1};
```

The default is 1 or true. The keyword removes the following pins from patterns:

- All output pins that are masked.
- All input pins that are constrained to CX and are not used in any test procedures.
- All bidi pins that are constrained to CX or CZ and are output masked and are either not used in any test procedure or are only forced to the constrained value and are not measured by a “measure” statement in a procedure.

This keyword does not remove any pin that is a scan pin, any EDT or scan control pin, any clocks, any pins constrained to a non-X value (or non-Z for a bidi), or any pins used in any procedure. For ATPG scan pattern retargeting, these are the only pins that are considered to be used. All others are considered to be unused, regardless of CX constraints or output masking.

This keyword does not remove pins from a multi-bit bus unless it can remove the entire bus.

Setting this keyword to 0 or false disables the pin removal.

This keyword is affected by the include_all_power_pins property in the Patterns/AdvancedOptions wrapper of the PatternsSpecification.

This keyword is affected by the include_in_mode_external_patterns pseudo_port attribute. Specifically, if the include_in_mode_external_patterns attribute is set to “true”, the attribute

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

overrides the ALL_EXCLUDE_UNUSED keyword setting and includes the pseudo_port in the pattern file and testbench.

Keyword Usage with IJTAG Patterns

For IJTAG patterns, a primary port is not considered unused if one of the following applies:

- The port is an IJTAG test clock (TCKPort or CaptureEnPort with Attribute function_modifier =“CaptureShiftClock” or function_modifier = “CaptureShiftClockInv”).
- The port has been added as a clock by means of the add_clocks command.
- The port is relevant for the application of the retargeted PDL as follows:
  - ScanInterface ports
  - DataInPorts and DataOutPorts required for the retargeting of iRead and iWrite commands
  - All ports that have been subject to iComparePort or iForcePort
  - If import_patterns_from_svf has been used; all ports that are in the SVF PIOMAP
  - All BSDL compliance ports (ICL ports with attribute compliance_value)

ALL_FIXED_CYCLES

Issuing a 2 with this keyword causes the tool to compute the largest number of cycles between scan loads by looking at all patterns in the pattern list, not just those being saved.

Usage

```
ALL_FIXED_CYCLES {2 | 1 | 0};
```

Issuing a 2 with this keyword causes the tool to compute the largest number of cycles between scan loads by looking at all patterns in the pattern list, not just those being saved. Issuing a 1 with this keyword specifies that the patterns contain an equal number of non-scan cycles between each scan chain loading. This non-scan cycle is applied before the first shift cycle. The default is 0.

This adds “Shift” cycles before the regular apply shift of the load_unload procedure. It adds these additional cycles before the scan chain load to make all patterns have the same total number of cycles per pattern, taking into account the differences in the number of capture cycles. The number of capture cycles can still vary depending on what ATPG needed or was directed with the Named Capture Procedures. However, no additional “Capture” cycles are added. The new cycles use the timeplate that the “Shift” procedure uses.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ALL_FLATTEN_TIMING

Issuing a 1 with this keyword causes all outputs to compute existing timing equations and use only the resulting numeric values in the output files.

Usage

```
ALL_FLATTEN_TIMING {1 | 0};
```

Issuing a 1 with this keyword causes all outputs to compute existing timing equations and use only the resulting numeric values in the output files. The default of 0 results in any output format which can support equation-based timing using the equations as they are specified.

This keyword is intended for having pattern output files not contain equation-based timing that has been added to the procedure file and preserved in the tool. Test data languages such as Verilog, WGL, and STIL have the ability to express time values in the timing blocks, as numerical values or as equations based on variables. Using equation-based timing allows one value to be specified for a global attribute, such as the test cycle period, while other values are derived from this using equations.

ALL_FULL_TIMEPLATES

When this keyword is set to 1 or true, all of the timeplates that have been defined and stored in the tool are written to the resulting pattern file or testbench.

Usage

```
ALL_FULL_TIMEPLATES {0 | 1};
```

Without this, only timeplates that are marked as used by the patterns in the pattern range being written are used. Similar to SIM_NAME_FILE, this should be used when writing out a Verilog testbench that can be used with one design but multiple pattern sets. Turning on ALL_FULL_TIMEPLATES ensures that all timeplates that could be referenced in the *.vec* files in different pattern sets are enumerated in the testbench.

ALL_FULL_VARIABLES

Forces inclusion of all variables and timing_variables defined in the procedure file, whether or not those variables are used by the pattern data.

Usage

```
ALL_FULL_VARIABLES {1 | 0);
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

By default, only those variables and timing_variables that are used by the timing data are included in pattern files that support them. Setting this keyword to 1 forces the tool to include all variables and timing_variables in the pattern files, regardless of whether the pattern data uses them.

ALL_HEADER_COMMENT

Specifies comment lines to add to the header of the pattern file.

Usage

```
ALL_HEADER_COMMENT string;
```

The string specified is added as a comment line to the header of the pattern file. Use an embedded newline '\n' character for the comment string to span multiple lines.

Suppose you want to add comments to the header of a verilog pattern file. Do this with the following command:

write_patterns results/pat.v -replace -procfile -ext -verilog -parameter_list \ {ALL_HEADER_COMMENT "Header Line 1\nHeader Line 2"}

These lines are added to the pattern file header:

```
// Header Line 1
// Header Line 2
```

ALL_IDDQ_TESTER_CYCLE

Issuing a 1 with this keyword forces the IDDQ file to contain tester cycle counts and serial simulation times even if the pattern file being saved is a parallel load pattern file.

Usage

```
ALL_IDDQ_TESTER_CYCLE {1 | 0};
```

Issuing a 1 with this keyword forces the IDDQ file to contain tester cycle counts and serial simulation times even if the pattern file being saved is a parallel load pattern file. Not using the keyword, or using it with a 0 argument invokes the default behavior where the IDDQ file uses tester cycle counts for serial pattern formats, and vector counts and parallel load times for parallel formats.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

In the example, saving a parallel load pattern file for a particular design and using the -iddq_file, the IDDQ file has the following content:

```
/* Pattern # Cycle # Time */
0 71 7100
1 76 7600
...
```

Saving the same parallel load pattern file and using -iddq_file, but also using ALL_IDDQ_TESTER_CYCLE 1; produces an IDDQ file with the following content:

```
/* Pattern # Cycle # Time */
0 195 19500
1 324 32400
...
```

The cycle counts and times given now correspond to serially loaded tester cycle counts and simulation time stamps

ALL_IGNORE_SIMULATION_ONLY

Specifies handling of “TESSENT_PRAGMA simulation_only” annotations.

Usage

```
ALL_IGNORE_SIMULATION_ONLY {1 | 0};
```

By default, cycles surrounded by ‘annotate "TESSENT_PRAGMA simulation_only begin"’ and ‘annotate "TESSENT_PRAGMA simulation_only end"’ statements in a procedure file are included only in simulation (Verilog) output. They are eliminated from output in any other format.

Set this keyword to 1 causes the tool to handle these TESSENT_PRAGMA directives as a normal annotation rather than as a special directive that affects tool output. In this case, the annotations appear as normal annotation statements in the output.

ALL_MAX_LOOP

Sets a maximum loop count to use for a single loop. Any run loops that are larger than this value are split into multiple smaller loops, with most being the size specified by ALL_MAX_LOOP and the last loop giving the remainder.

Note

This keyword does not affect repeats or loops that come from repeated sub_procedures or loop statements within the test procedure file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Usage

```
ALL_MAX_LOOP N
```

N is a long integer value specifying the maximum loop count. The minimum value is 4, and the maximum is 2^{40}. When you do not specify this parameter, the Verilog testbench uses a maximum loop count of 2^{30}. All other pattern formats use loops with no maximum loop count.

If you specify an incorrect value, the tool issues an error message similar to the following:

```
// Error: Parameter Keyword ALL_MAX_LOOP has illegal value.
```

As with all other parameter keywords, this affects only the STI pattern formats, such as the Verilog testbench, STIL, or WGL.

Related Topics

iRunLoop

ALL_MIN_SCAN_LOAD

Setting this keyword to 1 allows a scan load to specify only scan-in or scan out when appropriate.

Usage

```
ALL_MIN_SCAN_LOAD {1 | 0};
```

By default, all pattern outputs contain scan-in and scan out data for each scan load. Setting this keyword to 1 allows a scan load to specify only scan-in or scan out when appropriate (first scan in). The default is 0.

ALL_NO_CYCLE_OPT

This keyword specifies for all outputs to turn off cycle optimizations in non-scan areas of patterns and leave all cycles in the patterns.

Usage

```
ALL_NO_CYCLE_OPT {1 | 0};
```

When set to 1 (or omitting this keyword as it is the default), this keyword specifies that cycle optimization is not applied to remove duplicate cycles from clock_sequential and capture procedures. When set to 0, this keyword allows the following optimization in cycles that are

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

part of the Capture or Clock procedures: If any two identical cycles are generated and do not measure any outputs, the duplicate cycles is removed.

ALL_NO_LOOP

Disables the use of “loop” or “repeat” statements to represent sub_procedures.

Usage

```
ALL_NO_LOOP {1 | 0};
```

The default is off (0). For more information, refer to “Disabling Sub_procedure Looping” in the *Tessent Shell User’s Manual*.

ALL_NO_PATTERN_TYPE

If set to 1, this keyword eliminates the pattern type annotation.

Usage

```
ALL_NO_PATTERN_TYPE {1 | 0};
```

If set to 1, this keyword eliminates the pattern type annotation. This keyword applies to all Vector Interface formats. The default is 0.

Caution

This keyword should only be used if the pattern type annotations are causing problems. If you set this keyword to 1, you are not able to load the generated patterns back into the Tessent tools.

The keyword should not be used with any STIL or WGL pattern file to be read back in as it removes the pattern type annotation at the start of each pattern that identifies a pattern as something other than a basic scan pattern. All STI patterns always have overlapped scan load and unload, so the unload is always present in the pattern data.

ALL_NONSCAN_CONSTANT

Setting this keyword to 1 forces all non-scan pins, excluding control and clock pins, to the constant value of the pad state for all scan patterns.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Usage

```
ALL_NONSCAN_CONSTANT {1 | 0};
```

Setting this keyword to 1 forces all non-scan pins, excluding control and clock pins, to the constant value of the pad state for all scan patterns. This keyword may be useful, as an alternative to procedure force statements, for testers that require constant pin values. The default is 0.

Note

The ALL_NONSCAN_CONSTANT keyword parameter is ignored when writing SSN patterns.

ALL_ORIGINAL_INDEX

When set to 1, saved pattern files use the original pattern index (from before filtering) for the annotations and comments prior to each pattern.

Usage

```
ALL_ORIGINAL_INDEX {1 | 0};
```

When set to 1, saved pattern files use the original pattern index (from before filtering) for the annotations and comments prior to each pattern. When set to 0 (the default), the pattern numbering in the comments and annotations is numbered in order starting at 0 from the first pattern as if the filtered patterns are a new pattern set. Note that this affects all types of pattern outputs other than Verilog because the Verilog testbench uses a built-in pattern count mechanism.

ALL_OVERRIDE_STROBE_METHOD

This keyword defines the strobing behavior (edge or window).

Usage

```
ALL_OVERRIDE_STROBE_METHOD edge | window
```

The default strobing behavior is edge strobing, unless your test procedure includes a "set_strobe_window time <*n*>" statement where <*n*> > 0, in which case the tool uses window strobing.

If you specify "edge" with this keyword, the tool uses edge strobing regardless of the setting of the set_strobe_window_time value.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If you specify “window” with this keyword, the tool behaves as follows:

- “set_strobe_window time” value > 0 — the tool uses this value for the strobing window size.
- “set_strobe_window time” value = 0 — the tool automatically computes the strobing window size, and it reports a warning during the write_patterns procedure.
- “set_strobe_window time” statement is unspecified — the tool also automatically computes the strobing window size.

ALL_PRESERVE_FORCE_X

This keyword retains or removes explicitly forced X values on ports in the pattern file.

Usage

```
ALL_PRESERVE_FORCE_X {1 | 0};
```

This keyword instructs the tool to keep explicitly forced X values on ports in the pattern file. The default for this keyword is false (0). As with all other parameter keywords, this keyword only affects the STI pattern formats, such as the Verilog testbench, STIL, or WGL.

This keyword only affects X values that are explicitly forced on pins within the test procedures. A pin that is constrained to X does not count for this, unless it is also explicitly forced to X in the test procedure. A primary input pin that is not a clock and is constrained to X, and is not forced to any other value in the test_procedures is considered to be an unused pin so it is removed from the pattern set and not affected by this new keyword unless the ALL_EXCLUDE_UNUSED is set to OFF.

If X values are present on input pins in a vector, then previous force values for this vector are copied from previous vectors in place of this X value.

Setting this keyword to 0 or false (the default) means that the tool ignores the explicitly forced X values on ports in the pattern file.

Setting this keyword to 1 or true instructs the tool to retain explicitly forced X values on ports in the pattern file.

ALL_SINGLE_QUOTED_SSN_ANNOTATIONS

Causes all design names in SSN annotations to be written with single quotes (') instead of braces ({ }).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Usage

```
ALL_SINGLE_QUOTED_SSN_ANNOTATIONS {1 | 0};
```

When you set this to 1, the tool uses single quotes for SSN annotations containing design names, as in the following example:

```
…
{   Begin_Active_Ssh_Section }
{     instance     = 'processor_core_rtl2_tessent_ssn_scan_host_1_inst' }
{     icl_instance = 'processor_core_rtl2_tessent_ssn_scan_host_1_inst' }
{     bus_width    = 2 }
…
```

The default behavior (0) uses braces instead:

```
…
{   Begin_Active_Ssh_Section }
{     instance     = {processor_core_rtl2_tessent_ssn_scan_host_1_inst} }
{     icl_instance = {processor_core_rtl2_tessent_ssn_scan_host_1_inst} }
{     bus_width    = 2 }
…
```

Some pattern translation tools may not support the syntax of having embedded braces (that is, {} pairs inside other {} pairs). Using this keyword changes the annotations to a supported syntax.

ALL_SSN_MAXIMUM_BUS_SPEED

Instructs the tool to pad the packet when needed so that the bus can operate at the maximum frequency. This keyword is supported for all formats.

When simulating SSN patterns and applying them to just one core at a time (typically when simulating core-level patterns at the core level), or when testing a few cores concurrently at a higher level, it is possible that the SSN bus does not run at the full speed it is designed to run. This is because the SSN bus would otherwise exceed the internal shift speed of the core or cores.

However, when performing simulation with timing, you should validate that the bus can operate at full speed as part of your sign-off because when you test multiple cores concurrently at the top level, the SSN bus typically runs at its maximum speed. To allow proper simulation sign-off by running the bus at full speed, this keyword pads the SSN packets as needed so that the bus can run at full speed without exceeding the maximum shift frequency of the core or cores.

Usage

```
ALL_SSN_MAXIMUM_BUS_SPEED {1 | 0};
```

By default, the tool does not pad the packet.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ALL_TIME_RESOLUTION

This keyword is used to specify the number of significant bits for floating point time numbers.

Usage

```
ALL_TIME_RESOLUTION integer;
```

This keyword is used to specify the number of significant bits for floating point time numbers. The maximum is 4, and the default is 3.

ALL_USE_CYCLE_LOOP

Uses updated terminology in the generated pattern files to prevent confusion with Tessent Diagnosis terms.

Usage

```
ALL_USE_CYCLE_LOOP {1 | 0};
```

Uses updated terminology in the generated pattern files to prevent confusion with Tessent Diagnosis terms. The previous “cycle” and “loop” terms in the pattern files are replaced with “simulation cycle” or “vector” and “tester cycle”. Set this keyword to 1 to revert back to the “cycle” and “loop” terminology.

CTL_ONE_PAT

This keyword, if set to 1, causes all pattern data to be grouped into a single STIL pattern block, instead of one pattern block per ATPG pattern. The default is 0.

Usage

```
CTL_ONE_PAT {1 | 0};
```

This keyword, if set to 1, causes all pattern data to be grouped into a single STIL pattern block, instead of one pattern block per ATPG pattern. The default is 0.

CTL_NO_PROTO

This keyword, if set to 1, disallows protocol-level macros.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Usage

```
CTL_NO_PROTO {1 | 0};
```

This keyword, if set to 1, disallows protocol-level macros. The default of 0 allows protocol-level macros.

Note

In addition to the three CTL parameter file keywords listed above, all parameter file keywords for STIL output also work with CTL output. The tool-generated CTL output can be customized using the above three parameters to be a more structural STIL output.

CTL_STIL_0

This keyword, if set to 1, causes the output to be backward compatible with the IEEE 1450.6 standard.

Usage

```
CTL_STIL_0 {1 | 0};
```

This keyword, if set to 1, causes the output to be backward compatible with the IEEE 1450.6 standard. The Environment block is dropped and the STIL header does not mention the CTL extension. The default is 0.

Note

CTL (IEEE 1450.6) is an extension of STIL that creates a standard format to describe IP core and SOC test information.

CTL_STIL_1

This keyword, if set to 1, causes the output to eliminate all 1450.6 syntax, but leaves the IEEE 1450.1 syntax.

Usage

```
CTL_STIL_1 {1 | 0};
```

This keyword, if set to 1, causes the output to eliminate all 1450.6 syntax, but leaves the IEEE 1450.1 syntax. The default is 0, which produces the full 1450.6 and 1450.1 syntax used in the CTL output.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The CTL_STIL_0 parameter file keyword can be used in the CTL output to eliminate the 1450.1 features along with all of the 1450.6 syntax.

For more information, refer to the "write_patterns -CTL" command description in the *Tessent Shell Reference Manual*.

FTDL_DESIGNER

Specifies an informational string to be used as the designer name in the DESIGNER statement at the top of the FTDL file.

Usage

```
FTDL_DESIGNER string;
```

Specifies an informational string to be used as the designer name in the DESIGNER statement at the top of the FTDL file. If not specified, the default is "ATPG".

FTDL_MAX_PAT_BLOCK

Specifies the maximum number of vector statements allowed in a single TEST block, including all shifted values and repeated vectors.

Usage

```
FTDL_MAX_PAT_BLOCK integer;
```

Specifies the maximum number of vector statements allowed in a single TEST block, including all shifted values and repeated vectors. If not specified, the default is 5 million.

FTDL_REVISION

Specifies an informational string to be used as the revision string in the REVISION statement at the top of the FTDL file.

Usage

```
FTDL_REVISION string;
```

Specifies an informational string to be used as the revision string in the REVISION statement at the top of the FTDL file. If not specified, the default is "0001".

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

FTDL_TCOMMENT

Setting this keyword to a string allows the user to override the default TCOMMENT field in the FTDL pattern with a custom entry.

Usage

```
FTDL_TCOMMENT string;
```

FTDL_TEST_NAME

Specifies the root name to be used for the test block(s) within the FTDL file. If not specified, the default root name is “FN”.

Usage

```
FTDL_TEST_NAME string;
```

Specifies the root name to be used for the test block(s) within the FTDL file. If not specified, the default root name is “FN”. The root name is combined with a test number to create each test block name.

FTDL_TEST_NUM

Specifies the starting test block number, to be appended to the root test block name.

Usage

```
FTDL_TEST_NUM integer;
```

Specifies the starting test block number, to be appended to the root test block name. For each additional test block needed in the FTDL file, the test number is incremented. If not specified, the starting test block number is 1.

FTDL_TEST_SUFFIX

Specifies a string to add as a suffix to the test block name.

Usage

```
FTDL_TEST_SUFFIX string;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Specifies a string to add as a suffix to the test block name. The total test block name is constructed from FTDL_TEST_NAME, FTDL_TEST_NUM, and optionally, FTDL_TEST_SUFFIX. The required argument *string* is the suffix that is added after the test block number. The default is to have no suffix.

FTDL_ZMODE_MES

Controls whether the FTDL file allows for Z values to be measured on output pins and output sides of bidi pins.

Usage

```
FTDL_ZMODE_MES {1 | 0};
```

Controls whether the FTDL file allows for Z values to be measured on output pins and output sides of bidi pins. If set to 0, then the “ZMODE = NOMES” statement is placed in all TEST blocks within the FTDL file, and no Z values is used on output pins. If set to 1, then the “ZMODE = MES” statement are placed in all TEST blocks, and Z values are allowed on output pins. The default is 0, NOMES.

Keywords FTDL_TEST_NAME and FTDL_TEST_NUM are used together to create the name(s) of the TEST block(s) within the FTDL file.

Keyword FTDL_MAX_VEC_BLOCK is used to set the maximum number of vector statements allowed in a single TEST block within the FTLD file.

Keywords FTDL_REVISION and FTDL_DESIGNER simply supply informational strings to be used within the FTDL file.

Keyword FTDL_ZMODE_MES controls whether ZMODE = MES, or ZMODE = NOMES, and is used in the FTDL file.

The following is an example parameter file for FTDL.

```
FTDL_TEST_NAME “MyTest”;
FTDL_REVISION “001.1”;
FTDL_DESIGNER “Smith”;
FTDL_ZMODE_MES 1;
```

SIM_ANNOTATE_QUIET

When set to 1, this keyword suppresses the Verilog testbench from displaying the annotations during simulation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Usage

```
SIM_ANNOTATE_QUIET {1 | 0};
```

When set to 1, this keyword suppresses the Verilog testbench from displaying the annotations during simulation. The default behavior is to display the annotations.

SIM_CHAIN_ERROR

Setting this parameter to 1 adds a register/signal per scan chain to the Verilog testbench.

Usage

```
SIM_CHAIN_ERROR {1 | 0};
```

Setting this parameter to 1 adds a register/signal per scan chain to the Verilog testbench. This signal may be monitored during simulation and goes to 1 when a mismatch is detected on the scan chain. The name of the register/signal is mgcdft_*chainname*_error where *chainname* is the chain name specified on the add_scan_chains command line. The default is 0.

SIM_CLOCK_MONITOR

Generates the clock monitoring controls in the Verilog testbench.

Usage

```
SIM_CLOCK_MONITOR {1 | 0}
```

Generates the clock monitoring controls in the Verilog testbench.

Issuing a 1, the default behavior, generates the clock monitoring controls. Issuing a 0 disables the generation of the clock monitoring controls.

See "Clock Monitoring During Simulation" in the *Tessent Scan and ATPG User's Manual*.

SIM_COMPARE_SUMMARY

Writes a summary of compares in the testbench at the end of simulation.

Usage

```
SIM_COMPARE_SUMMARY {1 | 0};
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Using this keyword causes the following statements to be displayed by the testbench at the end of simulation:

- Simulated <*num*> patterns
- Simulation finished at time <*sim_time*>
- Number of miscompares = <*num*>

  (Reports the number of miscompares on the real ports.)
- Number of 0/1 compares = <*num*>

  (Reports the number of compares on the real ports.)
- Number of Z compares = <*num*>

  (Reports the number of Z compares on the real ports.)
- Number of scan cell miscompares = <*num*>

  (Reports the number of miscompares on scan cells. Only present for scan chains with parallel load.)
- Number of scan chain compares = <*num*>

  (Reports the number of comparison on scan chains. Only present for scan chains with parallel load.)
- Number of chain monitor compares = <*num*>

  (Only present when SIM_PARALLEL_MONITOR is specified.)
- Number of chain monitor miscompares = <*num*>

  (Only present when SIM_PARALLEL_MONITOR is specified.)

Because serial simulation also includes all scan chain unload data on the real scan-out port rather than on scan cells, the results for the “Number of 0/1 compares” value differs between serial and parallel simulation.

SIM_COMPARE_X

When set to 1, this parameter adds code to the testbench to not mask X states, but to compare for differences.

Usage

```
SIM_COMPARE_X {1 | 0};
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When set to 1, this parameter adds code to the testbench to not mask X states, but to compare for differences. This functionality is available in both Serial and Parallel testbenches. Issuing a 0, the default behavior, causes the testbench to not include these capabilities.

Note

While issuing a mismatch if we expect X is not required for ATPG, it is necessary for LibComp verification to identify cases where the translation is pessimistic; otherwise, modeling a state such as a Tie-X does not look like a problem since you won't get mismatches.

Note

The SIM_COMPARE_X keyword parameter is ignored when writing SSN patterns.

SIM_DELAY_SCAN_RELEASE

Delays the scan release in the parallel testbench by the specified integer.

Usage

```
SIM_DELAY_SCAN_RELEASE integer;
```

This keyword and positive integer pair delay the scan release in the parallel testbench by the specified integer. The unit of the specified *integer* is the same as the timescale units. If there are multiple chains, each release is delayed by the specified *integer* amount. This keyword does not support serial testbenches or SSN designs.

Note

The SIM_STRETCH_SCAN_RELEASE keyword is an alternative and preferable way to delay the scan release event. This is because it is safer to delay the release by extending the shift period rather than to delay the release into the next shift cycle and past events occurring in the next shift cycle.

By default, the parallel testbench uses the timing from the shift procedure to produce the force, release, and shift clock events. The release occurs at the end of the shift procedure. Since the parallel testbench forces the clock events at the device boundary, but the force and release events happen at the cell boundaries, clock propagation delay can cause the clock edges to arrive after the release. SIM_DELAY_SCAN_RELEASE does not add any extra delay to the shift procedure but instead adds a delay in the event that is triggered by the end of the shift procedure, which causes the release to be delayed but does not delay any other events. During simulation, the release event in the testbench still occurs at the end of the shift procedure for SIM_DELAY_SCAN_RELEASE, but the actual "release" statement is then delayed by the specified amount after that trigger event.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You cannot specify the SIM_DELAY_SCAN_RELEASE keyword and the SIM_STRETCH_SCAN_RELEASE keyword together because the two keywords have different mechanisms for delaying the scan release event. If you use both keywords, the SIM_STRETCH_SCAN_RELEASE keyword overrides the SIM_DELAY_SCAN_RELEASE keyword.

Note

If you use SIM_DELAY_SCAN_RELEASE in the presence of SSN, the tool ignores the parameter keyword. For SSN designs, use the SIM_STRETCH_SCAN_RELEASE keyword to delay the scan release.

SIM_DIAG_FILE

Controls the generation of the failure file in the testbench.

Usage

```
SIM_DIAG_FILE {2 | 1 | 0};
```

Controls the generation of the failure file in the testbench. The failure filename is generated by appending the testbench filename with *.fail*. For example, a testbench file named *testbench.v* would generate a failure file named *testbench.v.fail*. This failure file can be one of the following formats:

- **cycle-based** — For serial testbenches of scan ATPG patterns, IJTAG retargeting, and vector_set patterns
- **pattern-based** — For all other failure files

For uncompressed ATPG, this keyword works for both serial and parallel Verilog testbenches. For compressed ATPG, the keyword only works for the serial Verilog testbenches and has no impact on the parallel testbench.

For “patterns -scan” context, the default setting is 1. For all other contexts, the default is 0. The options for this parameter are as follows:

- **0 —** No Verilog code for generating the failure file is included in the testbench. This setting maintains backward compatibility with previous testbench versions.
- **1 —** Verilog code for generating the failure file is included in the testbench, but the file is not automatically generated. To activate generation of the failure file, you must edit the testbench, setting the “_write_DIAG_file” parameter to 1.
- **2 —** Verilog code for generating the failure file is included and activated in the testbench. You can explicitly disable failure file generation by setting the “_write_DIAG_file” parameter to 0 in the testbench.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SIM_DISTRIBUTED_PAT

Turns off the enhanced Verilog testbench that allows you to choose which patterns to simulate.

Usage

```
SIM_DISTRIBUTED_PAT {1 | 0};
```

Turns off the enhanced Verilog testbench that allows you to choose which patterns to simulate. When set to 0, all saved patterns are simulated. When set to 1 (the default), this statement allows you to choose which patterns to simulate.

You can then use the STARTPAT and ENDPAT PlusArgs keywords to specify the starting and ending patterns. For example, the following runs the simulation using pattern files 2 through 10:

```
vsim simple_testbench_v_ctl -c -do vsim.do +STARTPAT=2 +ENDPAT=10
```

If you specify only the ENDPAT keyword, the simulation begins with the first pattern or file and end at the specified pattern or file. For example, the following runs the simulation for the first four patterns:

```
vsim simple_testbench_v_ctl -c -do vsim.do +ENDPAT=4
```

If you specify only the STARTPAT keyword, the simulation starts with the specified pattern or file and end at the last pattern or file. For example, the following runs the simulation for pattern files 3 through 20, where 20 is the last pattern file:

```
vsim simple_testbench_v_ctl -c -do vsim.do +STARTPAT=3
```

To simulate a single pattern, specify the same value both the start and end keywords. For example, the following runs the simulation for only pattern 8:

```
vsim simple_testbench_v_ctl -c -do vsim.do +STARTPAT=8 +ENDPAT=8
```

SIM_DUMPFILE_PATH

Specifies the path of all dump files during Verilog simulation.

Usage

```
SIM_DUMPFILE_PATH pathname;
```

Specifies the path of all dump files during Verilog simulation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

See "Verilog Plusargs" in the *Tessent Scan and ATPG User's Manual* for complete information for creating Verilog dump files.

SIM_EARLY_RELEASE

Designates the Verilog testbench to put the "release" statement immediately after the trailing edge of the first clock pulse in the shift procedure.

Usage

```
SIM_EARLY_RELEASE {1 | 0};
```

Designates the Verilog testbench to put the "release" statement immediately after the trailing edge of the first clock pulse in the shift procedure. Issuing a 0, the default behavior, places the "release" statement at the end of the cycle.

Note

The SIM_EARLY_RELEASE keyword should only be used in parallel testbenches. If you use SIM_EARLY_RELEASE with a serial testbench, the tool issues a warning and the parameter keyword is ignored.

SIM_EARLY_RELEASE_LAST

Used in conjunction with the SIM_EARLY_RELEASE_TIME keyword.

Usage

```
SIM_EARLY_RELEASE_LAST {1 | 0};
```

When multi-cycle shift procedures are present, by default SIM_EARLY_RELEASE causes the scan release to happen in the cycle that has the force_sci statement at the time specified. When the SIM_EARLY_RELEASE_LAST keyword is specified, the time specified by SIM_EARLY_RELEASE_TIME is applied to the last cycle in the multi-cycle shift procedure.

Note

The SIM_EARLY_RELEASE_LAST keyword should only be used in parallel testbenches. If you use SIM_EARLY_RELEASE_LAST with a serial testbench, the tool issues a warning and the parameter keyword is ignored.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SIM_EARLY_RELEASE_TIME

Specifies exactly what time you want the Verilog testbench “release” statement to happen using the time scale specified in the procedure file.

Usage

```
SIM_EARLY_RELEASE_TIME integer;
```

Specifies exactly what time you want the Verilog testbench “release” statement to happen using the time scale specified in the procedure file. The integer value is a time value that is required to be in the same time scale as that used in the timeplate for the shift procedure. The time specified must be later than the time of the force_sci event, but not greater than the time of the end of the cycle that the force_sci event occurs in. The time is relative to the start of the cycle.

This keyword does not replace the SIM_EARLY_RELEASE keyword, but can be used as an alternative. If both the SIM_EARLY_RELEASE and the SIM_EARLY_RELEASE_TIME keywords are specified in the parameter file, then the SIM_EARLY_RELEASE_TIME keyword takes precedence. If neither keyword is used, then the default behavior of putting the “release” statement at the end of the cycle is still used.

Note

The SIM_EARLY_RELEASE and SIM_EARLY_RELEASE_TIME statements are helpful when complex scan cells that use multiple clocks need the release to happen at a certain time prior to the end of the shift procedure, in order for the parallel load testbench to load the correct values. That is, if the forced value is held too long, a second clock pulse latches in the wrong value.

Note

The SIM_EARLY_RELEASE_TIME keyword should only be used in parallel testbenches. If you use SIM_EARLY_RELEASE_TIME with a serial testbench, the tool issues a warning and the parameter keyword is ignored.

SIM_FILL_BIDISCAN

Setting this keyword to 1 sets the bidirectional scan_input pins to the scan-in value of the first cell.

Usage

```
SIM_FILL_BIDISCAN {1 | 0};
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

By default, the bidirectional scan_input pins are set to “X’ during parallel scan loading in the Verilog testbench. Setting this keyword to 1 sets the bidirectional scan_input pins to the scan-in value of the first cell.

SIM_FORCE_Z_AT_CYCLE

Controls when a bidi pin switches to a Z state in the Verilog testbench.

Usage

```
SIM_FORCE_Z_AT_CYCLE {1 | 0};
```

Controls when a bidi pin switches to a Z state in the Verilog testbench. The default behavior is that when a bidi pin is going to change to a Z value during a given simulation vector, then the Z is forced at time 0 for that vector. However, when you set the keyword value to 0, the Z value applies to the bidi pin at the force time specified in the timeplate rather than at time 0. If the force time happens after the internal driver for the bidi pin turns on, bus contention is possible.

SIM_INCLUDE

Adds an include statement into the Verilog testbench after the comments and before the first module.

Usage

```
SIM_INCLUDE filename;
```

Adds an include statement into the Verilog testbench after the comments and before the first module. The contents of the filename specified must contain legal Verilog statements and syntax. This statement is used only in the Verilog outputs. The Default behavior is to not add any include statements.

SIM_INSTANCE_ARGUMENTS

Adds extra arguments to the testbench DUT instance statement for the DUT in the testbench.

Usage

```
SIM_INSTANCE_ARGUMENTS string ;
```

This keyword passes extra arguments in a quoted string to parameters in the DUT module. You must correctly format the arguments in the string. The tool does not check string contents. If the contents do not match the DUT module parameters, the testbench fails to compile in Verilog.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

Command Line:

write_patterns pat.v -Verilog -replace -parameter_list SIM_INSTANCE_ARGUMENTS "#(BUS_WIDTH = 8, DATA_WIDTH = 64)"

Parameter File:

```
SIM_INSTANCE_ARGUMENTS "#(ARG1 = 20, ARG2 = 40)" ;
```

SIM_INSTANCE_NAME

Specifies the instance name for your design, to be used within the testbench.

Usage

```
SIM_INSTANCE_NAME string;
```

Specifies the instance name for your design, to be used within the testbench.

For example, if your current design is simple_chain2 and you want to specify the instance name presy, this is the entry in the parameter file:

```
SIM_INSTANCE_NAME presy ;
```

And this is the testbench:

```
//   top_module_name   = simple_chain2_pat_par_v_ctl
module simple_chain2_pat_par_v_ctl;


simple_chain2 presy (.reset(reset), .clk(clk), .in1(in1),
     .scan_in1(scan_in1), .scan_en(scan_en), .out0(out0),
     .out1(out1), .out2(out2), .out3(out3), .all0(all0),
     .all1(all1), .w1out(w1out), .scan_out1(scan_out1));
```

SIM_KEEP_PATH

Causes the Verilog testbench to use the same path as the testbench name for reading and writing new files.

Usage

```
SIM_KEEP_PATH {1 | 0};
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Causes the Verilog testbench to use the same path as the testbench name for reading and writing new files. When this is set to 0, the default behavior, the Verilog testbench uses the current directory for reading and writing files.

Use the +NEWPATH and +NEWOUTPATH Verilog plusargs to override this setting and specify different paths for reading and writing files, respectively. For more information about Verilog plusargs, refer to the topic "Verilog Plusargs" in the *Tessent Scan and ATPG User's Manual*.

SIM_MASK_FILE

Controls the generation of the mask file, which captures simulation mismatches, in the testbench.

Usage

```
SIM_MASK_FILE {2 | 1 | 0};
```

The mask filename is generated by appending the testbench filename with ".mask". For example, a testbench file named *testbench.v* would generate a mask file named *testbench.v.mask*.

For "patterns -scan" context, the default setting is 1. For all other contexts, the default is 0.

- **0** — No Verilog code for generating the mask file is included in the testbench. This setting maintains backward compatibility with previous testbench versions.
- **1** — Verilog code for generating the mask file is included in the testbench, but the file is not automatically generated. To activate generation of the mask file, you must edit the testbench, setting the "_write_MASK_file" parameter to 1.
- **2** — Verilog code for generating the mask file is included and activated in the testbench. You can explicitly disable mask file generation by setting the "_write_MASK_file" parameter to 0 in the testbench.

Note

The simulation mismatches for the ChainTest patterns are recorded in the SIM_MASK_FILE, but they are commented out.

SIM_MIN_NAME_FILE

Controls the number of *.name* output files generated when saving patterns as a Verilog testbench.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Usage

```
SIM_MIN_NAME_FILE {1 | 0};
```

Controls the number of *.name* output files generated when saving patterns as a Verilog testbench. Setting this keyword to 0 causes the tool to generate a separate set of *.name* files for each testbench file. When using the -maxloads switch for large designs with many scan chains and patterns, this generates a very large number of files. Setting this keyword to 1 reduces the number of output files by generating only one set of *.name* files for a design. The same *.name* files can be used for all generated Verilog testbenches for that design.

This parameter file keyword is on by default when you use the write_patterns command with the -maxloads switch. It is off by default otherwise.

SIM_MISCOMPARE_LIMIT

Sets a limit in the Verilog testbench for the number of miscompares to allow before the simulation terminates.

Usage

```
SIM_MISCOMPARE_LIMIT integer;
```

Sets a limit in the Verilog testbench for the number of miscompares to allow before the simulation terminates. By default, the testbench runs the simulation until all saved patterns are simulated, regardless of how many miscompares occur. This keyword causes the simulation to abort upon reaching the specified miscompare limit.

By default, the SIM_MISCOMPARE_LIMIT keyword is set to 0 (no limit).

SIM_MODULE_INCLUDE

Similar to the SIM_INCLUDE parameter, except it adds the include statement within the module definition for the testbench instead of after the comments and before the first module.

Usage

```
SIM_MODULE_INCLUDE filename;
```

Similar to the SIM_INCLUDE parameter, except it adds the include statement within the module definition for the testbench instead of after the comments and before the first module. The *filename* can be any valid file path. The contents of the specified file must contain legal Verilog statements and syntax. This statement is used only in the Verilog outputs. The default behavior is to not add any include statements.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SIM_MODULE_NAME

Specifies a string that overrides the module name of the DUT in the Verilog simulation testbench.

Usage

```
SIM_MODULE_NAME string;
```

Specifies a string that overrides the module name of the DUT in the Verilog simulation testbench.

SIM_NAME_FILE

Allow multiple testbenches to use the same name files regardless of the name of the testbench.

Usage

```
SIM_NAME_FILE {1 | 0};
```

When writing out multiple testbenches where only one is meant to be compiled and used, and the other data sets are used (*.cfg* and *.vec* files) to simulate patterns using the one common testbench, this keyword should be used so that only one set of name files is written (*.po.name* and *.chain.name*) to save on disk space.

SIM_NO_CHAINNAME_FILE

This keyword controls the creation of chain name files. Setting this keyword to 1 prevents the generation of chain name files.

Usage

```
SIM_NO_CHAINNAME_FILE {1 | 0};
```

When saving a Verilog testbench, the tool, by default, generates a single chain name file with all the scan cell names. The names are loaded into one register in the testbench so that when mismatches occur during simulation, the testbench reports the mismatches by cell name, rather than by scan chain name and cell number. To avoid exceeding the limit of the register size, if there are more than two million scan cells, the tool default changes to automatically create one chain name file per scan chain.

When the chain name files are created, there is an increase in the number of files written, the disk space used, and the memory footprint of the Verilog testbench. This impacts simulation time when the design is large and when, due to memory constraints, the Verilog simulator needs to swap memory while running. The performance impact of chain name file creation is due to

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

increased memory use rather than extra processing within the testbench during simulation, and influenced by design size and amount of memory available.

Setting this keyword to 1 prevents the tool from generating the chain name files. It also reduces the Verilog testbench memory footprint, the number of files written, and amount of data stored on the disk, and potentially speeds up simulation.

SIM_NO_INTERNAL_COMPARE

Enables or disables internal compares for the internal expect statements in the Verilog testbench.

Usage

```
SIM_NO_INTERNAL_COMPARE {1 | 0};
```

If there are expect statement in the procedure file on internal registers, then these expect statements generates internal compares in the generated Verilog testbench. When this parameter is set to 1 or true, the generated Verilog testbench does not have internal compares for the internal “expect” statements.

The default is 0 or false.

SIM_NO_MEASURE_IN

In the Verilog testbench, when a value is being forced on a bidirectional pin (as in input mode), the testbench also tries to measure the same value on the bidirectional pin when a measure happens.

Usage

```
SIM_NO_MEASURE_IN {1 | 0};
```

In the Verilog testbench, when a value is being forced on a bidirectional pin (as in input mode), the testbench also tries to measure the same value on the bidirectional pin when a measure happens. Setting this keyword to 1 causes the testbench to not measure input values on the output side of bidirectional pins. The default is 0.

SIM_PARALLEL_DELAY_SHIFT_CLOCKS

Delays the leading edge of all shift clocks in all shift procedures by the specified number of timescale units.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Usage

```
SIM_PARALLEL_DELAY_SHIFT_CLOCKS {integer};
```

Delays the leading edge of all shift clocks in all shift procedures by the specified number of timescale units. Any events occurring after the leading edge of the first shift clock is also delayed to maintain the event order. The specified number must be a positive integer. If the specified delay causes any event to occur beyond the period of the timeplate, the tool ignores this keyword.

This keyword is valid only for Verilog parallel testbench generation.

SIM_PARALLEL_EARLY_FORCE_AND_MEASURE

Setting this keyword to 1 or true moves the measure_sco event to time 0 and the force_sci event to time 1.

Usage

```
SIM_PARALLEL_EARLY_FORCE_AND_MEASURE {1 | 0 | true | false};
```

Setting this keyword to 1 or true moves the measure_sco event to time 0 and the force_sci event to time 1. By default (0 or false), the measure_sco event occurs at the measure_po time specified in the timeplate for the shift procedure. The force_sci event then occurs one timescale unit later, regardless of the force_pi time specified in the timeplate.

This keyword is valid only for Verilog parallel testbench generation.

SIM_PARALLEL_MONITOR

Setting this keyword to 1 for a Verilog serial testbench compares all scan cells in internal chains after scan loading and before scan unloading. It reports mismatches on specific internal scan cells when they occur.

Usage

```
SIM_PARALLEL_MONITOR {1 | 0};
```

Setting the SIM_PARALLEL_MONITOR keyword to 1 enables monitoring of the internal scan structures to verify the correct scan values are loaded into the internal scan chains before capture. This helps to identify scan chain loading issues instead of capture clock problems.

Setting this keyword to 1 creates additional *.vec* files with the Verilog serial testbench, which contains the parallel monitor pattern data.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Setting SIM_PARALLEL_MONITOR to 1 automatically sets SIM_MISCOMPARE_LIMIT to 100. You can change this value using the SIM_MISCOMPARE_LIMIT keyword.

By default, the SIM_PARALLEL_MONITOR keyword is set to 0.

This keyword is valid only for Verilog serial testbenches. Setting it for a parallel testbench results in a warning and the keyword being disabled.

Note

The testbench that this functionality produces contains SystemVerilog constructs, so you must compile it with SystemVerilog enabled. You can generally do this by writing the pattern file with a *.sv* extension instead of *.v*. Consult your compiler documentation for further information on how to enable SystemVerilog.

Related Topics

Parallel Scan Cell Monitoring During Serial Scan Pattern Simulation [Tessent Scan and ATPG User's Manual]

SIM_POST_SHIFT

Specifies the absolute number of independent post shifts to use during simulation.

Usage

```
SIM_POST_SHIFT integer;
```

Specifies the absolute number of independent post shifts to use during simulation. That is, if the load_unload procedure already has one post shift, setting this keyword to “2” causes two post shifts to be used rather than three.

If you specify a negative number or zero, the tool ignores this keyword. If you specify a number larger than the total number of shifts, the tool sets the number of post shifts to the total number of shifts minus one.

You can use this keyword to generate a parallel simulation testbench that applies a parallel load followed by N number of serial shift cycles. When used in the “patterns -scan” context, SIM_POST_SHIFT serial shift cycles are applied by forcing the scan-in values onto the scan pin of the design. You can also use this keyword to generate an EDT parallel simulation testbench; in this case, the serial shifts are performed on the EDT internal scan chains and the testbench forces values onto the scan ports driven by the decompressor.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

However, you cannot use this keyword to generate an EDT parallel simulation testbench in which the serial inputs are shifted in through the EDT channels because serial simulation through the decompressor and other EDT logic is not supported by parallel testbench.

SIM_PRE_SHIFT

Specifies the absolute number of independent pre shifts to the load_unload procedure (prior to main shift) to use during simulation.

Usage

```
SIM_PRE_SHIFT integer;
```

Specifies the absolute number of independent pre shifts to the load_unload procedure (prior to main shift) to use during simulation. That is, if the load_unload procedure already has one pre shift, setting this keyword to “2” causes two pre shifts to be used rather than three.

If you specify a negative number or zero, the tool ignores this keyword. If you specify a number larger than the total number of shifts, the tool sets the number of pre shifts to the total number of shifts minus one.

SIM_PRECISION

Specifies the timescale precision.

Usage

```
SIM_PRECISION string;
```

Specifies the timescale precision. The string is a number and a unit, for example:

```
SIM_PRECISION 1ns;
SIM_PRECISION 10ms;
```

The units supported are ms, us, ns, ps, and fs.

The tool checks this value to verify that it is less than or equal to the Timescale unit from the procedure file and outputs this value in the testbenches.

SIM_SERIAL_CHAIN

Creates a testbench that applies chain test patterns in serial mode and all other patterns in parallel mode (when set to 1).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Usage

```
SIM_SERIAL_CHAIN {1 | 0};
```

Creates a testbench that applies chain test patterns in serial mode and all other patterns in parallel mode (when set to 1). The keyword is set to 0 by default.

Note

The SIM_SERIAL_CHAIN keyword should only be used in parallel testbenches and should only be used without compression. If you use SIM_SERIAL_CHAIN with a serial testbench or when writing compressed scan patterns, the tool issues a warning and the parameter keyword is ignored.

SIM_SHIFT_DEBUG

Specifies for the serial Verilog testbench to use special test patterns that contain a parallel observe of the entire scan chain for every shift, while serially shifting the patterns.

Usage

```
SIM_SHIFT_DEBUG integer;
```

Specifies for the serial Verilog testbench to use special test patterns that contain a parallel observe of the entire scan chain for every shift, while serially shifting the patterns.

For debugging purposes only, this format makes it easier to pinpoint precisely which scan cell in a chain is causing a problem. Issuing a 0, which is the default behavior, causes the serial Verilog testbench to use test patterns that do not contain a parallel observe of the entire scan chain for every shift.

An integer value of 1 causes the testbench to shift serially and do a parallel compare for all shifts. An integer value of 2 or greater causes the testbench to shift serially and do parallel compares on the first 2 or greater shifts. This is done to reduce the amount of data in the pattern files.

The Verilog patterns you create using the keyword SIM_SHIFT_DEBUG typically take more time for the tool to create than other kinds of patterns and contain large amounts of data (so file sizes tend to be huge). Also, because they simulate serially, the patterns take more time to simulate. Be sure these are acceptable trade-offs before trying to use these patterns.

The tool ignores the SIM_SHIFT_DEBUG keyword unless you include the “write_patterns -Serial” switch. Also, if you do not include the -End or “-pattern_sets chain” options, the tool defaults to “-End 0”, saving chain test patterns and the first scan pattern. To save chain test patterns and a different set of scan patterns, use the -Begin and/or -End switches to specify the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

range of scan test patterns to save. To save only chain test patterns, include the "-pattern_sets chain" option.

Note that this functionality is available only for patterns created in the "patterns -scan" context and EDT bypass patterns (uncompressed).

SIM_SSN_MAXIMUM_BUS_SPEED

Instructs the tool to pad the packet when needed so that the bus can operate at the maximum frequency. This keyword applies only to the Verilog testbench.

When simulating SSN patterns and applying them to just one core at a time (typically when simulating core-level patterns at the core level), or when testing a few cores concurrently at a higher level, it is possible that the SSN bus does not run at the full speed it is designed to run. This is because the SSN bus would otherwise exceed the internal shift speed of the core or cores.

However, when performing simulation with timing, you should validate that the bus can operate at full speed as part of your sign-off because when you test multiple cores concurrently at the top level, the SSN bus typically runs at its maximum speed. To allow proper simulation sign-off by running the bus at full speed, this keyword pads the SSN packets as needed so that the bus can run at full speed without exceeding the maximum shift frequency of the core or cores.

Usage

```
SIM_SSN_MAXIMUM_BUS_SPEED {1 | 0};
```

By default, the tool does not pad the packet.

Related Topics

ALL_SSN_MAXIMUM_BUS_SPEED

SIM_SSN_TIMING_CHECKS

Controls whether the SSN shift and capture timing checks are active in Verilog testbench simulations.

Usage

```
SIM_SSN_TIMING_CHECKS {1 | 0};
```

By default, timing checks for shift and capture are excluded when writing SSN patterns into a Verilog testbench using the write_patterns command.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Capture mode timing checks only support internal_capture.

The following example shows how to turn on the SSN shift timing checks:

```
write_patterns example.v -verilog -serial -pattern_set scan
-parameter_list {SIM_SSN_TIMING_CHECKS 1}
```

Tip

If the Verilog testbench already has the SSN timing checks included, you can still avoid simulating them using the following simulation plusarg:

```
vlog example.v +define+TESSENT_SKIP_SSN_TIMING_CHECKS
```

SIM_STATUS_MSG

Specifies the frequency of status messages output from the Verilog simulation testbenches.

Usage

```
SIM_STATUS_MSG integer;
```

Specifies the frequency of status messages output from the Verilog simulation testbenches. This keyword accepts an integer that specifies the interval (number of patterns) to wait before outputting a message. The simulation message is “Simulated *n* patterns.”

SIM_STRETCH_SCAN_RELEASE

Delays the release scan event in the parallel Verilog testbench by stretching the end of the shift cycle.

Usage

```
SIM_STRETCH_SCAN_RELEASE integer;
```

This keyword and positive integer pair delay the release scan event in the parallel Verilog testbench by stretching the end of the shift cycle by the amount specified by the *integer* amount. The unit of the specified *integer* is the same as the timescale units. If there are multiple chains, each release scan event is delayed by the specified *integer*.

When writing the SSN parallel testbench for simulation, the tool automatically sets the keyword value to 50% of the shift period unless you specify a different keyword value. This fixes any simulation mismatch issues caused by the short time in the shift timeplate between the trailing edge of the clock and the end of the period. The stretch scan release value adds that time to the end of the main shift period, which allows for more time between the trailing edge of the clock

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

and the release of the scan forces. This does not affect the timing of any post shifts after the main shift.

In an SSN parallel testbench, the default scan release of internal scan pins is 1 unit before the measure_po of the next shift cycle. If you need the timing to be further released, use the SIM_STRETCH_SCAN_RELEASE value to override the computed value for SSN patterns. This keyword affects the timing of when to release the scan cell input values forced in the main shift of a parallel testbench. And it is unrelated to the timing used for other internal signals, including scan chain inputs.

SIM_TIMEPLATE_COMM

Turns on timeplate comments in the vector file.

Usage

```
SIM_TIMEPLATE_COMM {1 | 0};
```

Turns on timeplate comments in the vector file. The default is off (0). When you set this parameter to 1, the tool adds comments for the timeplate to the vector file. The format of the timeplate comment is:

```
// Timeplate: <timeplate_name>
```

SIM_TMP_REG_LENGTH

Changes the default length of the NEWPATH variable used in the generated testbench.

Usage

```
SIM_TMP_REG_LENGTH integer;
```

Overrides the default (512) limit for the NEWPATH variable used for filenames in the testbench.

If you specify a value less than the path of the generated testbench, the tool ignores this value and issues a warning similar to the following:

```
//  Warning: Ignored SIM_TMP_REG_LENGTH of 12, because its value cannot be
//  less than 54, the length of the file name
//  'results/user_specified_path_length_that_is_too_short.v'.
```

SIM_TOP_NAME

Specifies a new top level module name for the Verilog testbench.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Usage

```
SIM_TOP_NAME string;
```

Specifies a new top level module name for the Verilog testbench. The default behavior is to generate a default top level module name based on the design name and the name of the pattern file.

SIM_UNCOMPRESS_COMMAND

Defines the uncompress command for the Verilog testbench to use instead of the default.

Usage

```
SIM_UNCOMPRESS_COMMAND string ;
```

Examples

The following example uses **my_gzip** as the uncompress command in the Verilog testbench *top.v*:

write_patterns top.v -parameter_list {SIM_UNCOMPRESS_COMMAND my_gzip}

SIM_VECTOR_COMM

Set this keyword to 1 to turn on vector type comments in the vector file, the default is off.

Usage

```
SIM_VECTOR_COMM {1 | 0};
```

Set this keyword to 1 to turn on vector type comments in the vector file, the default is off. The format of the vector type comment is:

```
// Vector type: vector_type_string
```

Where *vector_type_string* is one of the vector types listed in Table A-1.

Table A-1. Vector Types

Vector Type	Definition
SSN_SETUP	Vector from ssn_setup procedure.
SSN_END	Vector from ssn_end procedure.
TEST_SETUP	Vector from test_setup procedure.
TEST_END	Vector from test_end procedure.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table A-1. Vector Types (cont.)

Vector Type	Definition
SEQ_TRANSPARENT	Vector from seq_transparent procedure.
CLOCK_PROCEDURE	Vector from clock procedure.
CLOCK_PROCEDURE_RESTORE	Vector from clock procedure that uses restore_bidi event.
LOAD_UNLOAD	Vector from load_unload procedure.
SHIFT	Vector from shift procedure.
SINGLE_SHIFT	Single shift vector applied in load_unload procedure.
SHIFT_EXTRA	Vector from multicycle shift procedure that does not contain the force_sci and measure_sco events.
SHADOW_CONTROL	Vector from shadow_control procedure.
MASTER_OBSERVE	Vector from master_observe procedure.
SHADOW_OBSERVE	Vector from shadow_observe procedure.
SKEW_LOAD	Vector from skew_load procedure.
PRIMARY[1]	Vector from a non-scan procedure such as capture, named capture, clock_sequential, or ram_sequential.
PRIMARY_MEASURE[1]	Vector from a non-scan procedure that measures primary outputs.
PRIMARY_CLOCKPO[1]	Vector from a clock_po procedure.

1. In addition to the above, the “PRIMARY” vector types can have “_IDDQ” appended to the end to indicate an iddq measure point.

SIM_VECTYPE_SIGNAL

When set to 0, specifies that all vector types in the currently active procedure be represented in the pattern file as a signal, and that these signals be “1” when that vector type is active.

Usage

```
SIM_VECTYPE_SIGNAL {1 | 0 | 2};
```

When set to 0, specifies that all vector types in the currently active procedure be represented in the pattern file as a signal, and that these signals be “1” when that vector type is active. This allows you to specify the patterns for tracing and viewing so you can see what type of vector is

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

currently active at any specified moment within a simulation. For an example of using this keyword, refer to "Simulation Data Analysis" in the *Tessent Scan and ATPG User's Manual*.

When set to 1, this keyword adds an encoded vector type to the Verilog or VHDL testbench in the bus MGCDFT_VECTYP[0-3]. Table A-2, lists the signal or variable names added to the testbench that map to the bus encoding. Setting the keyword to 0 causes the testbench to not include these capabilities.

Table A-2. Translation Table

MGCDFT_VECTYP[0-3] Encoding	Signal/Variable
0001	mgcdft_test_setup
0010	mgcdft_load_unload
0011	mgcdft_shift
0100	mgcdft_single_shift
0101	mgcdft_shift_extra
0110	mgcdft_shadow_control
0111	mgcdft_master_observe
1000	mgcdft_shadow_observe
1001	mgcdft_skew_load
1010	mgcdft_seq_transparent
1011	mgcdft_launch_capture
1100	mgcdft_shift
1101	mgcdft_clock_proc
1110	mgcdft_clock_proc_restore
1111	mgcdft_test_end
0000	mgcdft_unknown

Note

Adding signals indicating vector type to the Verilog testbench enables you to see what type of vector is currently active at any specified moment within a simulation and can simplify simulation mismatch debugging.

Setting this keyword to 2 (the default), which is available only with a Verilog testbench, also adds the encoded vector types (default behavior). Additionally, the testbench has the following _procedure_strings, which can be added to your WAVE display to show the currently active Procedure Type.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

CLOCK_PROC	SEQ_TRANSPARENT	SINGLE_SHIFT
CLOCK_PROC_RES	SHADOW_CONTROL	SKEW_LOAD
LAUNCH_CAPTURE	SHADOW_OBSERVE	TEST_SETUP
LOAD	SHIFT	TEST_END
MASTER_OBSERVE	SHIFT_EXTRA	UNKNOWN

STIL_2005_SCAN

Issuing a 1 with this keyword is similar to the STIL_STRUCTURAL keyword except it also uses the IEEE 1450.1 ScanStructures block.

Usage

```
STIL_2005_SCAN {1 | 0};
```

Issuing a 1 with this keyword is similar to the STIL_STRUCTURAL keyword except it also uses the IEEE 1450.1 ScanStructures block. The default is 0, which produces the existing IEEE 1450.0 ScanStructures block.

With the STIL_2005_SCAN keyword, the CTL output uses the 1450.1 ScanStructures block by default. You can use the CTL_STIL_0 parameter file keyword in the CTL output to eliminate the new 1450.1 features along with all of the 1450.6 syntax. The parameter file keyword CTL_STIL_1 can be used to eliminate all 1450.6 syntax but leave the 1450.1 syntax.

Note that the ScanStructures in your STIL structure must match the compressed structure.

In order to use this output, you must use the "write_patterns -STIL1999" command.

STIL_ADD_TIME_0_FORCE_PREVIOUS_EVENT

Adds a "force previous" event on the STIL waveform at time 0.

Usage

```
STIL_ADD_TIME_0_FORCE_PREVIOUS_EVENT {1 | 0};
```

By default, STIL waveforms do not include "force previous" events at time 0. Set this keyword to 1 adds "force previous" events at time 0 on the waveform, such as in the following, where the event is highlighted:

```
input_time_gen_1  { 01NZ { '0ns' P; '1ns' D/U/N/Z; }}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

STIL_ADD_TIME_0_MEASURE_OFF_EVENT

Adds measure events from time 0 to the first strobing edge to the STIL waveform of the output signals.

Usage

```
STIL_ADD_TIME_0_MEASURE_OFF_EVENT {1 | 0};
```

By default, output STIL waveforms do not include measure events before the first strobing edge. Set this keyword to 1 includes these events in the STIL file.

STIL_ALLOW_SCAN_BIDI_WAVEFORM

Controls whether the tool always uses a full or reduced WaveformChar set for bidi scan ports.

Usage

```
STIL_ALLOW_SCAN_BIDI_WAVEFORM {1 | 0} ;
```

By default, the tool uses the full WaveformChar set needed for bidi ports in STIL1999, STIL2005, and CTL pattern files only when necessary. It checks to see whether it can safely treat bidi scan input or output ports as only input or only output in the STIL WaveformTable. If any procedure events would attempt to use a bidi port in a manner that a reduced WaveformChar set does not support, it uses the full set.

Setting this keyword to 1 forces the tool to skip this automatic determination process and treat all bidi ports, including scan bidi ports, as full bidi ports with a full WaveformChar set in the STIL file, supporting all bidi activity.

Setting this keyword to 0 forces the tool to always use the reduced WaveformChar set for bidi scan ports when writing STIL pattern files, even if this reduced set would cause an error in the pattern file, as though the check for treating the ports as only input or output had passed. This can be useful in situations where it is important to conserve memory.

In cases where you have specified options that skip procedures in the pattern data, such as using “-test_setup off” or “-test_end off”, the process for automatically determining the proper WaveformChar set ignores the omitted procedures.

When this keyword is 0 and the tool requires a WaveformChar for a bidi port that is not specified in the pattern file, it reports an error similar to the following:

```
// Error:  Bidi port '<port_name>' being used as {input|output} only
// needs to use a <WaveformChar> waveform character.
//    This occurred in procedure <procedure_name> at cycle <number>.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

or:

```
// Error:  Bidi port '<port_name>' being used as {input|output} only
// needs to use a <WaveformChar> waveform character.
//    This occurred in an ATPG capture cycle, please check constraints on
//    tristate enable ports.
```

Examples

The normal WaveformChar set for a bidi port permits all possible operations for the port. The following is a sample Waveform definition for a bidi pin in the WaveformTable:

```
bidi_time_gen_0    { LHXT { '0ns' Z; '30ns' L/H/X/T; }
                     01Z  { '0ns' D/U/Z; } }
```

The green characters on both lines are the WaveformChars. For the first line, they correspond to measure events at 30 ns for compare low (L), compare high (H), compare unknown (X), and compare high-impedance (T). At 0 ns, the input side of the bidi port is forced off (Z). For the second line, the 01Z WaveformChars correspond to force down (D), force up (U), and force off (Z) at 0 ns.

A scan bidi port treated as only input uses a definition like the following (reduced set):

```
bidi_time_gen_1    {  01 { '0ns' D/U; } }
```

A scan bidi port treated only as output uses a definition like the following (reduced set):

```
bidi_time_gen_2    { LHX { '30ns' L/H/X; }
```

STIL_ANNOTATION_INTERVAL

Controls the annotation frequency in STIL pattern files.

Usage

```
STIL_ANNOTATION_INTERVAL integer;
```

Description

When a callback writes patterns, the write_pattern command adds annotations to the STIL files similar to the following:

```
ANN {* Pattern:0 Vector:6 TesterCycle:952 *}
```

Setting STIL_ANNOTATION_INTERVAL to a value less than one suppresses these annotations. Setting this parameter to a positive integer <*n*> causes the callback to write annotations after every <*n*>th vector.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For IJTAG patterns, the default value is one (annotations for every vector). For all other patterns, the default value is zero, meaning that the STIL files are not annotated by default.

STIL_CHAIN_CELLS

Setting this keyword to 0 removes the ScanCells list from the ScanChain definition in the ScanStructures block when saving non-compressed scan patterns that use scan chains.

Usage

```
STIL_CHAIN_CELLS {1 | 0};
```

Setting this keyword to 0 removes the ScanCells list from the ScanChain definition in the ScanStructures block when saving non-compressed scan patterns that use scan chains. The following example shows a ScanStructures block without the ScanCells list:

```
ScanStructures {
   ScanChain chain1 {
      ScanLength 100;
      ScanInversion 0;
      ScanIn "sci";
      ScanOut "sco";
   }
}
```

By default (1), the scan cell names are listed following the ScanLength statement.

For compressed patterns, see the STIL_CHANNEL_CELLS keyword.

STIL_CHANNEL_CELLS

Enables the listing of artificial scan cell names in the ScanStructures block when saving compressed patterns that use EDT channels.

Usage

```
STIL_CHANNEL_CELLS {1 | 0};
```

Enables the listing of artificial scan cell names (for example, “cell1”, “cell2”, …) in the ScanStructures block when saving compressed patterns that use EDT channels. In EDT patterns, references are to channel I/O, so scan cells are irrelevant. The listing of artificial cell names may be useful if your STIL parser requires that scan cell names be listed in the ScanStructures block.

By default (0), the artificial scan cell names are not listed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For non-compressed patterns, see the STIL_CHAIN_CELLS keyword.

STIL_CLOCK_WFC_PULSE

This parameter, used with the write_patterns command, controls the waveform characters of a clock signal in the generated STIL pattern file.

Usage

```
STIL_CLOCK_WFC_PULSE {TRUE | FALSE};
```

When STIL_CLOCK_WFC_PULSE is FALSE or not set at all, the waveform character set for clock signals is “01”. For a positive pulse, “1” indicates the clock signal is ON for the cycle and “0” is OFF for the cycle.

```
Timing STUCK_timing {
  WaveformTable tset_tp_shift_capture  {
    Period '32ns' ;
    Waveforms  {
      PI_grp_1  { 01 { '0ns' D; '10ns' D/U; '22ns' D;}}// positive pulse
      PI_grp_2  { 01 { '0ns' U; '10ns' D/U; '22ns' U;}}// negative pulse
    }
  }
}
```

When STIL_CLOCK_WFC_PULSE is TRUE, the waveform character set for clock signals is “0P” or “N1”. The waveform character set for a positive pulse is “0P”. The “P” indicates the clock signal is ON and “0” is OFF for the cycle.

The waveform character set for a negative pulse is “N1”. The “N” indicates the clock signal is ON and “1” is OFF for the cycle.

```
Timing STUCK_timing {
  WaveformTable tset_tp_shift_capture  {
    Period '32ns' ;
    Waveforms  {
      PI_grp_1  { 0P { '0ns' D; '10ns' D/U; '22ns' D;}}// positive pulse
      PI_grp_2  { N1 { '0ns' U; '10ns' D/U; '22ns' U;}}// negative pulse
    }
  }
}
```

STIL_DUAL_MODE

Enables dual mode support for IJTAG patterns that are in the STIL format.

Usage

```
STIL_DUAL_MODE {1 | 0};
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Turns on (1) dual mode support for IJTAG patterns that are in the STIL format.

You use this parameter to ensure the IJTAG STIL patterns support dual-mode, where two vectors can be executed by one ATE cycle. To support dual mode, the number of vectors before the loop, inside the loop, between loops, and after the loop must be a multiple of two.

To support this requirement, the tool may partially or fully unroll loops and add padding to the end of the test. Added padded vectors have all outputs masked to X and all clocks turned off to ensure that they do not affect the timing or functionality of the test.

To set to a multiple of 4, see the STIL_QUAD_MODE parameter keyword.

STIL_EXCLUDE_UNUSED_BUS_BITS

Instructs the STIL writer to not include unused bus bits in the STIL pattern file.

Usage

```
STIL_EXCLUDE_UNUSED_BUS_BITS {1 | 0};
```

You use this parameter to control the presence of unused bus bits in the STIL pattern file. The default is 0.

Note

STIL patterns generated with STIL_EXCLUDE_UNUSED_BUS_BITS=1 cannot be converted to a Verilog testbench with stand-alone Siemens EDA or third-party tools, because the information about the unused bus bits is not available in the pattern file. They can, however, be read in to Tessent Shell with read_patterns.

Related Topics

WGL_EXCLUDE_UNUSED_BUS_BITS

STIL_FULL_CHAIN

Causes the tool to shorten the list of scan cells by the number of pre shifts and post shifts.

Usage

```
STIL_FULL_CHAIN {1 | 0};
```

Causes the tool to shorten the list of scan cells by the number of pre shifts and post shifts. By default (1), all cells in the scan chain are listed in the STIL file regardless the number of pre shifts and post shifts.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

STIL_IDDQ_UPPERCASE

If this is set to 1 or true, IDDQ test points in the STIL file uses the term IDDQTestPoint, if 0 or false, which is the default, the STIL file uses IddqTestPoint.

Usage

```
STIL_IDDQ_UPPERCASE {1 | 0};
```

Some STIL parsers only accept one or the other.

STIL_MIN_QUOTE

Issuing a 1 with this keyword is for STIL2005 and CTL outputs, and causes the tool to write the STIL filebusing minimum quoting of signal names that use reserved characters.

Usage

```
STIL_MIN_QUOTE {1 | 0};
```

In addition to controlling the filebusing for STIL2005 and CTL outputs, any existing STIL1999 output can be changed to use the same minimum quoting rules. The default is 0 for the STIL1999 output.

Example

STIL99 default (STIL_MIN_QUOTE=0)

```
Signals
{ "tck" In; "tms" In; "trst" In; "tdi" In; "in_west_dot6_p" In;
"in_west_dot6_n" In; "tdo" Out; "io_west"[3] InOut; "io_west"[2]
InOut; "io_west"[1] InOut; "io_west"[0] InOut; "io_north"[3]
InOut; "io_north"[2] InOut; "io_north"[1] InOut; "io_north"[0]
InOut; "io_east1"[3] InOut; "io_east1"[2] InOut; "io_east1"[1]
InOut; "io_east1"[0] InOut; "io_east2"[3] InOut; "io_east2"[2]
InOut; "io_east2"[1] InOut; "io_east2"[0] InOut; "io_south"[3]
InOut; "io_south"[2] InOut; "io_south"[1] InOut; "io_south"[0]
InOut; }
```

STIL99 (STIL_MIN_QUOTE=1)

```
{   tck In; tms In; trst In; tdi In; in_west_dot6_p In;
in_west_dot6_n In;   tdo Out;   io_west[3] InOut; io_west[2]
InOut; io_west[1] InOut; io_west[0] InOut; io_north[3] InOut;
io_north[2] InOut; io_north[1] InOut; io_north[0] InOut;
io_east1[3] InOut; io_east1[2] InOut; io_east1[1] InOut;
io_east1[0] InOut; io_east2[3] InOut; io_east2[2] InOut;
io_east2[1] InOut; io_east2[0] InOut; io_south[3] InOut;
io_south[2] InOut; io_south[1] InOut; io_south[0] InOut; }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

STIL_NESTED_MACRO

Setting this keyword to 1 when saving STRUCTURAL_STIL, STIL2005, or CTL patterns causes sub procedures to be interpreted as Macros. As Macros, they are nested inside the calling Macros.

Usage

```
STIL_NESTED_MACRO {1 | 0};
```

Setting this keyword to 1 when saving STRUCTURAL_STIL, STIL2005, or CTL patterns causes sub procedures to be interpreted as Macros. As Macros, they are nested inside the calling Macros. The tool uses loop statements, if necessary, to repeat the Macro. This keyword is set to 0 by default.

STIL_NO_OPTIMIZE_VEC

Disables vector optimizations that can reduce pattern file size and improve writing performance in STIL files.

Syntax

```
STIL_NO_OPTIMIZE_VEC {0 | 1};
```

Description

By default, STIL pattern files written by Tessent tools employ the following optimizations that can reduce pattern file size and improve writing performance:

- (STIL1999 only) SignalGroup assignments in a vector are eliminated if the assignment is the same as in the previous vector.
- The repeat vector flag ("\r") is used when appropriate to reduce the length of WaveformChars in vector assignments.

Set this keyword to 1 to disable these optimizations for backward compatibility. This causes all vectors to be fully specified without using the repeat vector flag.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

With STIL_NO_OPTIMIZE_VEC set to 1, the tool would write these example vectors:

```
V {
      _pi_ = 0000000000001011 ;
      _po_ = XXXXXXXXXXXXXXXXX ;
}
V {
      _pi_ = 0000000000000011 ;
      _po_ = XXXXXXXXXXXXXXXXX ;
}
```

By default, the tool writes these vectors instead:

```
V {
      _pi_ = \r12 0 1011 ;
      _po_ = \r17 X ;
}
V {
      _pi_ = \r14 0 11 ;
}
```

STIL_NO_SCANSTRUCTURE

Setting this keyword to 1 removes the entire ScanStructures block from the STIL output.

Usage

```
STIL_NO_SCANSTRUCTURE {1 | 0};
```

Setting this keyword to 1 removes the entire ScanStructures block from the STIL output. For design flows where the ScanStructures block is not required, removing the block significantly reduces the size of the STIL file. The ScanStructures block is included by default (0).

STIL_NO_SCANX

If set to 1, this keyword replaces all X scan load values in the STIL pattern output with 0 scan load values. The default is 0.

Usage

```
STIL_NO_SCANX {1 | 0};
```

If set to 1, this keyword replaces all X scan load values in the STIL pattern output with 0 scan load values. The default is 0.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

STIL_NO_WFC_T

Removes the waveform character “T” from a STIL pattern.

Usage

```
STIL_NO_WFC_T {1 | 0};
```

Provides a method for removing the waveform character “T” from a STIL pattern. The default is 0 (false). Set to 1 (true) to instruct the tool to remove the waveform character “T”.

STIL_NOMEASURE_CLOCK

Issuing a 1 with this keyword drops the measure timing from the timing statement for bidirectional clocks and also verifies that this pin is never measured.

Usage

```
STIL_NOMEASURE_CLOCK {1 | 0};
```

Issuing a 1 with this keyword drops the measure timing from the timing statement for bidirectional clocks and also verifies that this pin is never measured. If the pin is measured within the patterns, the output generates an error and returns to the command prompt. To deactivate, issue a 0 (the default) with this keyword.

Examples follow:

```
Parameter file : default
IOclk { LHX { ’0ns’ X; ’300ns’ H/L/X; ’400ns’ X; }
            01 { ’0ns’ D; ’200ns’ U/D; ’300ns’ D } }

Parameter file : STIL_NOMEASURE_CLOCK 1;
IOclk { 01 { ’0ns’ D; ’200ns’ U/D; ’300ns’ D } }

Parameter file : STIL_TRIM_PULSE 1;
                 STIL_NOMEASURE_CLOCK 1;
IOclk { 01 { ’200ns’ U/D; ’300ns’ D } }
```

STIL_PAT_ANN

Specifies the annotation string that the tool outputs to the STIL pattern file just prior to each scan pattern.

Usage

```
STIL_PAT_ANN string %special_keyword string2;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Specifies the annotation string that the tool outputs to the STIL pattern file just prior to each scan pattern.

```
annotation "string %keywords string"
```

The following example shows lines in a parameter file and the resultant text generated in the output pattern file.

Parameter file entry:

```
STIL_PAT_ANN   "Pattern Number:%pat_num
(%pat_num_plus1), Cycle Number:%cycle_num"
```

Resultant pattern file statement:

```
Ann {* "Pattern Number:0 (1), Cycle Number:0" *}
```

STIL_PAT_CMT

Turns off (0) the pattern comment that precedes each ATPG pattern. The default is 1.

Usage

```
STIL_PAT_CMT {1 | 0};
```

Turns off (0) the pattern comment that precedes each ATPG pattern. The default is 1.

STIL_PAT_LAB

Turns off (0) the pattern label that precedes each ATPG pattern.

Usage

```
STIL_PAT_LAB {1 | 0};
```

Turns off (0) the pattern label that precedes each ATPG pattern. The default is 1.

STIL_QUAD_MODE

Enables quad mode support for IJTAG patterns that are in the STIL format.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Usage

```
STIL_QUAD_MODE { 1 | 0};
```

Turns on (1) quad mode support for IJTAG patterns that are in the STIL format.

You use this parameter to ensure the IJTAG STIL patterns support quad-mode, where four vectors can be executed by one ATE cycle. To support quad mode, the number of vectors before the loop, inside the loop, between loops, and after the loop must be a multiple of four.

To support this requirement, the tool may partially or fully unroll loops and add padding to the end of the test. Padded vectors have all output masked to X and all clocks disabled to ensure that they do not affect the timing or functionality of the test.

To set to a multiple of 2, see the STIL_DUAL_MODE parameter keyword.

STIL_QUOTE_ALL

Causes all user identifiers to be enclosed in double quotation marks in the STIL output file.

Usage

```
STIL_QUOTE_ALL {1 | 0};
```

Causes all user identifiers to be enclosed in double quotation marks in the STIL output file. By default (0), user identifiers are not quoted.

If the STIL_QUOTE_ALL and STIL_MIN_QUOTE keywords are both active (1), the tool ignores the STIL_MIN_QUOTE keyword.

STIL_SCAN_ANN1

Specifies the string the tool outputs to the STIL pattern just prior to the first scan pattern.

Usage

```
STIL_SCAN_ANN1 string %special_keyword string2;
```

Specifies the string the tool outputs to the STIL pattern just prior to the first scan pattern. There is only one STIL_SCAN_ANN1 statement written for each pattern file generated.

```
annotation "string %keywords string"
```

The following example shows lines in a parameter file and the resultant text generated in the output pattern file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Parameter file entry:

```
STIL_SCAN_ANN1 "Scan Chain:%chain, length:%length,
SI:%scan_in, SO:%scan_out"
```

Resultant pattern file statement:

```
Ann {*
"Scan Chain:chain1, length:2, SI:scan_in1, SO:Q" *}
```

STIL_SCAN_IN_OUT_USE_PSEUDO_PORT_NAME

Specifies to use the pseudo-port names for ScanIn and ScanOut in the ScanStructures of the STIL pattern file.

Usage

```
STIL_SCAN_IN_OUT_USE_PSEUDO_PORT_NAME {1 | 0};
```

By default, the STIL pattern file does not use pseudo-port names for ScanIn and ScanOut. Set this parameter to 1 to use the pseudo-port names for these channels.

STIL_STRUCTURAL

Causes the STIL file to be written using a more structural form.

Usage

```
STIL_STRUCTURAL {1 | 0};
```

Causes the STIL file to be written using a more structural form. This keyword set to 1 results in a STIL output form that uses some of the basic hierarchical approaches of CTL output, but not any of the IEEE 1450.6 CTL syntax extensions. As with CTL output, the structural STIL output creates STIL Macro definitions for all used procedures, and a STIL Procedure definition for the test_setup procedure, if present. The start of each pattern still consists of an annotation statement and a pattern label. Each pattern consists of the calls to each specific Macro, passing all needed data. Unlike the CTL output, a new “pattern type” macro is not created for each pattern, and the pattern data is not placed in a separate file and included with the “Include” statement.

To produce a flat or non-hierarchical set of vector data, set this keyword to 0 (default).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following is an example of the structural STIL output:

```
STIL 1.0 ;

Header { ... }

// Parameter File Keyword Settings
// STIL_STRUCTURAL true ;

Signals {
   "IN"[3..0] In; "CLK" In; "SDI" In; "SE" In;
   "OUT"[3..0] Out; "SDO" Out;
}

SignalGroups {
   _pi_[6..0] = '"IN"[3..0] + "CLK" + "SDI" + "SE"';
   _po_[4..0] = '"OUT"[3..0] + "SDO"';
   PI_grp_0[5..0] = '"IN"[3..0] + "SDI" + "SE"';
   PI_grp_1 = '"CLK"';
   "_chain_SDI_" = '"SDI"' {ScanIn 10;}
   "_chain_SDO_" = '"SDO"' {ScanOut 10;}
}

Timing TRANSITION_timing {
   WaveformTable tset_tp_shift {
      Period '400ns' ;
      Waveforms {
         PI_grp_0 { 01X { '0ns' D/U/N; }}
         PI_grp_1 { 01 { '0ns' D; '100ns' D/U; '400ns' D;}}
         _po_ { LHXZ { '0ns' X; '50ns' l/h/X/t; '52ns' X;}}
      }
   }
   WaveformTable tset_tp_capture {
      Period '400ns' ;
      Waveforms {
         PI_grp_0 { 01X { '0ns' D/U/N; }}
         PI_grp_1 { 01 { '0ns' D; '100ns' D/U; '400ns' D;}}
         _po_ { LHXZ { '0ns' X; '50ns' l/h/X/t; '52ns' X;}}
      }
   }
}

ScanStructures { ScanChain chain { ... } }

Procedures {
   test_setup {
      W tset_tp_shift;
      V { "SE" = 0; }
   }
}

MacroDefs {
   load_unload_grp1 {
      W tset_tp_shift;
      V { "CLK" = 0; }
      Shift { V { "SE" = 1; "_chain_SDI_" = #;
                                    "_chain_SDO_" = #; "CLK" = 1; }
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
      }
   }

   capture {
      W tset_tp_shift;
      V { _pi_ = %; _po_ = %; PI_grp_1 = %; }
      }
   }

   PatternBurst scanpats { PatList { scan_test } }

   PatternExec scanexec {
      Timing TRANSITION_timing;
      PatternBurst scanpats;
   }

   Pattern "scan_test" {
      Call test_setup ;
      Ann {* Begin chain test *}
      Ann {* Chain Pattern:0 Cycle:1 Loop:1 *}
      ...
      Ann {* End chain test *}
      Ann {* Pattern:0 Cycle:7 Loop:10 *}
      "pattern 0": Macro "load_unload_grp1" {
         "_chain_SDI_" = 1010001110;
         "_chain_SDO_" = XXXXXXXXXX;
   }
   Macro "capture" {
      _pi_ = X101011;
      _po_ = LLHHH;
   PI_grp_1 = 1;
   }

   ...
   Ann {* Pattern:7 Cycle:24 Loop:52 *}
   "pattern 0": Macro "load_unload_grp1" {
      "_chain_SDI_" = 1010010000;
      "_chain_SDO_" = HHLHHLLLLL;
   }

   Macro "capture" {
      _pi_ = X000011;
      _po_ = LHHLL;
      PI_grp_1 = 1;
   }

   last_unload: Macro "load_unload_grp1" {
      "_chain_SDI_" = 1010010000;
      "_chain_SDO_" = LHLLLHLLLH;
      }
   }
```

STIL_TI_TITLE

Turns on a TI-specific title block in the STIL Header block.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Usage

```
STIL_TI_TITLE {1 | 0};
```

Turns on a TI-specific title block in the STIL Header block. This title block contains all of the specific statements from the TITDL output file, and you can use the TITDL-specific keywords to change the fields in the title block. The default is to turn off the title block.

The TITDL-specific keywords include: TITDL_LIBRARY, TITDL_CUSTOMER, TITDL_PARTNUM, TITDL_SETNAME, TITDL_SETTYPE, TITDL_CHAIN_SETTYPE, and TITDL_REVISION.

STIL_TIMING_BLOCK_NAME

Overrides a timing block name in the STIL file.

Usage

```
STIL_TIMING_BLOCK_NAME string;
```

Provides a method to override a timing block name in the STIL file.

The following example overrides the STIL timing block name with “MyTime”.

write_patterns pat2.stil -stil -rep -parameter_list {STIL_TIMING_BLOCK_NAME MyTime}

STIL_TRIM_PULSE

Causes the input timing to drop the leading 0ns force to the off state of the clock.

Usage

```
STIL_TRIM_PULSE {1 | 0};
```

Causes the input timing to drop the leading 0ns force to the off state of the clock (keyword set to 1). Removing the initial force to the off state of a clock may not be appropriate for all testers and translation processes, and you need to understand all downstream processes that use the resultant files. To deactivate, set this keyword to 0 (default).

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples follow:

```
Parameter file : default
IOclk { LHX { '0ns' X; '300ns' H/L/X; '400ns' X; }
            01 { '0ns' D; '200ns' U/D; '300ns' D } }

Parameter file : STIL_TRIM_PULSE 1;
IOclk { LHX { '0ns' X; '300ns' H/L/X; '400ns' X; }
            01 { '200ns' U/D; '300ns' D } }

Parameter file : STIL_TRIM_PULSE 1;
                 STIL_NOMEASURE_CLOCK 1;
IOclk { 01 { '200ns' U/D; '300ns' D } }
```

STIL_VECTOR_ANN

Specifies the vector types to be added to the output pattern file.

Usage

```
STIL_VECTOR_ANN string %special_keyword string2;
```

Specifies the vector types to be added to the output pattern file.

```
annotation "string %keywords string"
```

Note

You must not use the "%" symbol as the leading character to any string fragment in conjunction with the %special_keyword.

The STIL_SCAN_ANN1, STIL_PAT_ANN, and STIL_VECTOR_ANN parameter file keywords use a (%special_keyword) that is translated at the specific time and place the annotation statement is written. Table A-3 lists the available special keywords. Table A-4 lists the pattern types. Table A-5 lists the vector types.

Table A-3. STIL Special Keywords

Keyword	Description
%chain	The chain name of the current chain.
%cycle_num	The current ATPG cycle number.
%length	The length of the current chain.
%pat_num	The current ATPG pattern number.
%pat_num_plus1	The current ATPG pattern number plus 1.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table A-3. STIL Special Keywords (cont.)

Keyword	Description
%pattern_type	The pattern type, for outputting annotations in the pattern file. It causes one of the pattern types in Table A-4 to be written into the pattern file.
%scan_in	The name of the Primary Input pin for the current chain.
%scan_out	The name of the Primary Output pin for the current chain.
%vector_type	This keyword causes one of the vector types in Table A-5 to be written into the pattern file.

Table A-4. STIL Pattern Types

basic	macrotest	ram_passthru
clock_sequential	multiple_load	

Table A-5. STIL Vector Types

CLOCK_PROCEDURE	PRIMARY_CLOCKPO	SHADOW_CONTROL
CLOCK_PROCEDURE_RESTORE	PRIMARY_CLOCKPO_IDDQ	SHADOW_OBSERVE
CORE_ACCESS	PRIMARY_IDDQ	SHIFT
CORE_INST_ISOLATE	PRIMARY_MEASURE	SINGLE_SHIFT
CORE_INST_TEST	PRIMARY_MEASURE_IDDQ	SKEW_LOAD
LOAD_UNLOAD	SCAN_IN	SEQ_TRANSPARENT
MASTER_OBSERVE	SCAN_IO	TEST_SETUP
PRIMARY	SCAN_OUT	

STIL_VERG_ESC

Setting this keyword to 1 changes all vectored signal names to scalars.

Usage

```
STIL_VERG_ESC {1 | 0};
```

Setting this keyword to 1 changes all vectored signal names to scalars. For example, the pin name “bus”[0] would change to “bus[0]”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

STIL_WFC_KEEP_N

Forces the N WaveformChar to be present for input signal groups in STIL pattern files.

Syntax

```
STIL_WFC_KEEP_N {1 | 0};
```

Description

By default, Tessent tools write STIL pattern files that attempt to minimize the WaveformChars used in a WaveformTable for a given signal. If the pattern writer determines that a particular signal or SignalGroup never needs to force an unknown value, it does not include the N WaveformChar. If you prefer to have the N value present in all WaveformTable entries, set this parameter to 1 to force the N value to appear in the STIL files.

Examples

With STIL_WFC_KEEP_N set to 0 (default):

```
input_time_gen_0 { 01 { '0ns' D/U; }}
```

With STIL_WFC_KEEP_N set to 1:

```
input_time_gen_0 { 01N { '0ns' D/U/N; }}
```

Related Topics

STIL_WFC_KEEP_Z

STIL_WFC_KEEP_Z

Forces the Z WaveformChar to be present for input signal groups in STIL pattern files.

Syntax

```
STIL_WFC_KEEP_Z {1 | 0};
```

Description

By default, Tessent tools write STIL pattern files that attempt to minimize the WaveformChars used in a WaveformTable for a given signal. If the pattern writer determines that a particular signal or SignalGroup never needs to drive a Z value, it does not include the Z WaveformChar. If you prefer to have the Z value present in all WaveformTable entries, set this parameter to 1 to force the Z value to appear in the STIL files.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Examples

With STIL_WFC_KEEP_Z set to 0 (default):

```
input_time_gen_0 { 01 { '0ns' D/U; }}
```

With STIL_WFC_KEEP_Z set to 1:

```
input_time_gen_0 { 01Z { '0ns' D/U/Z; }}
```

Related Topics

STIL_WFC_KEEP_N

STIL_WRAP_ANNOTATION

Breaks multi-line annotations into multiple annotation statements.

Usage

```
STIL_WRAP_ANNOTATION {1 | 0};
```

When this keyword is set to 1 or true in the parameter file, then any multi-line annotations are broken into multiple annotation statements, such as:

```
Ann {* this is a long *}
Ann {* multi-line annotation *}
```

If not present in the parameter file, the default for this keyword is 0 or off.

STIL2005_ONE_FILE

Controls whether the pattern writer writes only a single pattern file, keeping the pattern data in the main file instead of a separate pattern data file, or two pattern files.

Usage

```
STIL2005_ONE_FILE {1 | 0};
```

Description

By default, the STIL2005 pattern format writes pattern data into a single STIL file. Setting this keyword to 0 enables you to change this behavior, writing two files: one containing setup information and one containing the pattern data and an Include statement to reference the first file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This keyword does not affect the CTL pattern writer. If you use this keyword with the STIL1999 format, the tool reports a warning stating that it ignores this parameter for the STIL1999 format.

TITDL_CHAIN_SETTYPE

Specifies the string used to set the chain type.

Usage

```
TITDL_CHAIN_SETTYPE string;
```

Specifies the string used to set the chain type. If specified, the string is used in the output of Chain_Test-only patterns. If not specified, the default string is “SCANCHK”. To add this to the specific title block, use the STIL_TI_TITLE keyword.

TITDL_CUSTOMER

Specifies the string the tool uses in the CUSTOMER statement in the header of the TITDL output file.

Usage

```
TITDL_CUSTOMER string;
```

Specifies the string the tool uses in the

```
CUSTOMER=string
```

statement in the header of the TITDL output file. If not specified, the default is TEXAS INSTRUMENTS. To add this to the specific title block, use the STIL_TI_TITLE keyword.

TITDL_GROUP_TIMING

Setting this keyword to 1 in conjunction with the “write_patterns -Titdl” command groups like timing information into a single statement.

Usage

```
TITDL_GROUP_TIMING {1 | 0};
```

Setting this keyword to 1 in conjunction with the “write_patterns -Titdl” command groups like timing information into a single statement. The default is off.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

TITDL_KEEP_SCANOUT

Setting this keyword to 1 causes all SCANOUT statements to be issued, even if all of the scanout values for a specific pattern/chain are X.

Usage

```
TITDL_KEEP_SCANOUT {1 | 0};
```

Setting this keyword to 1 causes all SCANOUT statements to be issued, even if all of the scanout values for a specific pattern/chain are X. The default is 0, which causes the tool to not keep scanout statements whose scanout values are all X when writing TITDL format vectors.

TITDL_LIBRARY

Specifies the string that the tool uses in the LIBRARY_TYPE statement in the header of the TITDL output file.

Usage

```
TITDL_LIBRARY string;
```

Specifies the string that the tool uses in the

```
LIBRARY_TYPE=string
```

statement in the header of the TITDL output file. If not specified, the default is GS20. To add this to the specific title block, use the STIL_TI_TITLE keyword.

TITDL_PARTNUM

Specifies the string the tool uses in the TI_PART_NUMBER statement in the header of the TITDL output file.

Usage

```
TITDL_PARTNUM string;
```

Specifies the string the tool uses in the

```
TI_PART_NUMBER=string
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

statement in the header of the TITDL output file. If not specified, the default is F999999. To add this to the specific title block, see the parameter file keyword STIL_TI_TITLE.

TITDL_PSEUDO_PREFIX

Specifies the string that is prepended to the name of the clock pin to create the name of the pseudo clock pin.

Usage

```
TITDL_PSEUDO_PREFIX string;
```

Specifies the string that is prepended to the name of the clock pin to create the name of the pseudo clock pin. If not specified, the default is “pseudo_”.

For example, this keyword changes the default pseudo prefix:

```
TITDL_PSEUDO_PREFIX ps_ ;
```

And this is the result of what the MUX_PIN statement would look like in the TITDL output. The clock pin is called “clock”, and the added pseudo clock pin is called “ps_clock” because of the use of the above parameter file keyword.

```
...
CONNECT P,
VAR =(in1, in2, clock, ps_clock, out1, out2),
DEFPIN=(IN 4, OUT 2);
PERIOD = 100 NS;
CLOCK VAR=(clock),
HOLD0 = 20 NS, HOLD1 = 10 NS, PATTERN = 010;
CLOCK VAR = (ps_clock),
HOLD0 = 60 NS, HOLD1 = 10 NS, PATTERN = 010;
MUX_PIN clock, VAR=(clock, ps_clock);
DELAY VAR = (in1),
OFFSET = 0 NS;
...
```

TITDL_REVISION

Specifies the string that the tool uses in the REVISION statement in the header of the TITDL output file. I

Usage

```
TITDL_REVISION string;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Specifies the string that the tool uses in the

```
REVISION=string
```

statement in the header of the TITDL output file. If not specified, the default is 1.00. To add this to the specific title block, use the STIL_TI_TITLE keyword.

TITDL_SET_DESCRIPTION

Adds a user-specified string for use as the PATTERN_SET_DESCRIPTION.

Usage

```
TITDL_SET_DESCRIPTION string;
```

When set to a string, this is used in both TITDL files and in STIL files that also use STIL_TI_TITLE to add the PATTERN_SET_DESCRIPTION statement where the string is used as the description.

TITDL_SETNAME

Specifies the string that the tool uses in the PATTERN_SET_NAME statement in the header of the TITDL output file.

Usage

```
TITDL_SETNAME string;
```

Specifies the string that the tool uses in the

```
PATTERN_SET_NAME=string
```

statement in the header of the TITDL output file. The string can be a maximum of twenty characters. If not specified, the default is the filename with no extension. To add this to the specific title block, use the STIL_TI_TITLE keyword.

TITDL_SETTYPE

Specifies the string that the tool uses in the PATTERN_SET_TYPE statement in the header of the TITDL output file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Usage

```
TITDL_SETTYPE string;
```

Specifies the string that the tool uses in the

```
PATTERN_SET_TYPE=string
```

statement in the header of the TITDL output file. If not specified, the default is SCAN. To add this to the specific title block, use the STIL_TI_TITLE keyword.

WGL_ADD_LASTX

Specifies for the tool to add an additional vector to the end of the pattern set (when set to 1).

Usage

```
WGL_ADD_LASTX {1 | 0};
```

Specifies for the tool to add an additional vector to the end of the pattern set (when set to 1). This vector is a non-scan vector with all clocks disabled, all output pins set to “X” (not measuring), and all input pins set to the same value of the previous non-scan vector. To deactivate, set this keyword to 0 (the default).

WGL_ALT_BIDI

Changes the syntax of how bidirectional signals are represented in the vector pin list, and the actual states in the vector.

Usage

```
WGL_ALT_BIDI {1 | 0};
```

Changes the syntax of how bidirectional signals are represented in the vector pin list, and the actual states in the vector. The default representation (keyword set to 0) specifies the bidirectional signal name, a colon, then an I or O, for the direction of that pin. There must be two entries for the bidirectional pin, one specifying the input side and one specifying the output side. The vector lists a separate state for each.

Setting this keyword to 1 changes the representation so that the pin list lists only one entry for the bidirectional pin with no direction, but with two states. Every state in the actual vector is currently separated by spaces. With the addition of this WGL_ALT_BIDI 1 setting, the states for the bidirectional pin are listed side-by-side with no space separation. The input state is listed first.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

WGL_ALT_BIDI set to 0 (default) — The format of the vector order pin list and vector is as follows:

```
pattern test("A","B","C","BiDi1:I","E","F","G","BiDi1:O")
{Pattern 0 Cycle 0 Loop 0}
vector(+,TP1):=[1 0 1 - 0 1 0 1];
```

WGL_ALT_BIDI set to 1 — The vector order pinlist and vector are changed to the following:

```
pattern test("A","B","C","BiDi1","E","F","G")
{Pattern 0 Cycle 0 Loop 0}
vector(+,TP1):=[1 0 1 -1 0 1 0];
```

WGL_ALT_VECT_ANN

Specifies alternate annotation statements for beginnings of patterns and vector boundaries when this keyword is set to 1.

Usage

```
WGL_ALT_VECT_ANN {1 | 0};
```

Specifies alternate annotation statements for beginnings of patterns and vector boundaries when this keyword is set to 1. These alternate annotations can help the WGL file be processed by other third party tools. The default is 0.

Note

Using this keyword can produce a WGL file that you cannot read back into the Tessent tools.

WGL_ANNOTATION_INTERVAL

Controls the annotation frequency in WGL pattern files.

Usage

```
WGL_ANNOTATION_INTERVAL integer;
```

Description

When a callback writes patterns, the write_pattern command adds annotations to the WGL files similar to the following:

```
ANN {* Pattern:0 Vector:6 TesterCycle:952 *}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Setting WGL_ANNOTATION_INTERVAL to a value less than one suppresses these annotations. Setting this parameter to a positive integer <*n*> causes the callback to write annotations after every <*n*>th vector.

For IJTAG patterns, the default value is one (annotations for every vector). For all other patterns, the default value is zero, meaning that the WGL files are not annotated by default.

WGL_EDGE_STROBE

Specifies whether to include an X state after an edge strobe.

Usage

```
WGL_EDGE_STROBE {1 | 0};
```

The default value of this keyword is 0. This keyword is in effect when the tool uses edge strobing, which is the default behavior. Refer to the ALL_OVERRIDE_STROBE_METHOD keyword for more information about how the tool chooses whether to use edge or window strobing.

The WGL output specifies edge strobes by using the edge qualifier on the strobe times specified in timeplate definitions. Additionally, an output value showing the X state appears after the strobe time.

If the strobe_window is set to 0, setting this keyword to 1 eliminates from the WGL output this additional output value showing the X state after the strobe time.

The following is an example of an edge strobe statement without and with using WGL_EDGE_STROBE.

WGL_EDGE_STROBE set to 0 (default):

```
"outpin":=output[0ns:X,300ns:Q'edge,350ns:X];
```

WGL_EDGE_STROBE set to 1

```
"outpin":=output[0ns:X,300ns:Q'edge];
```

WGL_EXCLUDE_UNUSED_BUS_BITS

Instructs the WGL writer to not include unused bus bits in the STIL pattern file.

Usage

```
WGL_EXCLUDE_UNUSED_BUS_BITS {1 | 0};
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

You use this parameter to control the presence of unused bus bits in the WGL pattern file. The default is 0.

Note

WGL patterns generated with WGL_EXCLUDE_UNUSED_BUS_BITS=1 cannot be converted to a Verilog testbench with stand-alone Siemens EDA or third-party tools, because the information about the unused bus bits is not available in the pattern file. They can, however, be read in to Tessent Shell with read_patterns.

Related Topics

STIL_EXCLUDE_UNUSED_BUS_BITS

WGL_FULL_CHAIN

When set to 1, this keyword alters the scancell and scanchain blocks of the WGL to reflect all cells in the chain, regardless of the number of pre- or post-shifts (number of apply shifts).

Usage

```
WGL_FULL_CHAIN {1 | 0};
```

When set to 1, this keyword alters the scancell and scanchain blocks of the WGL to reflect all cells in the chain, regardless of the number of pre- or post-shifts (number of apply shifts). The group statements only reflect what is loaded minus the post shift. The scanstate and group have to match. To deactivate, and for backwards compatibility, issue a 0 (false).

Examples follow.

Original WGL File (WGL_FULL_CHAIN 0)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
...
    scancell
        { Group: grp1 }
        { Chain: REG1 }
        "u22"; "u23"; "u10"; "u11"; "u12";
        groupgrp1REG1 ["u22", "u23", "u10", "u11", "u12"];
    end
    scanchain
        { Group: grp1 }
        REG1_chain[ "sdi", "u12", "u11", !, "u10", "u23", "u22", "sdo"];
    end
 ...
    scanstate
        OREG1_sts000000_000002 := groupgrp1REG1 (XXXXX);
        IREG1_sts000000_000002 := groupgrp1REG1 (11010);
 ...
    end
 ...
```

WGL_FULL_CHAIN 1

```
...
    scancell
        { Group: grp1 }
        { Chain: REG1 }
        "u22"; "u23"; "u10"; "u11"; "u12"; "u13";
        groupgrp1REG1 ["u22", "u23", "u10", "u11", "u12"];
    end
    scanchain
        { Group: grp1 }
        REG1_chain[ "sdi", "u13", "u12", "u11", !, "u10", "u23", "u22",
"sdo"];
    end
 ...
    scanstate
        OREG1_sts000000_000002 := groupgrp1REG1 (XXXXX);
        IREG1_sts000000_000002 := groupgrp1REG1 (11010);
 ...
    end
 ...
```

WGL_FULL_SCANGROUP

Creates the Scan sections with the full chain (when set to 1).

Usage

```
WGL_FULL_SCANGROUP {1 | 0};
```

Creates the Scan sections with the full chain (when set to 1). This keyword alters the scanchain, scancell, and scanstate blocks in WGL to reflect all cells in the chain, regardless of the number of pre- or post-shifts. Setting the keyword to 0, deactivates this feature.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following is an example of WGL code when WGL_FULL_SCANGROUP and WGL_FULL_CHAIN are both set to 1:

```
...
    scancell
        { Group: grp1 }
        { Chain: REG1 }
        "u22"; "u23"; "u10"; "u11"; "u12"; "u13";
        groupgrp1REG1 ["u22", "u23", "u10", "u11", "u12", "u13"];
    end
    scanchain
        { Group: grp1 }
        REG1_chain[ "sdi", "u13", "u12", "u11", !, "u10", "u23", "u22",
                                                                "sdo"];
    end
 ...
    scanstate
        OREG1_sts000000_000002 := groupgrp1REG1 (XXXXXX);
        IREG1_sts000000_000002 := groupgrp1REG1 (X10100);
 ...
    end
 ...
```

WGL_GROUP_PIN

Specifies for the WGL output to create signal groups _PI_, _PO_, and _BIDI_ (if there are bidi's) when set to 1.

Usage

```
WGL_GROUP_PIN {1 | 0};
```

Specifies for the WGL output to create signal groups _PI_, _PO_, and _BIDI_ (if there are bidi's) when set to 1. The WGL code then uses these groups in the pattern statement and in actual patterns for grouping states. To deactivate, set this keyword to 0.

WGL_INV_SC

Removes inversion data from scan chain definitions and specifies scan data as it would appear to be shifted in, instead of parallel deposited.

Usage

```
WGL_INV_SC {1 | 0};
```

Removes inversion data from scan chain definitions and specifies scan data as it would appear to be shifted in, instead of parallel deposited. According to the WGL data model, a WGL file

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

that uses this option no longer correctly matches the design netlist. However, some third-party tools work faster with the inversion data removed. When set to 0 (default), this keyword leaves the inversion data intact.

WGL_LOOP_ONLY

Specifies that “loop” statements should always be used instead of “repeat” statements in WGL output.

Usage

```
WGL_LOOP_ONLY {1 | 0}
```

In WGL pattern file syntax, if one vector must be repeated multiple times, it is legal to use either a “repeat” statement or a “loop…end” block to repeat the single vector. By default, the WGL writer uses “repeat” statements for single vectors and loops for multiple vectors. Setting this keyword to 1 forces WGL output to always use “loop” blocks instead of “repeat” statements.

WGL_NO_SCANX

Replaces all X scan load values in the WGL pattern output with 0 scan load values.

Usage

```
WGL_NO_SCANX {1 | 0};
```

Replaces all X scan load values in the WGL pattern output with 0 scan load values. When set to 0 (default), this keyword deactivates this feature.

WGL_NOMEASURE_CLOCK

Drops the measure timing from the timing statement for bidirectional clocks and also verifies that this pin is never measured.

Usage

```
WGL_NOMEASURE_CLOCK {1 | 0};
```

Drops the measure timing from the timing statement for bidirectional clocks and also verifies that this pin is never measured. If the pin is measured within the patterns, the output generates an error and returns to the command prompt. Setting this keyword to 0 (default), deactivates this feature.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note that this keyword can cause some WGL readers to have problems reading the file. The tool issues a warning if the keyword is specified in the parameter file and if there is at least one signal using the altered syntax.

Examples follow:

```
Parameter file : default
IOclk := input [ 0ns:D, 200ns:S, 300ns:D ];
IOclk := output [ 0ns:X, 300ns:Q, 400ns:X ];

Parameter file : WGL_NOMEASURE_CLOCK 1;
IOclk := input [ 0ns:D, 200ns:S, 300ns:D ];

Parameter file : WGL_TRIM_PULSE 1;
                 WGL_NOMEASURE_CLOCK 1;
IOclk := input [ 200ns:S, 300ns:D ];
```

WGL_ONE_ILLINOIS

Causes the WGL output to have only one scan state defined per pattern for multiple scan chains that share a single scan input pin.

Usage

```
WGL_ONE_ILLINOIS {1 | 0};
```

Causes the WGL output to have only one scan state defined per pattern for multiple scan chains that share a single scan input pin. In other words, this keyword forces the tool to produce a single scan state for chains that share a common scan input pin (set to 1).

The default behavior when this keyword is set to 0 is to have scan states defined for each scan chain, whether or not they share scan input pins. The default for scan states when multiple scan chains share a common scan input is to list the scan states for each cell in each chain, the same as if the chains did not share a scan input.

WGL_PATTERN_NAME

Changes the default name of the WGL pattern block from “Chain_scan_test” to the value of the string.

Usage

```
WGL_PATTERN_NAME string;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Changes the default name of the WGL pattern block from “Chain_scan_test” to the value of the string.

WGL_TRIM_LEADING_P

Removes the leading 0ns:P timing event when a channel uses timing variables .

Usage

```
WGL_TRIM_LEADING_P {1 | 0};
```

Removes the leading 0ns:P timing event when a channel uses timing variables (keyword set to 1). For example, the default behavior produces a timeplate channel that looks like this:

```
"pi_sel_smcard" := input[0ns:P, fd_force:S];
```

When WGL_TRIM_LEADING_P is set to 1 in the parameter file, the timeplate channel looks like this:

```
"pi_sel_smcard" := input[fd_force:S];
```

WGL_TRIM_PULSE

Causes the input timing to drop the leading 0 ns force to the off state of the clock.

Usage

```
WGL_TRIM_PULSE {1 | 0};
```

Causes the input timing to drop the leading 0 ns force to the off state of the clock (keyword set to 1). Removing the initial force to the off state of a clock may not be appropriate for all testers and translation processes, and you need to understand all downstream processes that use the resultant files. Setting this keyword to 0 deactivates this feature.

Note that this keyword generates illegal syntax in the WGL file, which may cause some WGL readers to have problems reading the file. The tool issues a warning message if the keyword is specified in the parameter file and if there is at least one signal using the altered syntax.

Examples follow.

Parameter file : default

```
IOclk := input [ 0ns:D, 200ns:S, 300ns:D ];
IOclk := output [ 0ns:X, 300ns:Q, 400ns:X ];
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Parameter file : WGL_TRIM_PULSE 1;

```
IOclk := input [ 200ns:S, 300ns:D ];
IOclk := output [ 0ns:X, 300ns:Q, 400ns:X ];
```

Parameter file : WGL_TRIM_PULSE 1;

WGL_NOMEASURE_CLOCK 1;

```
IOclk := input [ 200ns:S, 300ns:D ];
```

WGL_VECTOR_ANN

Adds an annotation statement to the generated pattern file that denotes the vector type, vector number, and cycle number.

Usage

```
WGL_VECTOR_ANN {1 | 0};
```

The default value is 0. Setting this parameter to 1 activates this feature. The annotation added to the pattern file appears in the following format:

```
{ Vector type: <Type>, Vector: <VectorNum>, TesterCycle: <CycleNum> }
```

Examples

The following shows a sample vector annotation:

```
{ Vector type: PRIMARY, Vector: 8, TesterCycle: 302 }
```

WGL_VERG_ESC

Specifies for buses to be expanded in the declaration sections of the WGL output.

Usage

```
WGL_VERG_ESC {1 | 0};
```

Specifies for buses to be expanded in the declaration sections of the WGL output. Setting this keyword to 0 deactivates this feature.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

WGL_VTRAN_PADSC

Pads all scan data to be same length.

Usage

```
WGL_VTRAN_PADSC {1 | 0};
```

Pads all scan data to be same length (keyword set to 1). Padding all scan data to be the same length violates the WGL syntax according to the TDS WGL documentation. However, some third-party tools work better with the scan data padded. Setting this keyword to 0 deactivates this feature.

WGL_WRAP_ANNOTATION

Breaks multi-line annotations into multiple annotation statements.

Usage

```
WGL_WRAP_ANNOTATION {1 | 0};
```

By default, the tool automatically inserts newlines into annotations when a single line of the annotation exceeds 512 characters, regardless of the setting of WGL_WRAP_ANNOTATION. When the tool splits lines in this way, it can either split them into multiple annotation statements or into a single annotation statement that is split across lines.

When this keyword is set to 1 or true in the parameter file, then any multi-line annotations are broken into multiple annotation statements, such as:

```
Ann {* this is a long *}
Ann {* multi-line annotation *}
```

If not present in the parameter file, the default for this keyword is 0 or off. With the default behavior, a single annotation may cover multiple lines:

```
Ann {* this is a long
  multi-line annotation *}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Appendix B
HDL Limitations in the Tessent Shell Flow

This appendix describes the HDL limitations which affect functionality in the Tessent Shell Flow. The two tables below summarize the limitations. Full explanations of those limitations along with examples are given in the sections which follow.

Note

There are limits to the length of character names supported within Tessent Shell.

- For RTL, the character name limit is 4095.
- For gate-level, the character name limit is 8191.

The limitations can be categorized by the effect they have on the flow—see the "Effect on Flow" column in Table B-1 on page 5083 and Table B-2 on page 5088.

- **Impacts Compilation and/or Elaboration —** An error is typically issued by the read_verilog, the read_vhdl, or the set_current_design command.
- **Impacts Design Analysis —** When a limitation in this category is encountered, an error is typically issued when the tool is tracing through the elaborated circuit. Such errors typically occur during rules checking and/or in TS_Bscan during pad extraction. In most cases tracing in TS_Ijtag is not affected. Exceptions are noted in Table B-1 on page 5083 and Table B-2 on page 5088.
- **Impacts Design editing and/or Introspection —** An error is typically issued when the tool is modifying the design to insert new design components or during execution of an introspection command. Such errors typically occur in TS_DesignEditor or in TS_Bscan or TS_Membist during design editing.
- **Impacts Design Verification —** An error is typically issued during testbench generation.
- **Impacts Hierarchical Insertion —** Limitations in this category affect the ability to make needed modifications to designs with child blocks or cores, each with their own workspace. An error is typically issued when the tool flow performs design rule checking on the interface between child and parent. Such errors typically occur during design rule checking in the early stages of the flow. Some limitations result in errors in the child flow and others in the parent flow.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

Language constructs which are not in the RTL-synthesizeable subset are ignored. DC Shell requires that such constructs be surrounded by pragmas; otherwise, an error is generated. If tool edits create a need to update such a construct, for example a Verilog hierarchical name, then a manual edit by the user is required.

HDL Limitations in the Tessent Shell Flow for Verilog and SystemVerilog 5083
HDL Limitations in the Tessent Shell Flow for VHDL 5088
SystemVerilog Interfaces 5091
Multi-dimensional Instantiation of SystemVerilog Interfaces 5091
SystemVerilog Interfaces Cannot Be Edited 5091
Outside Connections on Interface Type Ports Cannot be Intercepted 5091
Insertion of an Instance of a Module With a SystemVerilog Interface Port Type 5092
SystemVerilog Generic Interface Port Declarations in the Root Module of a Child Block 5092
SystemVerilog Interfaces in Task or Function Invocation 5093
SystemVerilog Interfaces With Logic 5094
Modport Expressions 5094
Nested Modules 5094
Configurations for Verilog 5095
SystemVerilog Enumerations 5096
SystemVerilog Enum Methods 5096
SystemVerilog Enum Type in Port Declarations 5096
Complex Port Types for Verilog and SystemVerilog 5097
Editing of Ports and Nets of Complex Types 5097
Unpacked Dimensions and Other Complex Types in Port Declarations of the Root Module of a Design 5098
Complex Types in Top-Level Ports at the Chip Level 5098
Complex Object Permissions 5098
Functions, Tasks, and Procedural Statements for Verilog and SystemVerilog 5101
SystemVerilog Extern Modules 5101
Unrolled Generate Loops for Verilog and SystemVerilog 5102
Unrolling of Generate Loops Side Effects 5102
Insertion of Custom Logic Connecting to a Port or Net in an Unrolled Generate Loop . . 5105
Parameterized Designs for Verilog and SystemVerilog 5107
Parameter Overrides in the Instantiation of the Root Module of a Child Block 5107
Support for the Defparam Construct 5107
Implicit Port Syntax 5108
SystemVerilog Implicit Port Syntax in the Instantiation of the Root Module of a Child Block 5108
SystemVerilog 2009 5109
New Module Insertion in VHDL 5110

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Insertion of VHDL Entities with Unconstrained Ports 5110
Insertion of VHDL Entities with Uninitialized Generics 5110
Complex Port Types for VHDL 5111
Editing of Ports and Nets of Complex Types 5111
Array of Array and Record Types in Port Declarations in the Root Module of a Design . 5111
Complex Types in Top-Level Ports at the Chip Level 5111
Ports with Unconstrained Array Types in the Root Design Entity of a Child Block 5112
Procedures, Functions, and Process Statements for VHDL 5112
Configurations for VHDL 5112
VHDL 2008 5112
Parameterized Designs for VHDL 5114
Generic Overrides in the Instantiation of the Root Design Entity of a Child Block 5114

HDL Limitations in the Tessent Shell Flow for Verilog and SystemVerilog

The following HDL limitations apply to Verilog and SystemVerilog.

The following table lists and categorizes these limitations.

Table B-1. Verilog and SystemVerilog Limitations

Limitation	Effect on Flow	Workaround and Link to Description
SystemVerilog Interfaces		
Multi-dimensional instantiation of SV interfaces is not supported.	Affects tracing and design editing.	Use a single dimension. See "Multi-dimensional Instantiation of SystemVerilog Interfaces" on page 5091
SV interfaces cannot be edited.	Affects design editing.	Replace the interface with the equivalent expanded logic. See "SystemVerilog Interfaces Cannot Be Edited" on page 5091.
Outside connections on ports declared using an SV interface type cannot be intercepted.	Affects design editing.	Replace interface usage with equivalent expanded logic.See "Outside Connections on Interface Type Ports Cannot be Intercepted" on page 5091.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table B-1. Verilog and SystemVerilog Limitations (cont.)

Limitation	Effect on Flow	Workaround and Link to Description
Insertion of an instance of a module containing a port whose type is an SV interface is not supported.	Affects design editing.	Replace interface usage with equivalent expanded logic. See "Insertion of an Instance of a Module With a SystemVerilog Interface Port Type" on page 5092.
A port declared in the root module of a child block or core cannot be declared using an SV generic interface type.	Affects design elaboration, hierarchical insertion, and testbench generation.	Use a known interface type. See "SystemVerilog Generic Interface Port Declarations in the Root Module of a Child Block" on page 5092.
Multiple instantiations of an SV interface cannot be passed as arguments in a single task or function invocation.	Affects RTL quick synthesis needed for checking design rules and ICL extraction.	Uniquify the interface instantiations. See "SystemVerilog Interfaces in Task or Function Invocation" on page 5093.
SV interfaces with logic are not supported.	Affects introspection, tracing, and design editing.	Specify any needed logic without utilizing an SV interface. See "SystemVerilog Interfaces With Logic" on page 5094.
Modport expressions are not supported.	Affects introspection and design editing.	Replace the named port expression with a simple name. See "Modport Expressions" on page 5094
Nested Modules		
Nested modules are not supported in code which needs to be edited or introspected.	Affects design introspection and editing.	Change the RTL. See "Nested Modules" on page 5094.
Configurations		
The Tessent Shell environment does not support editing your Verilog configuration design units.	Restart required if RTL modifications to your configuration entities are required.	Change the RTL to avoid use of configurations or insert DFT at the gate-level, post-synthesis. See "Configurations for Verilog" on page 5095.
SystemVerilog Enumerations		

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table B-1. Verilog and SystemVerilog Limitations (cont.)

Limitation	Effect on Flow	Workaround and Link to Description
Logic containing SV enum methods such as next, prev, first, and last cannot be traced.	Affects tracing.	Specify the enumerated list values explicitly. See "SystemVerilog Enum Methods" on page 5096.
Ports declared using an SV enum type declared in the port declaration itself cannot be intercepted on the inside of the module.	Affects design editing.	Replace the enum type with an integer. See "SystemVerilog Enum Type in Port Declarations" on page 5096.
Complex Port Types		
The tool does not support %d notation when specifying existing complex ports or pins. It returns one error for %d notation and another error for incomplete indexing.	Affects the DftSpecification.	There is no workaround. You must specify complex signals using complete indexing. Do not use %d notation for complex signals.
The Tessent Shell environment does not support the System Verilog datatypes named "tagged and unpacked union" for introspection.	Affects design introspection.	**Note:** The tool does support the "packed union" datatype for introspection.
Ports/nets declared using a complex type cannot be addressed in Tcl editing commands.	Affects design editing.	Replace the complex type usage with equivalent expanded logic. See "Editing of Ports and Nets of Complex Types" on page 5097.
Unpacked dimensions and other complex types cannot be used in the definitions of root module ports which drive or receive data from inserted DFT logic.	Affects testbench generation.	Use scalars and simple vectors. See "Unpacked Dimensions and Other Complex Types in Port Declarations of the Root Module of a Design" on page 5098.
When boundary scan needs to be inserted into a design, top-level port declarations at the chip level are limited to scalars and simple vectors.	Affects tracing and design editing.	Replace the complex type usage with equivalent expanded logic. See "Complex Types in Top-Level Ports at the Chip Level" on page 5098.
Functions, Tasks, and Procedural Statements		

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table B-1. Verilog and SystemVerilog Limitations (cont.)

Limitation	Effect on Flow	Workaround and Link to Description
Tracing through logic containing non-constant function calls, task invocations, and/or procedural statements is not supported.	Affects tracing.	Change the RTL. See "Functions, Tasks, and Procedural Statements for Verilog and SystemVerilog" on page 5101.
SystemVerilog Extern Modules		
Extern modules in SV are ignored.	Affects design editing.	Change the RTL. See "SystemVerilog Extern Modules" on page 5101.
Unrolled Generate Loops		
Design modifications which require unrolling of generate loops may have undesired side effects.	Affects design editing.	See "Unrolling of Generate Loops Side Effects" on page 5102.
Insertion of custom logic which connects to a port or net inside an unrolled generate loop when the insertion of the custom logic and the unrolling are done in different insertion phases (unrolling phase first).	Affects design editing.	See "Insertion of Custom Logic Connecting to a Port or Net in an Unrolled Generate Loop" on page 5105.
Parameterized Designs		
For a parameter declared in the root module of a child block or core which is used in logic that is edited during insertion, such a parameter cannot be overridden in the instantiation of the child block.	Affects hierarchical insertion.	Use a fixed value in place of the parameter or set the default to the same value as the override. See "Parameter Overrides in the Instantiation of the Root Module of a Child Block" on page 5107.
Defparams whose paths are not relative to the current scope are not supported.	Affects design compilation / elaboration.	Move the defparam so that the specified path is relative to the current scope or use the parameter override construct. See "Support for the Defparam Construct" on page 5107.
Implicit Port Syntax		

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table B-1. Verilog and SystemVerilog Limitations (cont.)

Limitation	Effect on Flow	Workaround and Link to Description
Implicit port syntax (i.e., .*) cannot be used in the instantiation of a child block or core.	Affects hierarchical insertion.	Use expanded syntax. See "SystemVerilog Implicit Port Syntax in the Instantiation of the Root Module of a Child Block" on page 5108.
SystemVerilog 2009		
Certain limitations exist for a small subset of constructs specific to SystemVerilog 2009.	Affects design editing.	Avoid the unsupported constructs if they need to be updated to reflect edits during DFT insertion. See "SystemVerilog 2009" on page 5109.
Escaped Identifiers		
An escaped identifier cannot contain a backslash character (\).	Affects hierarchical insertion and testbench generation.	Change the identifier in RTL or use Tessent Shell design editing.
An escaped identifier cannot contain braces ({}).	Affects hierarchical insertion and testbench generation.	Change the identifier in RTL or use Tessent Shell design editing.
An escaped identifier cannot contain an opening parenthesis or bracket that is not closed.	Affects hierarchical insertion and testbench generation.	Change the identifier in RTL or use Tessent Shell design editing.
Clocks can be added on ports with simple escaped names (example: "\port1 "), but not on ports with real escaped names (example: "\port[1] ").	Affects ETChecker (it is impossible to write the *.etCheckerInfo* file if clocks are added on ports with real escaped names.)	Change the identifier in RTL or use Tessent Shell design editing.
Simple escaped clock names (example: {b1.\wr_clk1 }) appear in ETChecker's *.out_dofile* with double quotation marks (example: "{b1.\wr_clk1 }").	These quoted names are not recognized in Tessent Shell and errors result.	Edit the clock names in the *.out_dofile* to remove the double quotation marks.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table B-1. Verilog and SystemVerilog Limitations (cont.)

Limitation	Effect on Flow	Workaround and Link to Description
If a Verilog identifier (meaning a design object name) does not include special characters but is escaped with a leading backslash, Tessent removes the backslash and the trailing space from the object name.	The backslash does not appear in reports and files written by the tool. Any user interactions must also use only the non-escaped name. All Tessent Shell products are affected.	Update user scripts to eliminate redundant backslashes.
Even if allowed by the language, a Verilog identifier that includes a dollar sign ($) is not supported.	Affects hierarchical insertion and testbench generation.	Update any Verilog RTL that uses dollar sign ($) in an identifier.

Related Topics

read_verilog

HDL Limitations in the Tessent Shell Flow for VHDL

The following HDL limitations apply to VHDL.

The following table lists and categorizes these limitations.

Table B-2. VHDL Limitations

Limitation	Effect on Flow	Workaround and Link to Description
New Module Insertion		
Insertion of an instance of a VHDL entity containing a port declared using an unconstrained array type is not supported.	Affects design editing.	Always use constrained array types for ports which are arrays in VHDL entities to be inserted into the design. See "Insertion of VHDL Entities with Unconstrained Ports" on page 5110.
Insertion of an instance of a VHDL entity containing uninitialized generics is not supported.	Affects design editing.	Always initialize any generics in VHDL entities to be inserted into the design. See "Insertion of VHDL Entities with Uninitialized Generics" on page 5110.
Complex Port Types		

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table B-2. VHDL Limitations (cont.)

Limitation	Effect on Flow	Workaround and Link to Description
Ports/nets declared using a complex type cannot be addressed in Tcl editing commands.	Affects design editing.	Replace the complex type usage with equivalent expanded logic. See "Editing of Ports and Nets of Complex Types" on page 5111.
The array of array and record constructs cannot be used in the definitions of root module ports which drive or receive data from inserted DFT logic.	Affects testbench generation.	Use simple vectors. See "Array of Array and Record Types in Port Declarations in the Root Module of a Design" on page 5111.
When boundary scan needs to be inserted into a design, top-level port declarations at the chip level are limited to scalars and simple vectors.	Affects tracing and design editing.	Replace the complex type usage with equivalent expanded logic. See "Complex Types in Top-Level Ports at the Chip Level" on page 5111.
A port declared in the root module of a child block or core cannot be declared using an unconstrained array type.	Affects design elaboration, hierarchical insertion and testbench generation.	Use a constrained array type. See "Ports with Unconstrained Array Types in the Root Design Entity of a Child Block" on page 5112.
Procedures, Functions, and Process Statements		
Tracing through logic containing procedure calls, non-constant function calls, and/or process statements is not supported.	Affects tracing.	See "Procedures, Functions, and Process Statements for VHDL" on page 5112.
Configurations		
Format preservation is not done when editing VHDL configurations. Such edits are typically needed when uniquifying an entity/architecture pair which appears in a configuration.	Affects design editing.	Avoid the use of pragmas in configurations as they are not preserved when the configuration needs to be edited.See "Configurations for VHDL" on page 5112.
When there are multiple configurations for the same entity, only the elaborated / active one can be edited. The inactive configurations may need manual updates to avoid parsing errors later in the flow.	Affects design editing.	Manually update inactive configurations when needed. See "Configurations for VHDL" on page 5112.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table B-2. VHDL Limitations (cont.)

Limitation	Effect on Flow	Workaround and Link to Description

If a configuration needs to be edited, it needs to be of the following form in the original design, i.e., before insertion:
`configuration` *`config_name`* `of` *`entity_name`* `is`
`...`
`end [configuration]` *`config_name`*
	Affects design editing.	The workaround is described in the Limitation column. See "Configurations for VHDL" on page 5112.

VHDL 2008		
VHDL 2008 is not fully supported.	Affects design compilation / elaboration and design editing.	Avoid the unsupported VHDL 2008 constructs. See "VHDL 2008" on page 5112.
Parameterized Designs		
For a generic declared in the root module of a child block or core which is used in logic that is edited during insertion, such a generic cannot be overridden in the instantiation of the child block.	Affects hierarchical insertion.	Use a fixed value in place of the generic or set the default to the same value as the override. See "Generic Overrides in the Instantiation of the Root Design Entity of a Child Block" on page 5114.
Extended Identifiers		
An extended identifier cannot contain a backslash character (\).	Affects hierarchical insertion and testbench generation.	Change the identifier in RTL or use Tessent Shell design editing.
An extended identifier cannot contain an opening parenthesis, brace, or bracket that is not closed.	Affects hierarchical insertion and testbench generation.	Change the identifier in RTL or use Tessent Shell design editing.
IJTAG Graybox		
When using ijtag_graybox, the case of pinnames in the instantiation of the DUT must match the case of the IJTAG graybox.	Affects design optimization.	Change the pin names to match.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SystemVerilog Interfaces

Note the following when using SystemVerilog Interfaces.

Multi-dimensional Instantiation of SystemVerilog Interfaces 5091
SystemVerilog Interfaces Cannot Be Edited 5091
Outside Connections on Interface Type Ports Cannot be Intercepted 5091
Insertion of an Instance of a Module With a SystemVerilog Interface Port Type 5092
SystemVerilog Generic Interface Port Declarations in the Root Module of a Child Block 5092
SystemVerilog Interfaces in Task or Function Invocation 5093
SystemVerilog Interfaces With Logic 5094
Modport Expressions 5094

Multi-dimensional Instantiation of SystemVerilog Interfaces

Tessent tools do not support multi-dimensional instantiations of a SystemVerilog interface. Connections in module port lists which utilize such interface instantiations cannot be traced or modified.

The following example illustrates a multi-dimensional instantiation of a SystemVerilog interface:

```
interface intfA;
…
endinterface

module top ( input in1, output o1 );
…
intfA my_intf_array [3:0][4:0] (); // interface instantiation
…
endmodule
```

SystemVerilog Interfaces Cannot Be Edited

Tessent tools cannot modify the definition of a SystemVerilog interface.

Outside Connections on Interface Type Ports Cannot be Intercepted

Tessent tools cannot intercept outside connections on ports which are declared using a SystemVerilog interface type.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This limitation affects boundary scan insertion for cases in which pad cell keyports are declared using a SystemVerilog interface type. It also affects the insertion of custom logic. The following example illustrates the use of a SystemVerilog interface type in the declaration of a module port:

```
interface intfA;
…
endinterface

module joe ( intfA p1 );
…
endmodule
```

An outside connection on *p1* cannot be intercepted.

Insertion of an Instance of a Module With a SystemVerilog Interface Port Type

Tessent tools cannot insert an instance of a module which has a port declared with a SystemVerilog interface type.

SystemVerilog Generic Interface Port Declarations in the Root Module of a Child Block

In a bottom-up flow SystemVerilog generic interface port declarations cannot be used in the root module of a child block or core.

The following example illustrates a generic interface port:

```
module mem1 (interface pins, input clk);
…
endmodule
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SystemVerilog Interfaces in Task or Function Invocation

Multiple arguments which are defined as instantiations of the same SV interface type cannot be passed in a single task or function invocation. For example:

```
interface my_intf #( parameter SZ1 = 1);
  wire w1[SZ1-1: 0];
  wire w2[SZ1-1: 0];

// some logic around w1,w2 etc…
endinterface

my_intf #(10) intf_obj_1;
my_intf #(20) intf_obj_2;

…
// Single function,task call using objects of same interface
   my_func (intf_obj_1, intf_obj_2, <other_arguments>);
…
```

The work around is to uniquify the instantiations of the interface as follows:

```
interface my_intf #( parameter SZ1 = 1);
  wire w1[SZ1-1: 0];
  wire w2[SZ1-1: 0];

// some logic around w1,w2 etc…
endinterface

interface my_intf_1 #( parameter SZ1 = 1);
  wire w1[SZ1-1: 0];
  wire w2[SZ1-1: 0];


// some logic around w1,w2 etc…
endinterface

my_intf #(10) intf_obj_1;
my_intf_1 #(20) intf_obj_2;

…
// Single function,task call using objects of different interfaces
   my_func (intf_obj_1, intf_obj_2, <other_arguments>);
…
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SystemVerilog Interfaces With Logic

Tessent tools do not support introspection and editing of SystemVerilog interfaces with logic. This includes the following:

- A SystemVerilog interface declared with a port in its port list whose type is another SystemVerilog interface. Example:

```
interface joe;
...
endinterface
interface john( input in1, joe in2 );
...
endinterface
```

- A SystemVerilog interface declared with combinational logic or generate blocks inside.
- A SystemVerilog interface declared with an always block inside.
- An instantiation of a SystemVerilog interface within a parent SystemVerilog interface. Example:

```
interface joe;
...
endinterface
interface john( input in1 );
...
joe i1(...);
...
endinterface
```

Modport Expressions

Tessent tools do not support introspection and editing of modport expressions.

Here is an example of a modport expression:

```
modport A (output .P(r), input .Q(x) );
endinterface
```

Replace the above with the following to work around this limitation:

```
interface joe;
...
modport A (output r, input x);
endinterface
```

Nested Modules

Nested modules are not supported in SystemVerilog code which needs to be edited or introspected.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

An example of a nested module is as follows:

```
module my_core (input wire clk);
   submod i1 (…);
   module submod (…);
   …
   endmodule
endmodule
```

Configurations for Verilog

DFT insertion is supported in configured designs.

The following is an example of a Verilog configuration:

```
config myDesignConfig;
design myDesignLib.top;
// Specifies that myDesignLib.top are the library and cell of the
// top-level module in the design configured by myDesignConfig.

instance top.inst1 use myDesignLib.joe;
// Specifies that instance top.inst1 is to be bound to myDesignLib.joe.
endconfig
```

You can use Verilog configurations in your design as long as no modifications to the configuration entities are required.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SystemVerilog Enumerations

Note the following when using SystemVerilog enumerations.

SystemVerilog Enum Methods 5096
SystemVerilog Enum Type in Port Declarations 5096

SystemVerilog Enum Methods

Tessent tools cannot trace a signal through logic written using SystemVerilog enum methods such as next, prev, first, and last.

In the following example the return values of the first and last methods are used as alternatives in a case generate statement:

```
module joe ( input in1, output out1 );
typedef enum {S[2]} states_t;
parameter states_t p1 = S1;
case(p1)
     p1.first : begin
     assign out1 = p1.first;
     end
     p1.last : begin
     assign out1 = p1.last;
     end
endcase
endmodule
```

The signal *out1* cannot be traced through the above case generate statement.

SystemVerilog Enum Type in Port Declarations

Tessent tools cannot intercept ports which are declared using a SystemVerilog enum type. This limitation may affect the insertion of custom logic.

The following example illustrates a port declaration with an enum type:

```
module top( input enum {stop, go} in1, input clk, output out1 );
   reg r1;
   always_ff @(posedge clk)
     if( in1 == stop )
       r1 <= 1;
     else
       r1 <= 0;
   assign out1 = r1;
endmodule // top
```

The port *in1* cannot be intercepted, that is, the internal net *in1* cannot be disconnected from the port *in1* and connected to a new driver.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Complex Port Types for Verilog and SystemVerilog

Note the following when using complex port types.

Editing of Ports and Nets of Complex Types 5097
Unpacked Dimensions and Other Complex Types in Port Declarations of the Root Module of a Design. 5098
Complex Types in Top-Level Ports at the Chip Level 5098
Complex Object Permissions 5098

Editing of Ports and Nets of Complex Types

Ports and nets of the following types cannot appear in editing commands:

- User-defined types
- Structs (packed and unpacked)
- Unions
- Packed arrays of more than one dimension
- Unpacked arrays of one or more dimensions

The above limitation puts constraints on the types of connections which can be made with custom logic.

The following example illustrates a user-defined type:

```
module top();
   typedef logic [7:0] my_vec_type;
   my_vec_type w1, w2; // variables declared to be of type my_vec_type
endmodule // top
```

The following example illustrates a structure:

```
module top();
   struct {
      logic a;
      logic b;
   } my_struct;
   assign my_struct.a = 1'b0;
endmodule // top
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Unpacked Dimensions and Other Complex Types in Port Declarations of the Root Module of a Design

Unpacked dimensions and other complex types (see previous section for a list) cannot be used in the definitions of root module ports (including those in child blocks or cores and the chip-level design) which drive or receive data from inserted DFT logic.

An example of an unpacked dimension is the following:

```
input cb_flow_type [18:0];
```

Unpacked dimensions always appear to the right of the port identifier.

Complex Types in Top-Level Ports at the Chip Level

The insertion of boundary scan by Tessent tools imposes certain restrictions on top-level port declarations. In Verilog and SystemVerilog designs, top-level port declarations at the chip level are limited to one packed dimension and zero unpacked dimensions (scalars and simple vectors).

The following example shows a module port declaration with two packed dimensions, which is not allowed:

```
module top( input [1:0][7:0] in1, output out1 );
endmodule
```

The following example shows a module port declaration with one packed dimension and one unpacked dimension, which is not allowed:

```
module top( input [1:0] in1 [7:0], output out1 );
endmodule
```

Packed dimensions are those that appear to the left of the port identifier. Unpacked dimensions appear to the right of the port identifier.

Complex Object Permissions

Although you can edit simple object types such as scalar ports, pins, nets, and one-dimensional buses, editing complex objects is unsupported.

Methods of Editing Objects

You can edit simple objects with the following:

- Explicit commands such as create_connections or delete_connections

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Automatic fixes performed during DRC operations
- DFT insertion of Tessent instruments with user-defined logic

Complex objects include, for example, those defined by SystemVerilog interfaces, structures, and multidimensional arrays. The editing of complex objects is not yet available. The capability to edit complex objects will be added in a future release.

Safeguards to Prevent Complex Object Editing

The is_non_editable object attribute is set to false for RTL and quick synthesis views of objects inferred from complex types. Also, the is_non_editable_reason attribute for these objects is set to empty. However, certain complex types are not editable. For example:

- The sub-bundle objects defined by SystemVerilog Interfaces are not editable.
- The leaf objects defined by VHDL scalar data types, such as enumerate and integer, are not editable.

In these cases, the is_non_editable attribute is set to “true” and the is_non_editable_reason attribute is set to “is_part_of_complex_object”.

The Tessent DFT insertion infrastructure uses these attributes to prevent attempts at editing non-editable complex objects at all stages of the DFT process.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Automatic Fixes and Object Editing

When automatic fixes are attempted on complex objects, an error is raised when you run the check_design_rules command. See the following for an example:

```
//  --------------------------------
//  Begin circuit learning analyses.
//  --------------------------------
//  Learning completed, CPU time=0.05 sec.
//  Begin simulation of auto-generated load_unload procedure.
//  Simulation of load_unload procedure completed, CPU time=0.0 sec.
//  Scan segment = /top_RTL1_tessent_sib_sti_inst/ltest_so successfully
traced with scan_cells = 7.
//  7 scan cells have been identified in 1 scan segment.
//  Warning: 1 edge-triggered clock ports set to stable high. (D7)
//  Warning: There were 112 occurrences of bus contention in test
procedures. (E4)
//  Error: The pin 'u1/top_RTL1_tessent_mbist_bap_inst/
parallel_access_start_stop_sequencer/rtlcreg_bist_pipe/CP' (16197.1)
source tracing stopped at pin 'u1/top_RTL1_tessent_mbist_bap_inst/
tessent_persistent_cell_sys_clk_buf/A' (5248.1).
//        Correct the blocking conditions or use the 'add_clocks -period'
if this is an embedded oscillator
//         or 'add_clocks -reference' if this is an active PLL or clock
divider. (DFT_C6-1)
//  Error: The supplied port 'bap_ins[39]' is not editable because
//         the 'is_non_editable_reason' attribute of the port is
'is_part_of_complex_object'.
//  Error: The previous error was created when applying the auto
correction for DFT_C9. Use the
//         'add_dft_control_point -type async_set_reset' command to
specify a different location on the core side of the pad which is
editable.
//  Error: There was 1 DFT_C6 violation (Clock of scannable flip-flops not
properly sourced by a declared clock).
//  Error: Rules checking unsuccessful, cannot exit SETUP
mode.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Functions, Tasks, and Procedural Statements for Verilog and SystemVerilog

Tessent tools cannot trace a signal through logic written using a non-constant function call, a task invocation or a procedural statement.

SystemVerilog Extern Modules 5101

SystemVerilog Extern Modules

Tessent tools cannot modify the ports lists of any extern module declarations in SystemVerilog designs.

The following example illustrates an extern module declaration:

```
// Prototype of car
extern module car ( input wire in1, output wire out1 );

// Definition of car: the .* syntax tells the compiler to get
// the port list from the prototype
module car (.*);
endmodule
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Unrolled Generate Loops for Verilog and SystemVerilog

Note the following when using generate loops.

Unrolling of Generate Loops Side Effects 5102

Insertion of Custom Logic Connecting to a Port or Net in an Unrolled Generate Loop 5105

Unrolling of Generate Loops Side Effects

Tessent tools need to unroll generate loops for certain types of editing.

This type of transformation in Verilog design files can have undesired side effects. Unrolled generate loops in Verilog design files are, by default, written out as a series of if generate blocks. There is one if generate block for each value of the genvar index in the original generate loop. Text inside the loop such as compiler directives, inactive generate regions, and comments is not preserved. An option is available to preserve such text. In that case the unrolled loops are written out as a series of loop generate blocks of size one. There is one loop generate block for each value of the genvar index in the original generate loop. This is in contrast to how unrolled generate loops in edited SystemVerilog design files are written out: if generate blocks are always used to perform the unrolling and text for compiler directives, inactive generate regions, and comments is preserved. The reason for the different handling is due to lack of support in simple Verilog for localparam declarations inside if generate blocks - those are allowed in SystemVerilog. The localparam construct is needed for format preservation (see below). When the unrolling is done using if generate blocks, hierarchical paths (for example, paths in SDC constraints) which reference objects inside generate loops in Verilog design files still work in most tools. When the unrolling is done using loop generate blocks of size 1, those hierarchical paths are invalid without manual intervention. Timing scripts generated by Tessent TimingGen tool account for the use of loop generate blocks of size one; however, user-generated timing scripts are not updated and may be invalid after unrolling.

As mentioned above, you have control over the unrolling strategy which is used. The unrolling strategy is set via the -preserve_paths_for_unrolled_v2001_loops option in the set_insertion_options command. If the value is set to OFF, user-generated timing scripts may be affected.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The example below shows how unrolling is performed in Verilog 2001 design files for both values of the -preserve_paths_for_unrolled_v2001_loops option in the set_insertion_options command:

```
wire [1:0] in1, in2;
generate
genvar i;
for( i = 0; i < 2; i++ )
begin : joe
`ifdef TECHNOLOGY_32NM
     DUT i0 ( .A(in1[i]), .B(in2[i]), … );
`else
     DUT1 i0 ( .A(in1[i]), .B(in2[i]), … );
`endif
end for
end generate
```

Here is what is written after unrolling when the -preserve_paths_for_unrolled_v2001_loops option in the set_insertion_options command is set to OFF:

```
wire [1:0] in1, in2;
genvar i;
for( i = 0; i <=0 ; i++ ) begin : \joe[0]
`ifdef TECHNOLOGY_32NM
      DUT i0 ( .A(in1[i]), .b(in2[i]), … );
`else
      DUT1 i0 ( .A(in1[i]), .b(in2[i]), … );
`endif
end for
for( i = 1; i <= 1; i++ ) begin : \joe[1]
`ifdef TECHNOLOGY_32NM
      DUT i0 ( .A(in1[i]), .b(in2[i]), … );
`else
      DUT1 i0 ( .A(in1[i]), .b(in2[i]), … );
`endif
end for
```

Generate loops of size 1 are used to perform the unrolling. The compiler directives and the inactive region are preserved. Instance *i0* inside block *\joe[0]* must be referenced (for example in SDC constraints) using the following hierarchical path:

```
\joe[0] [0]/i0
```

Here is what is written after unrolling when the -preserve_paths_for_unrolled_v2001_loops option in the set_insertion_options command is set to ON (assuming that the ` ifdef region is inactive):

```
wire [1:0] in1, in2;
if(1) begin : \joe[0]
      DUT1 i0 ( .A(in1[0]), .b(in2[0]), … );
end
if(1) begin : \joe[1]
      DUT1 i0 ( .A(in1[1]), .b(in2[1]), … );
end
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

If generate blocks are used to perform the unrolling. The compiler directives and the inactive region are removed. Instance *i0* inside block *\joe[0]* can be referenced (for example, in SDC constraints) using either of the following hierarchical paths:

```
\joe[0] /i0
joe[0]/i0
```

If we treat the same example as a SystemVerilog design file, then here is what is written after unrolling, regardless of the setting of the -preserve_paths_for_unrolled_v2001_loops option in the set_insertion_options command:

```
wire [1:0] in1, in2;
if(1) begin : \joe[0]
localparam i = 0;
`ifdef TECHNOLOGY_32NM
      DUT i0 ( .A(in1[i]), .b(in2[i]), … );
`else
      DUT1 i0 ( .A(in1[i]), .b(in2[i]), … );
`endif
end
if(1) begin : \joe[1]
localparam i = 1;
`ifdef TECHNOLOGY_32NM
      DUT i0 ( .A(in1[i]), .b(in2[i]), … );
`else
      DUT1 i0 ( .A(in1[i]), .b(in2[i]), … );
`endif
end
```

If generate blocks are used to perform the unrolling. Since localparam declarations are allowed inside if generate blocks in SystemVerilog, the inactive region can be preserved. Without the localparam, the reference to the variable “i” would generate an error.

In summary the pros and cons of setting the -preserve_paths_for_unrolled_v2001_loops to 'off' (SDC paths are not preserved; however, text format, comments, and compiler directives are preserved) are as follows:

- **Pros —** Compiler directives, inactive regions, and comments are preserved inside unrolled generate loops in Verilog design files.
- **Cons —** User generated timing scripts which reference objects inside unrolled loops may not work. Manual intervention may be required to fix hierarchical paths in such scripts.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

To update the SDC paths changed when the -preserve_paths_for_unrolled_v2001_loops option is set to off, first create a Tcl list of paths to monitor. Next, create a Tcl dictionary of the paths after setting the current design and design level:

```
// create the list "path_list", then:
set paths [dict create]
foreach path $path_list {
    set obj [get_instances $path]
    set id [get_single_attribute_value -name object_id $obj]
    dict set paths $path $id
}
```

Next, create and process the DftSpecification:

```
create_dft_spec
set_insertion_options -preserve_paths_for_unrolled_v2001_loops off
process_dft_spec
set_system_mode setup
```

Create a new Tcl dictionary with the old path names and their corresponding new path names:

```
set new_paths [dict create]
dict for {key value} $paths {
    set id $value
    set new_name [get_single_name [get_design_objects -from_ids $id]]
    dict set new_paths $key $new_name
}
```

Create a report cross-referencing the new and old names:

```
dict for {key value} $new_paths {
    puts "old_name: $key"
    puts "new_name: $value"
}
```

Finally, use the new SDC path names to update the timing scripts for the design.

Insertion of Custom Logic Connecting to a Port or Net in an Unrolled Generate Loop

If custom logic is used to make connections to a net or port inside an unrolled generate loop, the escaped names used for the unrolled loop labels in the edited design file must be used in the custom logic.

For example, if the following path is used before the design is first edited:

```
A/B[0]/P1
```

then after loop unrolling, the required syntax is as follows:

```
A/\B[0] /P1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note the “\” character and the space character after the “]” character.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Parameterized Designs for Verilog and SystemVerilog

Note the following when using parameterized designs.

Parameter Overrides in the Instantiation of the Root Module of a Child Block. 5107
Support for the Defparam Construct. 5107

Parameter Overrides in the Instantiation of the Root Module of a Child Block

In a bottom-up flow, you cannot use parameter overrides in the instantiation of the root module of a child block or core if the overridden parameters are used in logic that the tool edits during insertion.

For example, you cannot use such a parameter as the loop count of a generate loop that instantiates memories or override parameters that impact the test logic of a child block after insertion.

Caution

You may use the "set_current_design -parameter_values" command to apply parameter overrides in the root module of a child block but only under the restrictions specified here.

Support for the Defparam Construct

Support for the defparam construct is limited to those for which the specified path is relative to the current scope.

For example:

```
module joe( input jPort );
  bar b1();
  bar b2();
  defparam b1.f1.p = 6;
  ...
endmodule

module bar( input bPort );
  foo f1();
  foo f2();
  ...
endmodule

module foo( input fPort );
  parameter p = 3;
  ...
endmodule
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Implicit Port Syntax

Note the following when using implicit port syntax.

SystemVerilog Implicit Port Syntax in the Instantiation of the Root Module of a Child Block 5108

SystemVerilog Implicit Port Syntax in the Instantiation of the Root Module of a Child Block

In a bottom-up flow SystemVerilog implicit port syntax (i.e., .*) cannot be used in the instantiation of the root module of a child block or core.

The following example illustrates the use of implicit port syntax:

```
module top( input in1, output out1 );
…
joe i1( .* );
endmodule

module joe( input in1, output out1 );
…
endmodule
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

SystemVerilog 2009

Tessent Shell supports reading and writing of all of the SV2009 constructs. However, a few constructs cannot be modified in Tessent Shell. Those are as follows:

- Hierarchical references starting from parent hierarchy
- Coverage constructs
  - covergroup/endgroup
- Class constructs
- Program block constructs
  - program/endprogram
- Assertion constructs
  - assert
  - assert property
  - property/endproperty
  - clocking/endclocking
  - sequence/endsequence
- Checker constructs
  - checker/endchecker
- Bind constructs

For example, if a net is renamed in a move_connections command and it appears in any of those constructs, that update is not reflected in the construct when the file containing the construct is written out.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

New Module Insertion in VHDL

Note the following when inserting VHDL entities.

Insertion of VHDL Entities with Unconstrained Ports 5110
Insertion of VHDL Entities with Uninitialized Generics 5110

Insertion of VHDL Entities with Unconstrained Ports

Insertion of VHDL entities containing ports defined using unconstrained array types is not supported.

See the section entitled "Ports with Unconstrained Array Types in the Root Design Entity of a Child Block" on page 5112 for an example of an unconstrained array type.

Insertion of VHDL Entities with Uninitialized Generics

Insertion of VHDL entities containing uninitialized generics is not supported. The following is an example:

```
entity Joe is
    generic ( N : INTEGER );
    port( in1 : in std_logic );
end entity;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Complex Port Types for VHDL

Note the following when using complex port types.

Editing of Ports and Nets of Complex Types 5111

Array of Array and Record Types in Port Declarations in the Root Module of a Design 5111

Complex Types in Top-Level Ports at the Chip Level 5111

Ports with Unconstrained Array Types in the Root Design Entity of a Child Block . . 5112

Editing of Ports and Nets of Complex Types

Net and port objects declared using an array of array or a record type cannot appear in editing commands.

The above limitation puts constraints on the types of connections which can be made with custom logic.

Array of Array and Record Types in Port Declarations in the Root Module of a Design

Array of array and record types cannot be used in the definitions of root module ports (including those in child blocks or cores and the chip-level design) which drive or receive data from inserted DFT logic.

The following example illustrates the use of an array of array in the declaration of a port:

```
package pck is
    type myArray is array( 1 downto 0) of bit_vector( 0 to 0 );
end;

entity john is
    port( in1 : in myArray; out1 : out bit);
end john;
```

Complex Types in Top-Level Ports at the Chip Level

The insertion of boundary scan by Tessent tools imposes certain restrictions on top-level port declarations. In VHDL designs top-level port declarations at the chip level must be scalars or simple vectors.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Ports with Unconstrained Array Types in the Root Design Entity of a Child Block

Ports declared in the root module of a child block or core cannot be declared using an unconstrained array type. The following is an example of an unconstrained array type:

```
type JOE is array (INTEGER range <>) of std_logic;
```

Procedures, Functions, and Process Statements for VHDL

Tessent tools cannot trace a signal through logic written using a non-constant function call, a procedure invocation or a process statement.

Configurations for VHDL

MemoryBIST and boundary scan insertion are not fully supported in configured designs.

Format is not preserved when configurations need to be edited. Therefore, pragmas and other comments are not preserved in edited design files which contain configurations.

In addition when there are multiple configurations for the same entity, only the elaborated / active one can be edited. The inactive configurations may need manual updates to avoid parsing errors later in the flow. For example, if a component referenced in a configuration is uniquified, then any inactive configuration(s) for the same entity may need manual updates.

In addition if a configuration needs to be edited, it needs to be of the following form in the original design, i.e., before insertion:

```
configuration config_name of entity_name is
...
end [configuration] config_name
```

In the above end statement, the keyword “configuration” is optional; the *config_name* is mandatory.

VHDL 2008

Tessent Shell supports the following VHDL 2008 constructs:

- New VHDL 2008 generate block syntax
- Unconstrained element support in arrays and records
- Logical reduction operator

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Expressions which appear directly in a port map (e.g., a boolean or logical expression whose evaluated result is connected to the named port)
- Reading of output ports
- Simplified sensitivity lists
- C-style block comments
- Operations involving both an array and a scalar (e.g., “A_bus and A_sel”; A_sel is 1 bit and it applies to all bits of A_bus)
- Extensions to bit string literals
- Slices in array aggregates

VHDL 2008 constructs which are not in the above list are currently not supported.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Parameterized Designs for VHDL

Note the following when using parameterized designs.

Generic Overrides in the Instantiation of the Root Design Entity of a Child Block . . . 5114

Generic Overrides in the Instantiation of the Root Design Entity of a Child Block

In a bottom-up flow, you cannot use generic overrides in the instantiation of the root design entity of a child block or core if those generics are used in logic that the tool edits during insertion. Furthermore, you must specify default values for such generics.

The following example illustrates a VHDL entity with two generics—one with a default value and one without:

```
entity joe is
    generic (g1:integer; g2:integer :=0);
    …
end;
```

Generic *g1* does not have a default value. The *g2* generic has a default value of 0.

You cannot use the above entity as the root of a child block if the *g1* generic is used in logic that the tool edits during insertion. If *g1* had been specified with a default value, you could use entity *joe* as the root of a child block only if it is instantiated in the parent block without overrides for the generics.

Also, you cannot override parameters that impact the test logic of a child block after insertion.

Caution

You can use the "set_current_design -parameter_values" command to apply parameter overrides in the root module of a child block but only under the restrictions specified here.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Appendix C
Getting Help

There are several ways to get help when setting up and using Tessent software tools. Depending on your need, help is available from documentation, online command help, and your Siemens representative.

The Tessent Documentation System 5115
Global Customer Support and Success 5116

The Tessent Documentation System

At the center of the documentation system is the InfoHub that supports both PDF and HTML content. From the InfoHub, you can access all locally installed product documentation, system administration documentation, videos, and tutorials. For users who want to use PDF, you have a PDF bookcase file that provides access to all the installed PDF files.

For information on defining default HTML browsers, setting up browser options, and setting the default PDF viewer, refer to the *Siemens® Software and Mentor® Documentation System* manual.

You can access the documentation in the following ways:

- **Shell Command —** On Linux platforms, enter mgcdocs at the shell prompt or invoke a Tessent tool with the -manual invocation switch.
- **File System —** Access the Tessent InfoHub or PDF bookcase directly from your file system, without invoking a Tessent tool. For example:

  HTML:

  ```
  firefox <software_release_tree>/doc/infohubs/index.html
  ```

  PDF:

  ```
  acroread <software_release_tree>/doc/pdfdocs/_tessent_pdf_qref.pdf
  ```

- **Application Online Help —** ou can get contextual online help within most Tessent tools by using the “help -manual” tool command. For example:

  > help dofile -manual

  This command opens the appropriate reference manual at the “dofile” command description.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Global Customer Support and Success

A support contract with Siemens Digital Industries Software is a valuable investment in your organization's success. With a support contract, you have 24/7 access to the comprehensive and personalized Support Center portal.

Support Center features an extensive knowledge base to quickly troubleshoot issues by product and version. You can also download the latest releases, access the most up-to-date documentation, and submit a support case through a streamlined process.

https://support.sw.siemens.com

If your site is under a current support contract, but you do not have a Support Center login, register here:

https://support.sw.siemens.com/register

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Index

— A —

A rules, 3191
add_atpg_constraints, 106
add_atpg_functions, 110
add_bist_capture_range, 113
add_black_box, 115
add_cell_constraints, 122
add_clocks, 136
add_edt_blocks, 247
add_false_paths, 252
add_fault_sites, 259
add_faults, 261
add_iddq_exceptions, 292
add_input_constraints, 299
add_lfsr_connections, 308
add_lfsr_taps, 310
add_lfsrs, 312
add_lists, 315
add_notest_points, 332
add_output_masks, 332
add_primary_inputs, 338
add_primary_outputs, 341
add_processors, 343
add_read_controls, 347
add_scan_chains, 358
add_scan_groups, 361
add_synchronous_clock_group, 391
add_tied_signals, 397
add_write_controls, 402
ALL_FIXED_CYCLES, 5009
ALL_FLATTEN_TIMING, 5009
ALL_MIN_SCAN_LOAD, 5012
ALL_NO_CYCLE_OPT, 5013
ALL_NO_LOOP, 5013
ALL_NO_PATTERN_TYPE, 5013
ALL_TIME_RESOLUTION, 5017
analyze_bus, 407
analyze_control_signals, 434
analyze_drc_violation, 437
analyze_fault, 439
analyze_graybox, 447
analyze_restrictions, 459
analyze_simulation_mismatches, 464
analyze_wrapper_cells, 479

— B —

BIST rules, 3211
Brules, 3211

— C —

C rules, 3222
Clock cone, 3235, 3241
Clocks, 2290
close_visualizer, 523
Commands
- add_atpg_constraints, 106
- add_atpg_functions, 110
- add_bist_capture_range, 113
- add_black_box, 115
- add_cell_constraints, 122
- add_cell_models, 129
- add_chain_masks, 131
- add_clocks, 136
- add_edt_blocks, 247
- add_false_paths, 252
- add_fault_sites, 259
- add_faults, 261
- add_iddq_exceptions, 292
- add_input_constraints, 299
- add_lfsr_connections, 308
- add_lfsr_taps, 310
- add_lfsrs, 312
- add_lists, 315
- add_notest_points, 332
- add_output_masks, 332
- add_primary_inputs, 338
- add_primary_outputs, 341
- add_processors, 343
- add_read_controls, 347

add_scan_chains, 358
add_scan_groups, 361
add_scan_instances, 364
add_synchronous_clock_group, 391
add_test_points, 397
add_tied_signals, 397
add_write_controls, 402
analyze_bus, 407
analyze_control_signals, 434
analyze_drc_violation, 437
analyze_fault, 439
analyze_graybox, 447
analyze_restrictions, 459
analyze_simulation_mismatches, 464
analyze_wrapper_cells, 479
close_visualizer, 523
compress_patterns, 528
create_capture_procedures, 539
create_flat_model, 571
create_initialization_patterns, 582
create_patterns, 606
delete_atpg_constraints, 655
delete_atpg_functions, 657
delete_bist_capture_range, 659
delete_black_box, 660
delete_capture_procedures, 666
delete_cell_constraints, 669
delete_cell_library, 671
delete_cell_models, 672
delete_clocks, 675
delete_design, 688
delete_edt_blocks, 703
delete_edt_configurations, 706
delete_false_paths, 708
delete_fault_sites, 711
delete_faults, 715
delete_iddq_exceptions, 733
delete_input_constraints, 736
delete_lfsr_connections, 746
delete_lfsr_taps, 747
delete_lfsrs, 748
delete_lists, 749
delete_multicycle_paths, 753
delete_nofaults, 758
delete_nonscan_instances, 761
delete_notest_points, 763
delete_output_masks, 766
delete_patterns, 771
delete_primary_inputs, 777
delete_primary_outputs, 779
delete_processors, 781
delete_read_controls, 784
delete_scan_chains, 789
delete_scan_groups, 791
delete_scan_instances, 792
delete_synchronous_clock_group, 804
delete_test_points, 807
delete_tied_signals, 809
delete_write_controls, 812
diagnose_failures, 813
display_diagnosis_report, 818
dofile, 823
echo, 825
find_design_names, 866
get_attribute_list, 898
get_attribute_option, 901, 907
get_attribute_value_list, 903
get_auxiliary_pins, 909
get_boundary_scan_port_option, 912
get_bundle_objects, 916
get_cdp_test_list, 926
get_clock_option, 927
get_clocks, 932
get_common_parent_instance, 935
get_config_elements, 936
get_config_messages, 940
get_config_value, 943
get_context, 958
get_current_design, 962
get_current_mode, 964
get_current_silicon_insight_setup, 966
get_current_simulation_context, 967
get_defaults_value, 969, 971
get_design_objects, 972
get_design_sources, 975
get_dfm_rules, 977
get_dft_cell, 979
get_dft_info_dictionary, 990
get_dft_signal, 997
get_dft_specification_requirement, 1005

get_drc_handling, 1007
get_equivalent_editable_node, 1008
get_fanins, 1010
get_fanouts, 1014
get_fault_type, 1018
get_gate_pins, 1020
get_icl_extraction_options, 1026
get_icl_fanins, 1027
get_icl_fanins_in_module, 1029
get_icl_fanouts, 1031
get_icl_fanouts_in_module, 1033
get_icl_instances, 1035
get_icl_module_parameter_list, 1043
get_icl_module_parameter_value, 1044
get_icl_modules, 1040
get_icl_network, 1046
get_icl_objects, 1048
get_icl_pins, 1055
get_icl_ports, 1059
get_icl_scan_interface_chain_list, 1062
get_icl_scan_interface_list, 1063
get_icl_scan_interface_port_list, 1065
get_icl_scan_mux, 1068
get_icl_scope, 1070
get_iclock_list, 1075
get_iclock_option, 1077
get_ijtag_instance_option, 1083
get_ijtag_instances, 1081
get_ijtag_retargeting_options, 1085
get_input_constraints, 1088
get_insert_test_logic_option, 1095
get_insertion_option, 1092
get_instance_polygons, 1098
get_instances, 1099
get_instrument_dictionary, 1106
get_instrument_parent_icl_module_list, 1112
get_iproc_argument_default, 1113
get_iproc_argument_list, 1115
get_iproc_body, 1116
get_iproc_list, 1117
get_layout_core_instance, 1120
get_license_queue_timeout, 1121
get_loadboard_loopback_option, 1127
get_logfile, 1128
get_logical_library_list, 1129
get_master_module_name, 1131
get_memory_instance_option, 1135
get_memory_instances, 1132
get_module_matching_options, 1138
get_module_parameter_list, 1142
get_module_parameter_value, 1144
get_modules, 1146
get_multiprocessing_option, 1153
get_name_list, 1155
get_nets, 1159
get_open_pattern_set, 1165
get_pattern_cycle_count, 1166
get_pattern_set_data, 1167
get_pattern_set_list, 1177
get_pattern_set_option, 1178, 1180
get_pins, 1181
get_port_name_list, 1192
get_ports, 1193
get_procfile_name, 1203
get_read_verilog_option, 1204
get_resource, 1205
get_run_synthesis_options, 1207
get_scan_chain_families, 1209, 1210
get_scan_elements, 1211
get_scan_modes, 1218
get_scratch_directory, 1219
get_silicon_insight_job_status, 1220
get_silicon_insight_option, 1221
get_simulation_context_list, 1224
get_simulation_library_sources, 1226
get_simulation_option, 1228
get_simulation_value_list, 1230
get_single_attribute_value, 1232
get_single_name, 1234
get_static_dft_signal_icall, 1240
get_synchronous_clock_groups, 1242
get_system_mode, 1243
get_tcl_shell_option, 1244
get_test_end_icall_list, 1245
get_test_point_type, 1254
get_test_points, 1255
get_test_setup_icall_list, 1260
get_testbench_simulation_options, 1261
get_timeplate_list, 1264

get_tool_info, 1265
get_tool_option, 1267
get_trace_flat_model_option, 1268
get_transcript_style, 1271
get_tsdb_info, 1273
get_tsdb_list, 1279
get_tsdb_output_directory, 1281
get_validated_objects, 1282
get_write_patterns_options, 1285
get_xclock_handling, 1288
get_xy_instance, 1289, 1290
help, 1291
identify_redundant_faults, 1310
open_visualizer, 1440
order_patterns, 1444
read_cell_library, 1480
read_cpf, 1486
read_failures, 1499
read_fault_sites, 1507
read_faults, 1519
read_flat_model, 1530
read_modelfile, 1539
read_patterns, 1543
read_procfile, 1555
read_sdc, 1559
read_sdf, 1566
read_upf, 1569, 1571
report_aborted_faults, 1662
report_atpg_constraints, 1665
report_atpg_functions, 1666
report_atpg_timing, 1667
report_bist_capture_range, 1681
report_black_box, 1682
report_bus_data, 1689
report_bypass_chains, 1692
report_capture_procedures, 1695
report_cell_constraints, 1703
report_clock_domains, 1712
report_clocks, 1719
report_compactor_connections, 1724
report_control_signals, 1741
report_drc_rules, 1776
report_edt_blocks, 1799
report_edt_instances, 1811
report_edt_pins, 1819
report_environment, 1822
report_external_simulator, 1824
report_failures, 1825
report_false_paths, 1831
report_fault_sites, 1840
report_faults, 1848
report_feedback_paths, 1865
report_flattener_rules, 1867
report_gates, 1869
report_graybox_statistics, 1895
report_id_stamp, 1906
report_iddq_exceptions, 1908
report_input_constraints, 1915
report_lfsr_connections, 1948
report_lfsrs, 1949
report_lists, 1951
report_loops, 1955
report_measure_cycles, 1960
report_mismatch_sources, 1987
report_multicycle_paths, 2001
report_multiprocessing_options, 2006
report_nofaults, 2009
report_nonscan_cells, 2011
report_nonscan_models, 2014
report_notest_points, 2015
report_output_masks, 2017
report_power_data, 2035
report_power_metrics, 2794
report_primary_inputs, 2048
report_primary_outputs, 2050
report_procedures, 2052
report_processors, 2056
report_read_controls, 2059
report_resources, 2064
report_scan_cells, 2070
report_scan_chains, 2082
report_scan_groups, 2093
report_scan_volume, 2116
report_seq_transparent_procedures, 2120
report_statistics, 2141
report_synchronous_clock_groups, 2161
report_test_end_icall, 2166
report_test_logic, 2167
report_test_point_statistics, 2172
report_test_points, 2168

report_test_setup_icall, 2174
report_test_stimulus, 2175
report_testbench_simulation_options, 2184
report_tied_signals, 2188
report_timeplates, 2190
report_wrapper_cells, 2199
report_write_controls, 2203
reset_au_faults, 2213
reset_bypass_chains, 2215
reset_compactor_connections, 2217
reset_design, 2219
reset_di_faults, 2222
reset_state, 2231, 2235
set_abort_limit, 2284
set_atpg_fill, 2286
set_atpg_limits, 2287
set_atpg_timing, 2290
set_attribute_value, 2299
set_au_analysis, 2299
set_bist_chain_test, 2309
set_bist_trace, 2314
set_bus_handling, 2323
set_bus_simulation, 2326
set_bypass_chains, 2328
set_capture_clock, 2331
set_capture_procedures, 2334
set_cell_library_options, 2344
set_chain_test, 2351
set_checkpointing_options, 2355
set_clock_off_simulation, 2371
set_clock_option, 2372
set_clock_restriction, 2375
set_compactor_connections, 2385
set_contention_check, 2391
set_current_edt_block, 2424
set_current_edt_configuration, 2427
set_design_include_directories, 2447
set_design_macros, 2455
set_diagnosis_options, 2457, 2466, 2473
set_drc_handling, 2489
set_driver_restriction, 2498
set_edt_instances, 2502
set_edt_mapping, 2505
set_edt_options, 2508
set_edt_pins, 2521
set_external_capture_options, 2537
set_external_simulator, 2540
set_fails_report, 2542
set_fault_mode, 2545
set_fault_sampling, 2547
set_fault_subclass_analysis, 2550
set_fault_type, 2553
set_flattener_rule_handling, 2567
set_gate_level, 2570
set_gate_report, 2571
set_gzip_options, 2606
set_iddq_checks, 2621
set_internal_fault, 2656
set_internal_name, 2657
set_io_mask, 2658
set_latch_handling, 2659
set_learn_report, 2676
set_lfsrs, 2679
set_list_file, 2683
set_logfile_handling, 2693
set_loop_handling, 2697
set_macrotest, 2716
set_multiprocessing_options, 2743
set_net_dominance, 2752
set_net_resolution, 2754
set_number_shifts, 2755
set_observation_point, 2757
set_output_masks, 2758
set_pattern_buffer, 2764
set_pattern_classification, 2765
set_pattern_source, 2778
set_pattern_type, 2787
set_possible_credit, 2793
set_procedure_cycle_checking, 2807
set_procfile_name, 2811
set_ram_initialization, 2817
set_random_clocks, 2818
set_random_patterns, 2820
set_shadow_check, 2851
set_skewed_load, 2865
set_split_capture_cycle, 2866
set_stability_check, 2873
set_static_learning, 2881
set_system_mode, 2883

set_test_logic, 2891
set_tied_signals, 2923
set_timing_exceptions_handling, 2925
set_tla_loop_handling, 2932
set_trace_report, 2939
set_transient_detection, 2943
set_transition_holdpi, 2945
set_visualizer_preferences, 2952
set_xclock_handling, 2968
set_z_handling, 2970
set_zhold_behavior, 2972
setup_test_point_insertion, 2906, 2923
simulate_patterns, 2983
stil2mgc, 77
update_implication_detections, 3019
write_atpg_setup, 3027
write_design, 3040
write_edt_files, 3059
write_failures, 3074
write_fault_sites, 3079
write_faults, 3083
write_flat_model, 3096
write_history, 3103
write_loops, 3103
write_modelfile, 3108
write_paths, 3109
write_patterns, 3109
write_primary_inputs, 3142
write_primary_outputs, 3143
write_procedure_testbench, 3144
write_procfile, 3148
write_scan_order, 3150
Compac tor
specifying, 2515
Compactor
Xpr ess compactor m asking, 2515
Xpress compactor masking, 2515
compress_patterns, 528
create_capture_procedures, 539
create_flat_model, 571
create_initialization_patterns, 582
create_patterns, 606
CTL_ONE_PAT, 5018
CTL_STIL_0, 5018
CTL_STIL_1, 5019
CTL_TRIM_PULSE, 5061

— D —

D rules, 3267, 3280
delete_atpg_con straints, 655
delete_atpg_functions, 657
delete_bist_capture_range, 659
delete_black_box, 660
delete_capture_procedures, 666
delete_cell_constraints, 669
delete_cell_library, 671
delete_clocks, 675
delete_design, 688
delete_edt_blocks, 703
delete_edt_configurations, 706
delete_false_paths, 708
delete_fault_sites, 711
delete_faults, 715
delete_iddq_exceptions, 733
delete_input_constraints, 736
delete_lfsr_connections, 746
delete_lfsr_taps, 747
delete_lfsrs, 748
delete_lists, 749
delete_multicycle_paths, 753
delete_nofaults, 758
delete_notest_points, 763
delete_output_masks, 766
delete_patterns, 771
delete_primary_inputs, 777
delete_primary_outputs, 779
delete_processors, 781
delete_read_controls, 784
delete_scan_chains, 789
delete_scan_groups, 791
delete_synchronous_clock_group, 804
delete_tied_signals, 809
delete_write_controls, 812
Design Rule Checking, how to display violations, 1776
Design rules checking
basic troubleshooting, 3170
BIST rules, 3211
clock rules, 3222
data rules, 3267, 3280
EDT rules, 3445

extra rules, 3312
general rules, 3368
procedure rules, 3466
RAM rules, 3191
reporting gate data, 3173
scannability rules, 3539
setting gate data, 3172
setting gate level, 3171
setting rule handling, 3170
timing rules, 3623
trace rules, 3587
with ATPG analysis, 3170
diagnose_failures, 813
display_diagnosis_report, 818
dofile, 823
DRC messages
oscillation limitation, 3647
other DRC messages, 3647
RAM summary results and test capability, 3647
DRC, how to display violations, 1776

— E —

E rules, 3312
Effect co ne, 3235
EMPTY, 3176, 3200, 3255, 3278, 3329, 3373, 3463, 3497, 3545, 3608, 3630

— F —

Faults
timing-critical, 2290
FDTL_TEST_NAME, 5020
FDTL_TEST_NUM, 5021
FDTL_TEST_SUFFIX, 5021
find_design_names, 866
FTDL parameter file example, 5022
FTDL_DESIGNER, 5019
FTDL_MAX_PAT_BLOCK, 5019
FTDL_REVISION, 5020
FTDL_ZMODE_MES, 5021

— G —

G rules, 3368
get_attribute_list, 898
get_attribute_option, 901
get_attribute_value_list, 903
get_attributed_objects, 907
get_auxiliary_pins, 909
get_boundary_scan_port_option, 912
get_bundle_objects, 916
get_cdp_test_list, 926
get_clock_option, 927
get_clocks, 932
get_common_parent_instance, 935
get_config_elements, 936
get_config_messages, 940
get_config_value, 943
get_context, 958
get_current_design, 962
get_current_mode, 964
get_current_silicon_insight_setup, 966
get_current_simulation_context, 967
get_defaults_value, 969
get_design_level, 971
get_design_objects, 972
get_design_sources, 975
get_dfm_rules, 977
get_dft_cell, 979
get_dft_info_dictionary, 990
get_dft_signal, 997
get_dft_specification_requirement, 1005
get_drc_handling, 1007
get_equivalent_editable_node, 1008
get_fanins, 1010
get_fanouts, 1014
get_fault_type, 1018
get_gate_pins, 1020
get_icl_extraction_options, 1026
get_icl_fanins, 1027
get_icl_fanins_in_module, 1029
get_icl_fanouts, 1031
get_icl_fanouts_in_module, 1033
get_icl_instances, 1035
get_icl_module_parameter_list, 1043
get_icl_module_parameter_value, 1044
get_icl_modules, 1040
get_icl_network, 1046
get_icl_objects, 1048
get_icl_pins, 1055
get_icl_ports, 1059
get_icl_scan_interface_chain_list, 1062

get_icl_scan_interface_list, 1063
get_icl_scan_interface_port_list, 1065
get_icl_scan_mux, 1068
get_icl_scope, 1070
get_iclock_list, 1075
get_iclock_option, 1077
get_ijtag_instance_option, 1083
get_ijtag_instances, 1081
get_ijtag_retargeting_options, 1085
get_input_constraints, 1088
get_insert_test_logic_option, 1095
get_insertion_option, 1092
get_instance_polygons, 1098
get_instances, 1099
get_instrument_dictionary, 1106
get_instrument_parent_icl_module_list, 1112
get_iproc_argument_default, 1113
get_iproc_argument_list, 1115
get_iproc_body, 1116
get_iproc_list, 1117
get_layout_core_instance, 1120
get_license_queue_timeout, 1121
get_loadboard_loopback_option, 1127
get_logfile, 1128
get_logical_library_list, 1129
get_master_module_name, 1131
get_memory_instance_option, 1135
get_memory_instances, 1132
get_module_matching_options, 1138
get_module_parameter_list, 1142
get_module_parameter_value, 1144
get_modules, 1146
get_multiprocessing_option, 1153
get_name_list, 1155
get_nets, 1159
get_open_pattern_set, 1165
get_pattern_cycle_count, 1166
get_pattern_set_data, 1167
get_pattern_set_list, 1177
get_pattern_set_option, 1178
get_pin_name_list, 1180
get_pins, 1181
get_port_name_list, 1192
get_ports, 1193
get_procfile_name, 1203
get_read_verilog_option, 1204
get_resource, 1205
get_run_synthesis_options, 1207
get_scan_chain_families, 1209
get_scan_chain_option, 1210
get_scan_elements, 1211
get_scan_modes, 1218
get_scratch_directory, 1219
get_silicon_insight_job_status, 1220
get_silicon_insight_option, 1221
get_simulation_context_list, 1224
get_simulation_library_sources, 1226
get_simulation_option, 1228
get_simulation_value_list, 1230
get_single_attribute_value, 1232
get_single_name, 1234
get_static_dft_signal_icall, 1240
get_synchronous_clock_groups, 1242
get_system_mode, 1243
get_tcl_shell_option, 1244
get_test_end_icall_list, 1245
get_test_point_type, 1254
get_test_points, 1255
get_test_setup_icall_list, 1260
get_testbench_simulation_options, 1261
get_timeplate_list, 1264
get_tool_info, 1265
get_tool_option, 1267
get_trace_flat_model_option, 1268
get_transcript_style, 1271
get_tsdb_info, 1273
get_tsdb_list, 1279
get_tsdb_output_directory, 1281
get_validated_objects, 1282
get_write_patterns_options, 1285
get_xclock_handling, 1288
get_xy_instance, 1289, 1290
group timing information, 5066
Group TITDL timing information, 5066

— H —

help, 1291
Hold_pi, 2290

— I —

identify_redundant_faults, 1310

Instance name, 5030
Introduction, 59

— K —

K rules, 3445
Keywords
ALL_FIXED_CYCLES, 5009
ALL_FLATTEN_TIMING, 5009
ALL_MIN_SCAN_LOAD, 5012
ALL_NO_CYCLE_OPT, 5013
ALL_NO_LOOP, 5013
ALL_NO_PATTERN_TYPE, 5013
ALL_TIME_RESOLUTION, 5017
CTL_ONE_PAT, 5018
CTL_STIL_0, 5018
CTL_STIL_1, 5019
CTL_TRIM_PULSE, 5061
FTDL_DESIGNER, 5019
FTDL_MAX_PAT_BLOCK, 5019
FTDL_REVISION, 5020
FTDL_TEST_NAME, 5020
FTDL_TEST_NUM, 5021
FTDL_TEST_SUFFIX, 5021
FTDL_ZMODE_MES, 5021
SIM_COMPARE_X, 5024
SIM_EARLY_RELEASE, 5028
SIM_EARLY_RELEASE_TIME, 5028
SIM_INCLUDE, 5030
SIM_INSTANCE_NAME, 5030
SIM_MASK_FILE, 5031
SIM_MISCOMPARE_LIMIT, 5032
SIM_MODULE_INCLUDE, 5033
SIM_SHIFT_DEBUG, 5038
SIM_STATUS_MSG, 5041
SIM_TIMEPLATE_COM, 5042
SIM_TOP_NAME, 5042
SIM_VECTOR_COMM, 5042
SIM_VECTYPE_SIGNAL, 5044
STIL_2005_SCAN, 5045
STIL_MIN_QUOTE, 5052
STIL_NOMESURE_CLOCK, 5054
STIL_PAT_ANN, 5055
STIL_PAT_CMT, 5055
STIL_PAT_LAB, 5056
STIL_SCAN_ANN1, 5056
STIL_STRUCTURAL, 5057
STIL_TI_TITLE, 5060
STIL_VECTOR_ANN, 5061
TITDL_CHAIN_SETTYPE, 5065
TITDL_CUSTOMER, 5065
TITDL_GROUP_TIMING, 5066
TITDL_KEEP_SCANOUT, 5066
TITDL_LIBRARY, 5066
TITDL_PARTNUM, 5067
TITDL_PSEUDO_PREFIX, 5067
TITDL_REVISION, 5068
TITDL_SETNAME, 5068
TITDL_SETTYPE, 5069
WGL_ADD_LASTX, 5069
WGL_ALT_BIDI, 5069
WGL_ALT_VECT_ANN, 5071
WGL_EDGE_STROBE, 5071
WGL_FULL_CHAIN, 5072
WGL_FULL_SCANGROUP, 5074
WGL_GROUP_PIN, 5074
WGL_INV_SC, 5075
WGL_NOMEASURE_CLOCK, 5076
WGL_ONE_ILLINOIS, 5076
WGL_PATTERN_NAME, 5077
WGL_TRIM_LEADING_P, 5077
WGL_TRIM_PULSE, 5077
WGL_VECTOR_ANN, 5078
WGL_VERG_ESC, 5079
WGL_VTRAN_PADSC, 5079

— M —

Mask_po, 2290
Masking scan chains, 131

— N —

newlink
BondingConfiguratonis99EmbeddedBScan, 4112
newlink Interface99, 4234
newlink
Scan99pre_scan_drc_on_boundary_only, 4798
newlink Tap/HostSTap99SHSI, 4041
newlink Tap/HostSTap99STapSHSI, 4040
Non-scan initialization values, 361

— O —

open_visualizer, 1440
order_patterns, 1444
Oscillation limitation, 3647

— P —

P rules, 3466
Parameter file keywords
    FTDL_DESIGNER, 5019
    FTDL_MAX_PAT_BLOCK, 5019
    FTDL_REVISION, 5020
    FTDL_TEST_NAME, 5020
    FTDL_TEST_NUM, 5021
    FTDL_TEST_SUFFIX, 5021
    FTDL_ZMODE_MES, 5021

— R —

RAM
    rules checking, 3191
RAM summary result s and test capability, 3647
read_cell_library, 1480
read_cpf, 1486
read_failures, 1499
read_fault_sites, 1507
read_faults, 1519
read_modelfile, 1539
read_patterns, 1543
read_procfile, 1555
read_sdc, 1559
read_sdf, 1566
read_upf, 1569
read_verilog, 1571
report_aborted_faults, 1662
report_atpg_constraints, 1665
report_atpg_functions, 1666
report_atpg_timing, 1667
report_bist_capture_range, 1681
report_black_box, 1682
report_bus_data, 1689
report_bypass_chains, 1692
report_capture_procedures, 1695
report_cell_constraints, 1703
report_clock_domains, 1712
report_clocks, 1719
report_compactor_connections, 1724
report_drc_rules, 1776
report_edt_blocks, 1799
report_edt_instances, 1811
report_edt_pins, 1819
report_environment, 1822
report_external_simulator, 1824
report_failures, 1825
report_false_paths, 1831
report_fault_sites, 1840
report_faults, 1848
report_feedback_paths, 1865
report_flattener_rules, 1867
report_gates, 1869
report_graybox_statistics, 1895
report_id_stamp, 1906
report_iddq_exceptions, 1908
report_input_constraints, 1915
report_lfsr_connections, 1948
report_lfsrs, 1949
report_lists, 1951
report_loops, 1955
report_measure_cycles, 1960
report_mismatch_sources, 1987
report_multicycle_paths, 2001
report_multiprocessing_options, 2006
report_nofaults, 2009
report_nonscan_cells, 2011
report_notest_points, 2015
report_output_masks, 2017
report_power_data, 2035
report_power_metrics, 2794
report_primary_inputs, 2048
report_primary_outputs, 2050
report_procedures, 2052
report_processors, 2056
report_read_controls, 2059
report_resources, 2064
report_scan_cells, 2070
report_scan_chains, 2082
report_scan_groups, 2093
report_scan_volume, 2116
report_seq_transparent_procedures, 2120
report_synchronous_clock_groups, 2161
report_test_end_icall, 2166
report_test_point_statistics, 2172

report_test_setup_icall, 2174
report_test_stimulus, 2175
report_testbench_simulation_options, 2184
report_tied_signals, 2188
report_timeplates, 2190
report_write_controls, 2203
reset_au_faults, 2213
reset_bypass_chains, 2215
reset_compactor_connections, 2217
reset_design, 2219
reset_di_faults, 2222
reset_state, 2231, 2235

— S —

Scan chain trace rules, 3587
Scan chains
    masking, 131
Scannability rules checking, 3539
SDF file, 1566
set_abort_limit, 2284
set_atpg_fill, 2286
set_atpg_limits, 2287
set_atpg_timing, 2290
set_attribute_value, 2299
set_au_analysis, 2299
set_bist_chain_test, 2309
set_bist_trace, 2314
set_bus_handling, 2323
set_bus_simulation, 2326
set_bypass_chains, 2328
set_capture_clock, 2331
set_capture_procedures, 2334
set_cell_library_options, 2344
set_chain_test, 2351
set_checkpointing_options, 2355
set_clock_off_simulation, 2371
set_clock_option, 2372
set_clock_restriction, 2375
set_compactor_connections, 2385
set_contention_check, 2391
set_current_edt_block, 2424
set_current_edt_configuration, 2427
set_design_include_directories, 2447
set_design_macros, 2455
set_diagnosis_options, 2457, 2466, 2473
set_drc_handling, 2489
set_driver_restriction, 2498
set_edt_options, 2508
set_edt_pins, 2521
set_external_capture_options, 2537
set_external_simulator, 2540
set_fails_report, 2542
set_fault_mode, 2545
set_fault_sampling, 2547
set_fault_subclass_analysis, 2550
set_fault_type, 2553
set_flattener_rule_handling, 2567
set_gate_level, 2570
set_gate_report, 2571
set_gzip_options, 2606
set_iddq_checks, 2621
set_internal_fault, 2656
set_internal_name, 2657
set_io_mask, 2658
set_learn_report, 2676
set_lfsrs, 2679
set_list_file, 2683
set_logfile_handling, 2693
set_loop_handling, 2697
set_macrotest, 2716
set_multiprocessing_options, 2743
set_net_dominance, 2752
set_net_resolution, 2754
set_number_shifts, 2755
set_observation_point, 2757
set_output_masks, 2758
set_pattern_buffer, 2764
set_pattern_classification, 2765
set_pattern_source, 2778
set_pattern_type, 2787
set_possible_credit, 2793
set_procedure_cycle_checking, 2807
set_procfile_name, 2811
set_ram_initialization, 2817
set_random_clocks, 2818
set_random_patterns, 2820
set_shadow_check, 2851
set_skewed_load, 2865
set_split_capture_cycle, 2866
set_stability_check, 2873
set_static_learning, 2881

set_system_mode, 2883
set_tied_signals, 2923
set_timing_exceptions_handling, 2925
set_tla_loop_handling, 2932
set_trace_report, 2939
set_transient_detection, 2943
set_transition_holdpi, 2945
set_visualizer_preferences, 2952
set_xclock_handling, 2968
set_z_handling, 2970
set_zhold_behavior, 2972
setup_test_point_insertion, 2906
SIM_COMPARE_X, 5024
SIM_EARLY_RELEASE, 5028
SIM_EARLY_RELEASE_TIME, 5028
SIM_INCLUDE, 5030
SIM_INSTANCE_NAME, 5030
SIM_MASK_FILE, 5031
SIM_MISCOMPARE_LIMIT, 5032
SIM_MODULE_INCLUDE, 5033
SIM_SHIFT_DEBUG, 5038
SIM_STATUS_MSG, 5041
SIM_TIMEPLATE_COM, 5042
SIM_TOP_NAME, 5042
SIM_VECTOR_COMM, 5042
SIM_VECTYPE_SIGNAL, 5044
simulate_patterns, 2983
Simulation mismatches
    for ChainTest patterns, 5032
Slack margin, 2290
STIL_2005_SCAN, 5045
STIL_MIN_QUOTE, 5052
STIL_NOMESURE_CLOCK, 5054
STIL_PAT_ANN, 5055
STIL_PAT_CMT, 5055
STIL_PAT_LAB, 5056
STIL_SCAN_ANN1, 5056
STIL_STRUCTURAL, 5057
STIL_VECTOR_ANN, 5061
stil2mgc script, 77
Suffix
    adding to the test block, 5021

— T —

T rules, 3587
test block
    adding a suffix, 5021
Test Procedure File, definition of, 65, 71
Test structures
    identification interactions, 1713
TI title block
    changing with STIL_TI_TITLE, 5060
Timing-aware ATPG
    setting up, 2290
Timing-critical faults, 2290
TITDL, 5066
TITDL_CHAIN_SETTYPE, 5065
TITDL_CUSTOMER, 5065
TITDL_GROUP_TIMING, 5066
TITDL_KEEP_SCANOUT, 5066
TITDL_LIBRARY, 5066
TITDL_PARTNUM, 5067
TITDL_PSEUDO_PREFIX, 5067
TITDL_REVISION, 5068
TITDL_SETNAME, 5068
TITDL_SETTYPE, 5069
Title block
    changing with STIL_TI_TITLE, 5060
Trace rules checking, 3587

— U —

update_implication_detections, 3019

— V —

Verilog
    instance name, 5030

— W —

WGL_ADD_LASTX, 5069
WGL_ALT_BIDI, 5069
WGL_ALT_VECT_ANN, 5071
WGL_EDGE_STROBE, 5071
WGL_GROUP_PIN, 5074
WGL_INV_SC, 5075
WGL_NOMEASURE_CLOCK, 5076
WGL_ONE_ILLINOIS, 5076
WGL_PATTERN_NAME, 5077
WGL_TRIM_LEADING_P, 5077
WGL_TRIM_PULSE, 5077
WGL_VECTOR_ANN, 5078
WGL_VERG_ESC, 5079
WGL_VTRAN_PADSC, 5079

write_design, 3040
write_edt_files, 3059
write_failures, 3074
write_fault_sites, 3079
write_faults, 3083
write_flat_model, 3096
write_history, 3103
write_loops, 3103
write_modelfile, 3108
write_paths, 3109
write_patterns, 3109
write_procedure_testbench, 3144
write_procfile, 3148

Third-Party Information

Details on open source and third-party software that may be included with this product are available in the *<your_software_installation_location>/legal* directory.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.